NTNU
Det skapende universitet

Android-basert biomedisinsk søk

Eric Nordvik

Master i informatikk
Innlevert: September 2012
Hovedveileder: Herindrasana Ramampiaro, IDI

Norges teknisk-naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

Android based biomedical search client

Eric Nordvik

September 1, 2012

Abstract

The work in this report reflects on the choices to make when deciding on which approach is best suited to port a web-based biomedical search engine to mobile devices with small screens. It summarizes some of the valid options which exists with current technology and tries to give an overview of what considerations one need to take in this scenario. The focus is on how the BioTracer search engine could be implemented for Android devices and varies from implementing a native application via cross platform frameworks to scaling the web version to adapt to smaller screens.

Contents

Chapter 1

Background

This thesis is based on the current implementation of BioTracer and how to implement it as a separate application optimized for mobile devices with smaller screens than tablets and desktop computers.

BioTracer is a biomedical search engine developed by Heri Ramampiaro and Chen Li[17]. It is an effective search engine to support bio engineers with relevant articles based on their information need in their field of work. In its current state it is a web-based search engine which has a single input field for the users to type their query. Search results are presented below the input field with a title and an abstract of hopefully relevant articles. It also has an estimation of the relevance factor and the publication year.

To the right of the search results, another search result is displayed. These are suggestions to the user on related queries based on the user input. The user can click on any of these to bring up search results based on the related query. It is also possible to insert the selected related query into the input field next to the current query by clicking one of three icons in the related query list.

Working on desktop computers or laptops and even tablet computers, the current implementation of BioTracer works very well with the screens having lots of space to show the desired functions, results, and white-space needed to make the user comfortable using the tool. However, using the same tool on a mobile device with the same layout and same functionality does not invite to the same smooth interaction.

As we have seen in recent years, with the introduction of touch based small screen devices starting with the iPhone in 2007 and Android devices hitting the audience in 2009, the smart phone market has exploded, increasing the density of small touch based screens in a way that almost "everyone" has a micro computer in their pockets at any time.

Based on the above information, the need to implement a separate version of BioTracer targeted for small screen devices has emerged. A small screen implementation need to be efficient enough

for the users to use it in their natural environment and aid them without the need of using the desktop browser implementation. There are a range of factors coming into play when designing and implementing for small screen devices. This thesis will compare web based solutions with their small screen counterparts and point out what makes them separate and how the same functionality can be presented in different ways based on their screen size differences.

There are many considerations to make when implementing applications for mobile devices. The screen real-estate is very expensive as the physical size of the device has limited room for nice-to-have features. Font size, color use, shortcut links need to be well thought through when designing such applications. Also, we need to make sure that the main functionality is preserved in all versions of the product. If main user features included in the web based version are left out in the mobile and small screen versions, users have no need for these new versions and are likely to return to the web based version to serve their needs.

Then, what are the core features of BioTracer and how should these be implemented in a small screen version? How can we make users choose the mobile version when the situation calls for it, rather than taking their time to find a computer and using the web based version?

Chapter 2

Current situation and possibilities

2.1 State-of-the-art

2.1.1 Current search engines and their implementations.

Google[11] is of course the main reference search engine in todays modern web. Its growth and high adoption by most users throughout the world makes it the company that sets the standard in the search world. It's web search page is known for its simplicity as well as its fast and relevant result pages.

The format of the Google search page is as simple as a single search field and the Google logo. Typing in the search field immediately brings up suggestions as a list directly below the search field floating above the content. When a search is being submitted either by clicking the search button or by choosing one of the suggestions, the user is forwarded to a result page with the ten most relevant results. The results are displayed as a list with title, date and description for each result. In addition there is a menu on the left-hand side of the screen where the user can choose to narrow her search based on the type of results she is looking for, e.g. images, maps or videos.

This is now also the case for the mobile version. Due to the emergence of modern smart phones with fairly large screens, the mobile search page can be displayed in nearly the same format as the web version. Searching for phrases gives you related content as well as direct text results. Its speed is limited to the capacity of the device's hardware, and also the current connection of the device, be it mobile network or wifi.

2.1.2 Other applications and how they achieve the best possible user interaction and results.

PubMed[12] is an alternative closer to BioTracer than Google. However, its interface mimics that of Google, with a simple search field displaying search results as a list of links below the search field. The results display the full title of the biomedical articles related to the search phrase, listing the authors and publication year directly below each title. To the left of the result list a number of filter options are given, ranging from publication dates to types of articles. On the right-hand side you can see a section with pictures corresponding to your search in addition to related articles with the current search query present in the title or in the full text versions of the articles.

