The Leaf project

A first application

FEDERICO ENNI

KTH Information and
Communication Technology

Degree project in
Communication Systems
Second level, 30.0 HEC
Stockholm, Sweden

The Leaf project: a first application

Federico Enni, enni@kth.se

Masters Thesis - 2011-11-03

Examiner
Professor Gerald Q. Maguire Jr.

Supervisor
Professor Gerald Q. Maguire Jr.

KTH

Abstract

Today large publishers are developing platforms via which their content, such as magazines, books, and newspapers, are distributed based upon the well established pattern of ‘pay-per-content’, via a multitude of mobile applications. Despite of the recent flourishing market of subscription possibilities, users currently have to buy single items of content at an average expenditure that is approximately equal to the cost of the printed media, because publishers are still investing in printing hardcover versions of their publications.

Furthermore currently digital documents are mainly “scanned” versions of the printed ones, an unattractive format which does not exploit the potential of digital publishing. This format is rendered by the device in a way that does not permit the user to take real advantages of a digital environment, making the e-reading experience something that has no real added values, compared to regular printed publications.

In order to provide to people an improved experience in terms of both accessing and enjoying their favorite material, the Leaf project aims to create a platform in which both publishers and readers can take advantage of an innovative business model and advanced technological solutions.

As part of the Leaf project, the main objective in this subproject is the realization of a client that can access a set of online contents and offer these to the reader, by streaming content rather than requiring that the complete object should be downloaded in order to be rendered by the device. The access to the material should follow modern security standards, including data encryption, in order to prevent unauthorized use of the content. Furthermore, the client should present the content in an innovative way and allow the development of a mobile platform that could be supported based upon advertising, with an approach based on using information about the user (i.e. favorite places or profile details) as well as content-based advertising.

In addition to the client application, the development of this subproject includes the development of a series of server-side utilities for the uploading and elaboration of publications on the server infrastructure.

The document first reviews e-reading systems, focusing on the ePub file definition and e-reading solutions currently in use. The report then describes the Android OS development environment. The document then explains the implementation for both client and server side applications, giving a detailed overview of the chosen strategies and of the applications’ structure. Finally the report concludes with a list of the achieved goals, a discussion on the project’s limitations and then an exploration on what is the future for this client application.

Sammanfattning

Idag utvecklar många utgivare plattformar vilkas innehåll, såsom tidningar, böcker och veckotidningar distribueras via ett flertal mobila applikationer enligt den väl etablerade ”pay-per-content” modellen.

Trots den blommande nya marknaden som ger möjligheten till prenumeration, får konsumenterna fortfarande köpa enstaka publiceringar som ungefär kostar lika mycket som de tryckta media. Detta för att utgivarna fortfarande investerar i den inbundna versionen av deras publikationer.

Dessutom är de digitala dokumenten främst en “skannade” version av det tryckta, en oattraktiv format som inte utnyttjar det digitala formatets potentialitet. Denna format utförs av enheter på ett sätt som inte tillåter användare ta fördel av digitala möjligheter. Därmed, jämfört med tryckt material, erhåller läsningsupplevelsen inte något mervärde.

För att ge en bättre upplevelse, både när det gäller att få tillgång och att ”njuta” av ens favorit material, Leaf projektet vill skapa en plattform i vilken båda utgivare och läsare kan ta fördel av en innovativ businessmodell och avancerade tekniska lösningar.

Huvudmålet i detta delprojekt, som tillhör huvudprojektet Leaf, är realisering av en client applikation som har tillgång till online material. Detta material erbjuds till läsaren via streaming istället för att hela nedladdningen ska ske innan kundsenheten ska kunna börja använda det. Tillgången till materialet ska ske enligt moderna säkerhetsstandarder, datakryptering inbegripen, för att undvika otillåten användning av detta material. Dessutom ska client applikationen presentera innehållet på ett innovativ sätt och tillåta utvecklingen av en mobil plattform som ska kunna stödjas utav reklam. Tillvägagångssättet för att genomföra reklam ska baseras på användningen av information om användaren (till exempel favorita platser eller profildetaljer) likväl som innehållbaserad reklam.