In its mobile version, which is a pure mobile website, the only thing PubMed have left from the web is the search field and the list of search results. One even have to click a "next"-button to view more results after the list of the 10 first results. There is made no room for images or related articles, and only a small subset of the filters are available. Of course, users are still able to find the articles they are looking for, but in my opinion this stripped down version of the PubMed search engine could have been done better for mobile devices. Even though the options that are not available on the mobile version is not the core features, it could have been part of the solution in a number of ways. Either as hidden menu options or elements minimized where the user can select to show them when needed.

2.2 Technology descriptions

2.2.1 HTML 5, Javascript and Cascading Style Sheets

HTML 5 is the latest standard of the Hypertext Markup Language (HTML) provided by the World Wide Web Consortium, W3C[22]. The standard defines which elements are to be supported when writing and parsing documents for display on the world wide web.

Javascript (JS)[25] is a scripting language developed by Netscape with support for objects and functions. Currently there are numerous libraries built using JS, e.g. Mootools and JQuery. These libraries makes it easy to use both visual effects and more complex tasks like asynchronous calls with out refreshing the entire web site.

Cascading Style Sheets (CSS)[23] is a language to describe the look and formatting of a web site. It has a number of options the developer can take advantage of, like positioning, background color, text attributes and so forth. By using e.g. gradients, transparency and rounded corners one can get the look of almost any graphical design without the need of actual graphic images

like JPEG or PNG cut into the correct sizes. It also ensures a more dynamic layout fitting the screen of any device the site is meant for.

Later years has seen the development of very dynamic and interactive web sites by the use of HTML, Javascript and CSS in interaction with each other. It paves the way for rich content and endless interaction possibilities making it a "product" worthy of competing with desktop or mobile applications.

2.2.2 Responsive web design

Responsive Web Design (RWD) is an approach to developing web site using a virtual grid system and CSS to make the content of a site float and resize according to the current screen the user is browsing through. The need for panning and resizing is minimized even on smaller screens because the content is "pushed" down and left when the screen is narrower than the actual page size. Also, elements like graphics, illustrations, videos and other components that are not needed to fulfill the needs for the users can be left out of smaller screens, only displaying the most important parts of the web site.

Lets take the website of NTNUs Technology Transfer office, tto.ntnu.no, as an example. They have recently had a major revamp of their website which adheres to the RWD principles. Figure 2.1 shows the full web page, figure 2.2 show the web page on a mobile phone (in this case a Samsung Galaxy Nexus using the Google Chrome browser), and figure 2.3 shows the web page on the same mobile phone only now in landscape mode.

As one can see, when the web site is visited in a desktop browser there is a search field in the top right corner, the menu is spread across the top of the page and there is a nice illustration taking up the majority of the site. Below this we can see three images or sections which brings the users to the next part of the site according to choice.

Stepping forward to the mobile site displayed on a phone in portrait orientation we see that the search field has disappeared, the menu items are floated below each other where it fits and the nice illustration is gone. The main part left is the three sections left for user selection. Looking at the landscape version we can see that the information visible is practically the same as the portrait version of the mobile site, only with more space horizontally for both the menu and the sections. Obviously the main focus of this web site are the three different sections in addition to the main menu, since the mobile version do not show the search field or the illustration.

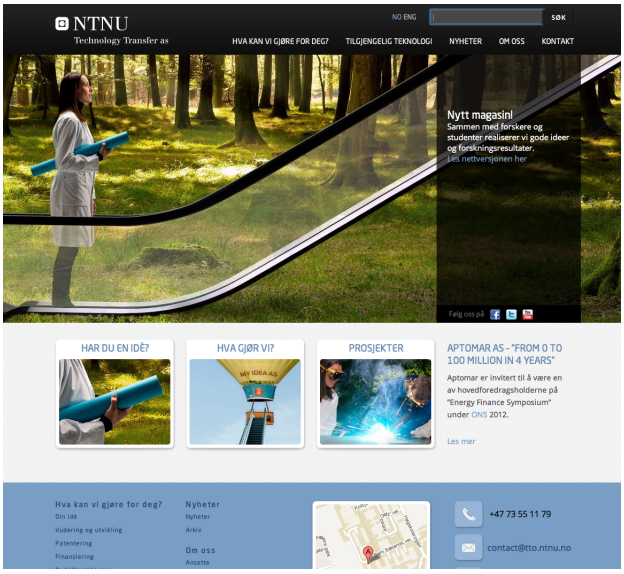

Figure 2.1: TTO web site

Figure 2.2: TTO mobile site

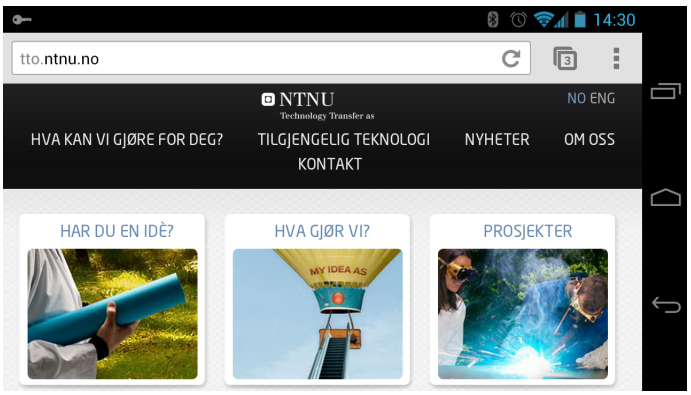

Figure 2.3: TTO mobile site landscape

2.3 Different approaches on technology

2.3.1 Android and iOS

Apple[3] introduced the iPhone in 2007[24]. It came with the iOS operating system specifically fitted for the small device the iPhone is. It obviously took its inspiration from the well known and successful Mac OS X[4], and with Apple's close integration between hardware and software as has always been the case, it was an immediate success. The iPhone was embraced by end users and developers. A year after the launch of the iPhone, the App Store was opened providing developers the means necessary to make their applications widely available to their end users, either as a paid or a free application. At it's launch day, the App Store contained 500 applications, a number that has increased at an explosive rate counting over 500.000 today[2].

According to Statista, the iPhone has been sold in over 240 million units up until Q3 2012[19] making it the most important product for Apple. Apple actually made a market share of 26.6 percent of the U.S. smartphone market in 2011. Obviously such a success has made other companies want to follow in Apple's footsteps. Recent years has seen different players in the field trying to make their own success, Google being the strongest and most successful competitor with it's open source operating system, Android[9].

Unlike iOS, Android is not tied to one specific hardware vendor. While Apple has control of both the hardware and the software running on the device, Google makes its OS available to hardware vendors to implement on their devices. A compatibility test suite[16] is made available for producers to test that their devices are compatible with the Android OS. This, in turn has made the fragmentation of Android vast and there are now over 1600 different devices running Android.

Of course, the fragmentation of Android means that the application developers face challenges providing consistent application user experiences across devices. This has also been some of the argumentation against Android, whereas the iOS has a very limited range of devices.

The user interface battle is also a factor. The iOS development kit gives a lot of standard graphical components which the developers are encouraged to use to give the users a common reference to equal actions across applications. Android has also had ready-made common graphics, however these have only been available and not something Google has expected developers to be using. The last year has seen a change of direction from the Android team at Google. With the release of the 4.0 version of Android, commonly known as Ice Cream Sandwich (ICS), along followed design guidelines with a stencils package Google hopes many developers will use. It is easy to believe that this is a choice Google made because they see the advantages consistency across applications gives as to what the users expects.

2.3.2 HTML5 advantages

But what about HTML 5 and it's wide adoption across the web? Most companies today have their own website and more and more of these sites are bringing the power of HTML 5, modern Javascript and Cascading Style Sheets (CSS). We've even gotten to the point where responsive web design is gaining popularity.

From a historical perspective, the trend has been that companies developed a website to promote their products and/or services. The focus was mainly on providing information to desktop computer users. Later on, mobile phones with WAP[26] functionality became popular, meaning users could browse the web on stripped-down versions of company or private sites. This meant at least two versions of the web site needed to be maintained by the site owners.

After some years, mainly around the time the iPhone was introduced, suddenly "everyone" had a browsing device with them at all times making the demand of being able to browse the web at some "mid-level" between the desktop version of the web site and the mobile version. This made way for a third version of the web site. You do not want to exclude the users still using the WAP version of your site by changing the mobile site to a touch site.

This is where responsive web design comes in handy. By structuring your web site according to the principals of responsive web design, your desktop web site degrades and fits automatically into the limited space of the browsers of e.g. an iPhone or a Samsung Galaxy SII. Modern smart phones comes with powerful browsers as Safari, Firefox and Chrome which supports technologies like Javascript and CSS, making the user experience similar to those of desktop computer browsers.