Förutom client applikationen inkluderar detta projekt också utvecklingen av en serie server-side verktig för uppladdningen och utarbetandet av publikationer på serverns infrastruktur.

Denna rapport ger först en recension av e-reading system, med fokus på ePub file definition samt e-reading lösningar som i dagsläget är använda. Efteråt förklarar rapporten implementationen för både client och server side applikationer genom att ge en detaljerad översikt av de valda strategierna samt applikationernas struktur. Slutligen avslutas rapporten med en lista över de mål som har nåtts, en diskussion om projektets begränsningar och en prospektering på denna client applikationens framtid.

Table of contents

List of Figures vi
List of Tables vii
List of Acronyms and Abbreviations viii
1. Introduction 1
1.1 Current publishing infrastructures 1
1.2 The e-reading history 1
1.3 The Leaf project 2
1.4 Client application 3
1.4.1 The Android OS as main interface for the Leaf platform 4
2. Electronic publications 6
2.1 Overview of electronic document formats 6
2.2 The ePub standard 7
2.2.1 Open Publication Structure 7
2.2.2 Open Packaging Format 8
2.2.3 OEBPS Container Format 9
2.2.4 ePub 3.0 draft specification 9
2.2.5 ePub 2.0/3.0 as the standard for the Leaf platform 10
2.3 Existing platforms 10
2.3.1 Evaluation of existing e-readers softwares and features 12
3. Android OS and SDK environment 14
3.1 Android OS for mobile devices 14
3.2 The Android SDK 15
3.2.1 Technical overview on the Android SDK 15
3.2.2 Android project's structure, resources, and R.java 15
3.2.3 Android SDK Layouts 17
3.2.4 Android SDK source code 18
3.2.5 Android SDK project Manifest 19
3.2.6 Code compilation and the .apk packaging format 21

4. Client application components **22**

4.1 Connection to the server and authentication 23

4.1.1 Authentication via authActivity.class to manage users ***24***

4.1.2 authManager.class ***25***

4.2 Data presentation 26

4.2.1 Data synchronization and libraryActivity.class ***26***

4.2.2 Retrieving information with dbConnector.class ***27***

4.3 E-Reader software 28

4.3.1 Webkit engine ***28***

4.3.2 readingActivity.class ***29***

4.3.3 Javascript interface ***30***

4.3.4 jsTools.js ***30***

4.4 Streaming buffer and cache 31

4.5 Graphical User Interface 31

4.5.1 Support for multiple screen sizes and resolutions ***32***

4.5.2 Page number handling vs navigation controls ***33***

4.5.3 Transitions ***34***

4.5.4 User controls ***34***

5. Server side applications **36**

5.1 Web interface 37

5.2 Java utilities 38

5.2.1 chapter.class ***38***

5.2.2 book.class ***38***

5.2.3 dbInformation.class ***39***

5.2.4 summaryBuilder.class ***40***

5.3 User authentication interface and Cookies handling 41

5.3.1 User authentication interface with loginApp.php ***41***

5.3.2 Cookies handling and the auth_memCookie module ***42***

Analysis of the results **43**

6.1 Implementation decisions 43

6.2 Empirical results 45

6.2.1 Analysis of the empirical results ***46***

6.3 Comparison with existing results 48

7. Conclusions **50**

7.1 Achieved goals 50

7.2 Limitations of the current solution 51

7.2.1 Authorization system's limitations ***51***

7.3 Future work 52

7.3.1 Future work on the client application ***52***

7.3.2 Future technologies ***54***

References **56**

Appendixes **60**

Appendix A - Uploading a book through the web interface 60

Appendix B - Sample "accessCookie" 62

Appendix C - Apache configuration for the auth_memCookie 63

List of Figures

Figure 1 - Sample strings.xml file 17

Figure 2 - Sample of a TabsLayout 18

Figure 3 - Sample AndroidManifest.xml file 20

Figure 4 - A sample of <permissions> tag 20

Figure 5 - *Overall functioning of client application* 23

Figure 6 - *User Authentication* 25

Figure 7 - *Basic functioning of libraryActivity* 26

Figure 8 - *Basic functioning of readingActivity* 30

Figure 9 - *Screen sizes supported in the Leaf client* 33

Figure 10 - *Server side applications when documents are uploaded* 36

Figure 11 - *Decisional path for the design and development of the client application* 44