2.3.3 Cross platform application frameworks

In the recent years following the launch of iOS and Android, many cross platform application frameworks has gained popularity in the market. Companies like PhoneGap[15], Appcelerator[1],

Senscha[18] and Appspresso[5], among others, are all making developer frameworks meant to make it easy for web or application developers to port their website or application to different platforms using the current code the developers have. Like Appcelerator states: "Create iOS, Android, and mobile web apps from a single code base".

Most of these frameworks actually supports a whole range of native APIs making it more integrated with the platform it is running on. To a lot of companies and single developers this sounds like the right way to go these days. The obvious thing to believe is that if you make your site or product available to all major platforms like the web, mobile web, Android and iOS, your success is guaranteed.

It is a truth with some modifications. For the most part it is only the native APIs the framework is able to reach through an extended version of the built in browser for each platform. This could, however, be a fair amount of APIs. One can get access to camera, device info, sensors and much more. Nevertheless, the application lives within a WebView or a browser and does not get the full benefit of all the optimized view structures like e.g. lists, fragments, and viewpagers. Take lists on Android as an example. The lists are optimized by reusing the view objects that are no longer in the viewport when new view objects are requested. That way you do not need a thousand view objects for a list of thousand items, it is sufficient with the ones currently visible in addition to a couple above or below the viewport.

If one were to start off today with some new product or service, it might sound like a good idea to start of with a cross platform framework. You get to write your code once and run it "everywhere". The choice to be made is mostly strategic rather than technological. Developers tend to adopt to new technologies once they are familiar with some framework or programming language, so the extra effort needed to learn a new framework should normally not be the largest investment in the development process.

One drawback of using some of these frameworks is that it might load unnecessary plugins not needed for the current application. This might add significant size to the application footprint, and given the nature of sparse resources on mobile devices it could be of importance for the user. It is vital that developers working on the implementation has the proper knowledge on how to decide what plugins are needed.

2.3.4 Native application or not

When you are to choose between native or cross platform implementations there are some strategic decisions to be made. For instance, do one really need the product or service to be a native application with native look and feel of fellow native applications? If the application is not going to use OS graphics like buttons, sliders and checkboxes, that is one reason not to develop a native application. If the look and feel is something totally different, it might be wiser

to develop the application in HTML and its collaborators JS and CSS and then port it to all needed platforms using the framework of your choice. It might be wiser to let your developers learn one popular framework really well, and then stick with that to develop the application or the range of applications planned by the company.

However, the approach and requirements can be totally different varying from project to project. Some applications might live perfectly well within a browser and does not require other possibilities than the frameworks can provide. Other applications might need more optimized parts of the OS and the limitations of using a browser is probably not enough.

2.3.5 Types of input

The most commonly used way to provide input for a small screen touch based application is by using either a one-handed or two-handed grip around the device and touching the screen with the thumb(s) at the appropriate position. By using an Android device there are plenty of configuration options, including different keyboard layouts, as opposed to the closed state of the iOS devices.

Of course, Android devices also comes in versions with a hardware keyboard. Some users prefer these devices to software on-screen keyboards because they can feel that they are pressing the right key. Touch-based keyboards may be harder to touch correctly as there are little (haptic) or no feedback at all when pressing a key.

Based on the field of bio medical research one might consider choosing an optimal keyboard layout giving easy access to frequently used special characters. The keyboard should also be localized to English language, as most articles on the field are written in English. Obviously, hardware keyboards do not have the same possibility for customization.

There are numerous developers implementing all kinds of keyboard applications for Android devices. Some of these have specialized layouts, others include prediction algorithms and ways to learn the behavior of its users. Currently SwiftKey[21] is one of the most used third-party keyboards for Android devices. What is also interesting about SwiftKey, is that they offer a specialized keyboard for use in healthcare. The keyboard application is backed by several components including spelling correction, a prediction engine and dynamic learning. This means the keyboard will become more and more useful the more it is used.

A study made by Park and Han[14] tried to select a natural size of touch buttons for mobile devices when a one-handed thumb interaction was used. Figure 2.4 shows their findings in how the user perceived their touch point represented by the black dots and what the actual contact area was as the grey ellipses. The actual button is shown as the rectangles.

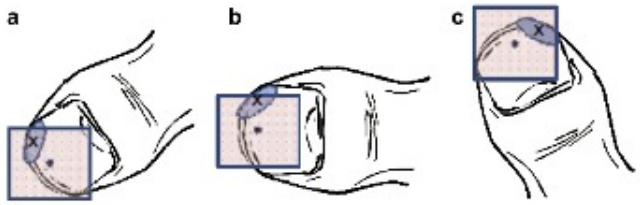

Figure 2.4: Thumb touch points

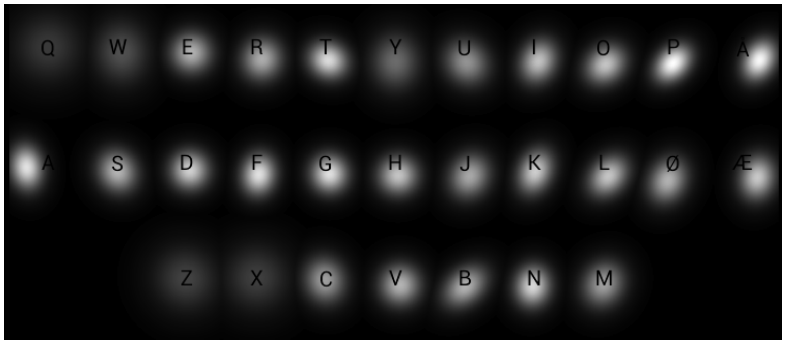

Figure 2.5: SwiftKey heat map

This point is important because of the way users perceive their touch inputs when using on-screen keyboards. In figure 2.5 I have included my own “heat map” from the SwiftKey application. This map is generated by SwiftKey based on my key presses along with what words I have typed and which I have or have not corrected afterwards. As you can see I am far off the actual keys when I am trying to hit the keys near the edges of the keyboard, like ’A’ and ’Æ’, but when I am using keys in the middle of the keyboard the key strokes are more accurate.

Chapter 3

Design

3.1 Assumptions

The web version of the BioTracer search engine is implemented using JSP and Servlets on top of a Java application. The application provides a search interface and sophisticated techniques to filter queries made by the user. A major effort has been put into the application to index bio medical articles and building on top of this is the main focus of this report. Since BioTracer requires a network connection to query the data sources it uses, the application requires one too. There is no need to integrate the whole backend service codebase from BioTracer to the Android application. Mainly, the Android application should implement the API provided by the BioTracer source code.

Therefore, the Android application should only need access to the search interface and make use of the same classes for performing searches as the web application does. This reduces the Android application to an interface to the BioTracer search engine. The focus will be on the user interface and which technology is best suited to create a small screen version of BioTracer.

Some of the design principles in the following sections are collected from Nilsson [13]. In his report he runs through different user interface design patterns and suggests where to use and where not to use them when designing mobile applications. Not all of the patterns fit our needs, but some of them are quite useful.

3.2 Requirements

The main purpose of the application is to make biomedical articles easily accessible on demand for the users. Hence, the application consists of three main features; search, a result set and a reader for the results.

When we focus on the search part of the application this is mostly done by having an input field that takes a keyboard as input source and provides a button or an action when the user presses the enter key on the keyboard. This action performs the request to get the search results and, in our case, sends a request to the BioTracer server to fetch the results related to the text in the input field.

Next we need to have a way of displaying the search results to the user. The most common way to do this is having a list of results shown below the search input field containing the titles of the resulting documents, accompanied by some sort of timestamp and a short summary of the contents of the articles. We can also have the result list show the authors of the articles to help the user find the correct link to press.

Finally we need to decide how the selected article should be displayed for the user. The articles often comes in the form of pdf documents which is supported natively by most Android devices. When the user clicks on an article we can open a dialog showing the supported applications installed on the device for reading pdf documents. Natively this dialog comes with a check box which enables the user to select a default application for this operation, making the next selection of an article one click less for the user.

In addition to these three main features, the web version of the BioTracer search engine displays a list of related search results. This is done to help the user find what she is looking for. Taking the limited screen real estate into account it can be troublesome to have the same feature available in the Android application. As this is a nice-to-have feature, we can choose to hide it as a default, but giving the user the ability to select whether it should be displayed or not.

3.2.1 Lightweight implementation

By choosing a native Android application we are left to implement a very lightweight package which only holds the necessary components needed to display the contents we need. Android uses a concept of providing resources which is optimized for each device density and screen size. No resources like images, stylesheets or source code needs to be downloaded through the web once the application is installed. The only data traffic involved in this type of application are the requests performed by the user and the response of results from the cloud.

The application can be streamlined to use very sparse resources when it comes to graphical components. Like HTML and CSS can provide gradients and images made at runtime, Android can use xml to define layouts, graphics and gradients. It also supports svg to create graphic at compile time.