Figure 12 - *Relation between WiFi and 3G in the application* ...47

Figure 13 - *Screenshot of the document inserting form* 60

Figure 14 - *Code for insertingBook.html* 61

Figure 15 - *Code for insertedBook.html* 61

Figure 16 - *Data present in the accessCookie* 62

Figure 17 - *Extract of the server's configuration for the auth_memCookie module* 63

List of Tables

Table 1 - *Comparison of features in different e-reader clients*................. *13*

Table 2 - *Resolution formats scheme* *16*

Table 3 - *Application's empirical data on components' speed tests* *46*

Table 4 - *Comparison of features in different e-reader clients, revised* . *48*

List of Acronyms and Abbreviations

Android environment

APK	Android PacKage
JIT	Just In Time (Code Compilation)
OS	Operating System
SDK	Software Development Kit

Communication Protocols and networking acronyms

AVFSD	A Virtual File System
DN	Distinguished Name
HTTPS	HyperText Transfer Protocol Secure
LDAP	Lightweight Directory Access Protocol
SSL	Secure Socket Layer
TLS	Transport Layer Security
URL	Uniform Resource Locator

E-publications

EPub	Electronic Publication
NCX	Navigation Control for XML
OPS	Open Publication Structure
OPF	Open Packaging Format
OCF	OEBPS/Open Container Format
SVG	Scalable Vector Graphic

Programming environment

CSS	Cascading Style Sheet
GUI	Graphical User Interface
IDE	Integrated development Environment
PHP	Personal Home Page/Hypertext PreProcessor
UID	Unique IDentifier
XML	eXtensible Markup Language
(X)HTML	(eXtensible) HyperText Markup Language

1. Introduction

1.1 Current publishing infrastructures

Information distribution systems have evolved dramatically, due to the development of digital communication systems. Looking at the change in how news is distributed by publishers and received by the readers just a decade ago in contrast to now, it is evident that technological innovation has driven the information world into a new era, where the delay between writers and readers is often imperceptible and the time it takes to write a line is longer than the time required to distribute the complete article to millions of readers worldwide.

Personal computers and high-speed Internet connections play a key role in this process, and as these devices evolve into something more personal, information channels will follow the same evolutionary path.

Currently the global information market is trying to penetrate ever more deeply into immediate distribution, thanks to the popularity of personal devices such as advanced smart phones, e-book readers, and tablets. Due to the wide spread adoption of publishing-oriented software, newspapers, magazines, and books are now accessible “on-the-go”. Additionally, access to digital material via these devices allows content to be dynamic, rather than being restricted to static content and it also allows the content itself to refer to other content which the “reader” can see or hear - either by choice or involuntarily .

The mobile market represents a battle field in which all the major technology firms are currently fighting, and it is experiencing extraordinary growth. At the same time the publishing industry is trying to adapt its infrastructure in order to use mobile devices and networks as the main medium to reach readers. However, the main problem to solve is how to change peoples well engrained habits regarding their reading.

The Leaf project is an idea being carried out by KTH Masters students Diego Botero, Federico Enni, George Khalil, and Sebastian Galiano. The team has identified several potential areas that are still not well developed by the publishing and distribution industry. The project’s vision is that by addressing these areas we can present a service that is attractive for peoples who are still skeptic about changing how they enjoy their favorite reading material. Further details of the project are given in section 1.3 starting on page 2.