We are also in total control of which plugins and libraries our application includes. From my point of view the approach of having to include what is necessary is easier to control than the framework approach having to exclude libraries one do not need.

3.2.2 Application components

3.2.2.1 The search interface

First and foremost the application needs to display a search field for the user input. Typically this is displayed on top of the screen or in the centre. When using an Android device without a hardware keyboard, the keyboard is displayed instantly when the search field is clicked, making the input field ready for user interaction. Of course, it is also possible to search via the hardware keyboard if the device is equipped with one.

There are two main alternatives for providing a search field. One is using a simple EditText view component which inherits from the TextView and the other is to use the more modern SearchView. The SearchView was introduced in Android API level 11 which is version 3.0 of the Android OS.

One of the main features of the 3.0 version of Android was the introduction of the ActionBar which sits on top of the applications and provides a single point of navigation and action for an application. The ActionBar is the main point of navigation in newer Android applications. When one look at applications made by Google, it is used everywhere possible. What is also the case is that the ActionBar is, as its name implies, the place for actions related to the application. That is, actions that are not part of the main application window, like selecting list items, play buttons and so on. Actions like search, options, settings, help and their likes fit nicely into the ActionBar. The more importance the action has, the more visible it should be to the user.

If we look at the YouTube application[10] as an example, its main functionality is to let users discover videos. Other than the logo and the current category title, the ActionBar in the YouTube application features a magnifying glass which is at least the Android standard icon for searching. Pressing the search icon brings up a search input field and the user is ready to explore the vast universe of videos. As Nilsson[13] points out, there is no need to start out making brand icons of your own if they are available in the OS, unless you are making your own branding. In this case we are showing the user where to go when he wishes to search and it makes sense to use the standard icon to make the user experience easier.

If we take the discovery based thought on to BioTracer we are on the right track. Users of BioTracer aims to seek information. A natural part of the ActionBar will be to have the search field fully expanded at startup. The default behavior of the SearchView is to display the current search phrase after the user has performed a search. This makes it easy for the user to modify the search as well as clear the search by using the default clear button visible in this state.

What more is that the SearchView comes with an auto-complete feature. For example, it can be configured to trigger the auto completion after x number of characters has been typed. This

means having the application search for articles based on the current content of the SearchView. The completion suggestions pops up below the SearchView in its own interface component, just like the experience is in the web based version of Google.

Looking at the OS distribution across devices as of August 2012[7], we can see that the majority of Android devices, approximately 60%, are running version 2.3.x. This means that the ActionBar, and subsequently the SearchView, is only natively available for about 19% of the available devices. As this project focuses on phones and not tablets which are the only devices running version 3.x, we are down to 16.7% of the devices.

To aid with considerations like these, Google has made available a support library which offers a back port support of newer version API components to older versions of Android[8] back to version 1.6. The ActionBar comes as a part of this library and we can easily include this part of the API even if we are targeting the application to 2.x devices.

3.2.2.2 Providing the search results

Second, the application needs to display the search results to the user. This will be in the form of a ListView as the layout used to hold the elements in the list. Each element will consist of several TextViews with larger font size for the title of each element. The task of the ListView is to display the elements and add more elements to the list when scrolling. As for the actual data in the list, a ListAdapter need to be set to the ListView. The adapter contains all data elements and is used by the ListView to determine which element is the next to be displayed in the list.

An interesting point was made by Sweeney and Crestani[20]. They compared search results and the size of the summary for each result item on different screen sizes and measured the success rate for users trying to gain information. What they found was that the size of the summary should not in general be selected according to the screen size. One should think that larger screens with much more space for text should try to use that advantage and show more of the summaries. They saw, however, through their experiments that shorter summaries worked better across all screen sizes. This is obviously an advantage for the project at hand, making BioTracer available for small screen devices.

When a search is performed, an asynchronous call is made to fetch the results from the server. In Android this is accomplished by using a CursorLoader. The ListAdapter uses a Cursor to hold its data, and a CursorLoader loads data either through requests to an internal ContentProvider on the current device or through a network request. When the data is loaded, the CursorLoader uses the callback method onLoadFinished to insert the new data into the current dataset. This way we will get a seamless user experience while scrolling through the search result, not needing to press some link to get more results.

3.2.2.3 Showing the details

The third part of the application is how we are to display information about the selected article when the user makes a selection. There are several alternatives to this feature. For instance, a click in the list on an article could make the application open a details page providing more information on the article. From there, the user can read more complete information like more authors, the complete abstract and so on. In addition, such a details page can include some sort of list of articles related to the current article.