1.2 The e-reading history

The first prototypes of digital books were developed during the early nineteen seventies, thanks to the *Gutenberg Project[1]*, which started producing electronic documents for the public. Despite the initial indifference during the following twenty years, when industrial actors were focused on the development of the personal computer and Internet, electronic books started arousing public interest in the late nineteen nineties, due to the development of universal document formats such as *Adobe’s Portable Document Format (PDF)*. In the same period several industrial actors introduced into the

market the first models of eBook readers, such as SoftBook. However, these failed to be successful because of the limited availability of contents, the lack of a fast and user-friendly distribution infrastructure, and the high costs of both content and readers.

With the continued development of information and communication systems and the mobile revolution, interest in digital publishing/reading solutions have come back in vogue, attracting all of the actors involved in the information distribution chain. This enables writers to access a broader public, while readers can take advantage of a larger universe of information sources. Today publishers are trying to evolve the distribution of contents to its next stage, a process that implies a major change in their role in the digital world.

Despite the fact that eBooks and eMagazines distribution is entering a transitional phase, the Web 2.0 represents nowadays the main alternative to traditional reading and information sources. Thanks to its multitude of websites, blogs, forums, and social networks, which are often easy to use and free to access for everyone, a large number of users are increasingly getting their content via Web 2.0 “publishers” more than from traditional publishers.

A good example of a company that is already working in digital content distribution is OverDrive Inc., that provides a multitude of services such as books catalogue and distribution platforms to many libraries, institutions, and retailers. Other companies involved in the “digitalization” of content are Digital Media Initiatives, Aptara Corporation, TexTech and Innodata Isogen.

1.3 The Leaf project

In this dynamic and fragmented market, the publishing industry is trying to apply its strategies and models in this new technological environment. This industry’s primary objective is a “*soft*” migration from paper to digital media. This migration, which is expected to be slow because of users‘ skepticism regarding the merits of changing something as habitual as reading a book or a magazine, has lead to the publication and distribution of contents through virtual bookshops, in the case of books, and specific mobile applications for individual magazines.

The current eBooks industry is based on the development of text that will be presented in a way which tries to be as similar as possible to the paper version of the same text, so that the user has the possibility to browse paper-like pages and reading the text in a linear fashion. Today books can be purchased from several online services, at prices very near to bookshop prices of books. This happens because of many factors, such as the lack of a clear income division pattern for a digital book (the percentages currently used are often the ones used in paper publications, where publishing and physical transport have a big influence). This and other elements are clear signals that publishers still do not invest enough of their energies in the digital publications market, as they look at this conversion as a potential economical risk. However, it is necessary to mention other important efforts made to help the growth of digital publications, as the e-libraries are doing. Even if they offer loans services as they used to do with printed books, publishers such as the british HarperCollins or Simon & Schuster are now trying to protect themselves by applying inhibitions and time limits to these loans.

Regarding magazines, comics, and journals, the largest publishers are proposing a number of device-centric applications which offer readers advanced reading experiences, augmented contents, and integrated multimedia. However, these applications need to be individually downloaded and installed. Readers need to purchase as many applications as there are magazines they are interested in reading, and then to buy subscriptions or pay a single issue price for every issue they would like to read.

According to data extracted from many sources *[2] [3] [4]*, such the Newspapers Association of America statistics of 2010, people from occidental countries increasingly prefer reading news and articles from websites and blogs more than publications, either paper or digital ones, because of the ease of access, social-sharing possibilities, zero costs, and many other reasons.

Our initial analysis of the current industry situation and market, led to initiation of the Leaf project. This project's vision is to provide people with a way to enjoy their favorite reading material while taking advantage of the benefits offered by the digital era, such as integrated multimedia and interactive elements. In addition, innovative business models may offer the possibility of offering contents to people without asking the users to pay for each individual accessed resource, be this a book or magazine issue. The primary goal of this project is to create a platform on which writers and publishers can distribute their content and by which users can easily access a large variety and collection of magazines, books, while enjoying comics, and innovative forms of readings on their choice of mobile device.

On the business side, the project will examine alternative and innovative ways of distributing contents, generating for the authors, editors, and publishers a profitable revenue stream. One idea is to enable different types of mobile advertisements and the creation of partnerships with "hosting companies" that will manage free distribution spots, offering users different ways to access content via this proposed platform.

Among the technical issues that the project must address, is the creation of publishing solutions to provide authors and publishers with an easy means to edit and upload contents. We propose to do this using a mobile device-centric application that will enable content producers to distribute new material. In addition, a client application will allow readers to access the published content. The distribution infrastructure should be designed to safely and efficiently store and distribute contents.

1.4 Client application

The main objective of this specific master thesis project is the design, development, and evaluation of a client application for the Leaf project. The client software, which in this thesis project will be developed as an Android OS application, connects to the server and presents to the user an overview of the available contents. This presentation may depend on the user's profile, that in the first phase will be unique for each user, even if in the future a user may have several profiles. After selecting the desired contents, the user should be able to (nearly immediately) begin enjoying the content, leafing through pages, and navigating through the entire e-document — as occurs after downloading with existing e-reader applications. The primary benefit of this new client will be the advantage of immediate access and augmented interactive content.

In order to support interactive content, the application should support HTML5[6] and CSS3[7] languages, especially for <*canvas*> elements. For the same reason, native Javascript[8] as well as major frameworks (such as JQuery[9] and Prototype[10]) must be fully supported. The chosen format that we have decided to utilize is the digital document format ePub 3.0[11]. Details of this format will be presented in section 2.2, starting on page 6. The reasons for selecting this format are also explained during section 2.2.

An intuitive and user-friendly graphical user interface (GUI) will be designed. The client must support bookmarking, appearance controls, and review & manipulation of recent history. Finally, the possibility of adding plugins to improve the user's experience will be evaluated.

During the development of the first version of this application, contents will be accessible only when the user is connected to the Internet and data will be transferred to the user's device in a streaming fashion. This implies the use of a buffer (in memory) to store pages to be read and a temporary cache to keep recently visited content in order to quickly load this content. This strategy will allow us to achieve two important results: we limit the amount of contents that the users have in their devices, because users do not need to download entire documents to their devices; and we allow the user to access content from different devices with the same account. In a second stage of the project the option of adding an offline mode will be evaluated.

Another important part of the software, the development of which will depend on commercial strategies still to be defined by my collaborators, will concern retrieving data about the device and the user, and sending these data to the server. These data can be used to allow an advertisement platform to be more user-specific and less intrusive to the reading experience than existing approaches. The nature and variety of the data that are to be collected remain to be defined and will entirely depend on developments in the other related parts of the overall Leaf project.

The development of the whole application, client and server sides, has been achieved even with the usage of a remote repository running a Subversion server[12]. This allowed the team to easily collaborate and exchange portions of code, as well as developing and testing on different machines an configurations.

Finally, is important to specify that the Leaf project takes advantage of a streaming-like strategy, to avoid the need to fully download documents on the user's device, in order to satisfy a set of business policies. These policies are described in the companion masters thesis by George Khalil[L2] and Diego Botero[L1].

1.4.1 The Android OS as main interface for the Leaf platform

The main mobile interface upon which the client application will run is the Android OS, currently distributed for both smartphones and tablets.

The first reason behind this choice is the wide adoption of Android OS, worldwide, which makes this Operating System to be the most used mobile OS for smartphones, according to many statistics, such as the Millennial Mobile Mix[13] for this summer or Gartner's press release[14] of April 2011. This mobile OS is currently, third quarter of 2011, distributed in its latest releases 2.3 Gingerbread for smartphones, and 3.0 Honeycomb for tablets. Among the mobile devices manufacturers that produce Android devices, we can

find Samsung[15], HTC [16], Sony-Ericsson [17], Motorola[18] and many more. The vast spread of Android OS can represent a key point when the application will be ready to be distributed to the public, which will be composed in this way by a potential of millions of users all around the world; of course this number will be better refined by the market estimations that will be performed by the business team.