Another alternative is to make the Android device search for the most suited application to read the selected article directly after the user has clicked in the list. Most Android devices comes with a built in reader for PDF documents, either Document Viewer or e.g. Adobe Reader. When the user selects an article, the first time this is done a dialog will appear listing the applications supporting the format of the document being requested. The dialog will be opened each time the user selects an article unless the user chooses to use one of the applications as the default viewer for subsequent selections.

3.2.2.4 Related queries vs search results

There need to be place for related queries in the user interface of the application. Given the limited amount of space on small screens, the information should be possible to minimize in case the user finds it to be less relevant at any point in time during the session.

One way this can be done is to show a tab on the right-hand side of the screen which is expanded when the user touches it or drags it into view. The right-hand side is selected because both the web based versions of BioTracer and PubMed uses this part of the user interface to display related queries and articles. Building on user interface consistency even across platforms is often a sensible choice both for the user and the developers.

3.2.3 Search history

Another advantage with writing a native Android application is the ability to include a means for saving previous queries directly within the application. This is done by creating a `SearchRecentSuggestionsProvider` and saving each query to the provider each time the user issues a query[6].

To make this work a searchable activity needs to be implemented. In addition a `ContentProvider` must be added which will take care of querying the local database of recent searches and presenting it to the searchable activity. Then, each time a query is performed, the query text is saved using the `saveRecentQuery` method of the `SearchRecentSuggestions` class.

Chapter 4

Implementation details

4.1 Classes:

`BioTracer extends ListActivity`

This is the main class of the application. It is the starting point where the `ActionBar` is displayed with a logo and the `SearchView` expanded for direct user interaction. It might also be wise to have some sort of "welcome" screen explaining the purpose of the application or a quick tour of the application.

The `onCreate` method of this class sets up the major parts of the application. This includes finding the list from the layout, initializing the adapter to hold result items and focusing on the search input field.

`ListAdapter extends ArrayAdapter<ListItem>`

The `ListAdapter` is to contain all search results. It is filled with `ListItem`s after the search has returned and by scrolling the list more items will be loaded automatically as long as there are more results to show. The `getView` method of the adapter will always try to reuse previously used `View`s. The Android `BaseAdapter` class has a built-in functionality that gives the `getView` method the current `View` to use for the item. Items that has gone out of the viewport while scrolling is made available for the new items that come into view.

`ListItem`

When the list of search results is filled, these are the items put in the adapter. It will contain a title, author names and a summary in separate TextViews. The summary will be maximum two lines long to keep the information limited but still helpful. In addition, each ListItem will contain an icon linked directly with the article. If the user clicks the icon, the article is opened, and if any other part of the ListItem is clicked, the detail page is opened.

`ArticleDetailsFragment extends Fragment`

The detail page will show an extended view of the article from the list item. It features an `ActionBar` with the title of the article on top and the complete abstract of the article below author names and publication information.

`SuggestionsProvider extends SearchRecentSuggestionsProvider`

The provider is responsible for saving recent searches, and is also the class that will respond to the search activity with suggestions when the user enters search text.

These classes should make up the main part of the application. There are in addition graphic files like images, logos and so on. Also, each layout is defined in its own XML-file composing the user interface of the application.

4.2 Benefits of using existing code

As BioTracer has been implemented in Java and Java Server Pages(JSP), the implementation of BioTracer for Android should come as a slight challenge. But knowing Java is still an advantage. Writing Android applications forces the developer to know the framework as well as syntax and the object-oriented aspect of Java in general. The life of an Android application is handled by the Dalvik Virtual Machine which is present on all Android devices. It controls the lifecycle states of each component and also handles memory usage for each application. Still, it is vital to be aware of the limitations of mobile phones regarding the hardware.

Even though Dalvik handles garbage collection of unused Java objects, it is important not to challenge the system. There are several ways to help the Dalvik VM let your application run smoothly. One of the most important lessons I have learned is that the Views that make up the user interface, and especially images, take up a lot of memory. Therefore, one should not have too many Views at the same time in the same screen. One can also benefit from reusing Views that have been on screen, but are currently out of the viewport. This way, the objects need not be constructed once again, and at the same time, unused objects do not remain in the memory.

Another issue is to always leave heavy tasks like computing or using I/O-resources such as files or network off of the main application thread. The thread running the application takes care of all user interface tasks, like drawing all elements on screen and handling actions from the user. If you are to call a network resource, this should always be performed in another thread.

Chapter 5

Conclusion

Although the web version of BioTracer can be used on small screen devices, its complex user interface does not invite to extensive use on a mobile phone. It would highly recommend the implementation of an optimized version of BioTracer for mobile devices. As the last years have shown, mobile devices in the forms of smart phones has come to stay. Touch based devices makes the web and other applications accessible at any time, anywhere. Some applications, like BioTracer, do require an internet connection to work, but in general, most devices are connected at all times.

The question is whether to implement this as a native Android application, use cross platform frameworks or make an effort to make the current web version adhere to the principles of responsive web design.

First of all, as I have outlined in this report, a native Android application can be implemented with a fairly small footprint. It will still take advantage of the possibilities in the Android OS making use of concepts like lists with automatic loading of more results, opening articles in the most suited application and so on. The application will be responsive and adhere to the design guidelines presented by Google which will make users familiar with Android instantly familiar with the BioTracer application. I think Android users would like a BioTracer Android application. As an Android user myself, I am enjoying using applications that follow the standard design guidelines by Google and use the newest APIs.

If we look to the concept of cross platform frameworks, this too will provide access to native APIs in each platform supported. Most concepts mentioned in this report will be available to the application by using one of the common frameworks. We need to be aware of the dangers of putting to much plugins into the application making it far larger than it really should be. However, the benefits of getting the application ported to the to major platforms, iOS and Android, might be a more important goal than to make the application size small.

I was previously an iPhone user and still use products like the iPad and Macs. There are a number of applications I have previously used on the iOS platform that I currently use on

Android. The developers tend to choose differently and I can not answer to the strategic decisions that has been made. In my opinion, the applications that have been ported to native Android applications are a much better user experience than applications written as web applications and probably ported using some cross-platform framework.

Bibliography

[1] Appcelerator. http://www.appcelerator.com/platform, 2012.

[2] Apple. Apps in the app store: http://www.apple.com/iphone/from-the-app-store/, August 2012.

[3] Apple. http://www.apple.com/, 2012.

[4] Apple. http://www.apple.com/osx/, 2012.

[5] Appspresso. http://appspresso.com/, 2012.

[6] Google. Adding recent query suggestions: http://developer.android.com/guide/topics/search/adding-recent-query-suggestions.html.

[7] Google. Android platform versions: http://developer.android.com/about/dashboards/index.html, 2012.

[8] Google. Android support library: http://developer.android.com/tools/extras/support-library.html, 2012.

[9] Google. http://developer.android.com/index.html, 2012.

[10] Google. https://play.google.com/store/apps/details?id=com.google.android.youtube, 2012.

[11] Google. https://www.google.com/, 2012.

[12] NCBI. http://www.ncbi.nlm.nih.gov/pubmed/, 2012.

[13] Erik G. Nilsson. Design patterns for user interface for mobile applications. *Advances in Engineering Software*, 40:1318–1328, 2009.

[14] Yong S. Park and Sung H. Han. One-handed thumb interaction of mobile devices from the input accuracy perspective. *International Journal of Industrial Ergonomics*, 40:746–756, 2010.

[15] PhoneGap. http://phonegap.com/, 2012.

[16] Android Open Source Project. http://source.android.com/compatibility/cts-intro.html, 2012.

[17] Heri Ramampiaro and Chen Li. Supporting biomedical information retrieval: The biotracer approach. *Transactions on Large-Scale Data- and Knowledge-Centered Systems 4*, 6990:73–94, 2011.

[18] Sencha. Sencha touch: http://www.sencha.com/products/touch/, 2012.

[19] Statista. Iphone sales from q3 2007 to q3 2012: http://www.statista.com/statistics/12743/worldwide-apple-iphone-sales-since-3rd-quarter-2007/, July 2012.

[20] Simon Sweeney and Fabio Crestani. Effective search results summary size and device screen size: Is there a relationship? *Information Processing and Management*, 42, 2006.

[21] SwiftKey. Swiftkey healthcare: http://www.swiftkey.net/healthcare, 2012.

[22] W3C. http://www.w3.org/html/wg/, 2012.

[23] W3C. http://www.w3.org/style/css/, 2012.

[24] Wikipedia. Apple iphone: http://en.wikipedia.org/wiki/index.html?curid=8841749.

[25] Wikipedia. http://en.wikipedia.org/wiki/javascript, 2012.

[26] Wikipedia. http://en.wikipedia.org/wiki/wireless-application-protocol, 2012.