The development environment that characterizes Android is a free environment, in which the developers do not need to pay for the Android SDK[19] or for any other tool. In the case of other mobile platforms, like the Apple[20] case, developers have to pay a fee for the right of using their development kit. In addition to this, in these “closed” environments the basic language is a “customized” version of Java that the developers have to learn in advance. On the contrary, the Android SDK is an environment based on Java, with some added functions to interact with the GUI controls. The GUI is constructed and managed thanks to XML[21] files, one per activity.

1.5 Server-side application

Together with the Android client, the project will include a server-side application to implement a series of operations while uploading a new digital publication in the platform.

The server side application includes two highly connected parts which allow a user to upload a new document in the server. The basic architecture of the server involves an Apache web server to host publications and webpages and a PostgreSQL database to keep track of users and documents (see [L3] Sebastian Galiano’s project Thesis for a complete overview of the server’s infrastructure).

The first part of the server side application is represented by a PHP based website, in which the user can insert a new publication and specify a title for this document. Then the website performs the upload of the publication into a dedicated folder on the web server and inserts the publication’s data into the database.

The second part of the server side application is a Java package that analyses the documents currently “in upload”, retrieves data about the publication for the database and automatically builds a document summary to improve the content navigation.

Measurements and implementation details regarding the client and server applications can be found in chapters 4 and 5.

The document will now introduce a background on the world of electronic publications, in chapter 2. Then chapter 3 will give a detailed insight into Android OS and the development of Android applications. The components of the client application are explained in details on chapter 4, then chapter 5 describes the implementation of the served side for this project. In chapter 6 the measurements made on the application are first illustrated and then analyzed. In last chapter, chapter 7, the document gives a description of the achieved goals and future work plans for this project.

2. Electronic publications

In this chapter's first part, the report analyzes the different types of electronic publications currently in use. The attention is mainly focused on the ePub format, its development and its technical structure.

The second part describes and then evaluates the existing e-reading platforms currently in use. In subsection 2.3.1 a general discourse introduces the most famous products currently in the market, then it illustrates their features and performances. A table will summarize and highlight the advantages and disadvantages of each of these solutions.

2.1 Overview of electronic document formats

Digital publications have been produced in several different formats - using a variety of standards. These different formats have caused a significant level of uncertainty for both industrial actors and consumers. Adobe Systems Incorporated developed the widely used Portable Document Format (PDF)[22]. This format has been very popular because the specification was public and Adobe made a reader available for nearly all popular platforms. Additionally, there are open source implementations of PDF readers, such as Evince, Okular, and GSview.

In the late nineteen nineties a company called SoftBook Press Inc.[23] introduced its first eBook reader, which was able to read books in a special standard called "Open eBook". The Open eBook[24] standard, OEB, can be considered as an ancestor of the current eBook format ePub. Released in September 1999, an OEB file consists of a compressed ZIP archive containing XHTML pages, XML[21] documents for information handling, a manifest file to hold the document description, and external resources such as CSS[7] style sheets. The description of resources such as images and other media is based upon a collection of metadata selectors called the Dublin Core[25]. The Open eBook standard, also called OEBPS (Open Ebook Publication Structure), was formally replaced after eight years, September 2007, with the Open Publication Structure 2.0[26], that evolved into the ePub standard.

The ePub is a digital publications format managed by the International Digital Publishing Forum (IDPF). The ePub format currently represents the most widely adopted and compatible document format used by e-Reader devices.

In the digital publications market, it is possible to find many other formats, such as Palm Media's "*PalmDOC*"[27], DjVu[28], Mobipocket[29], and Amazon's "*.AZW*"[30]. However, these formats are proprietary standards owned by specific e-Reader device manufacturers and publishing/distribution companies, hence they are considered *de facto* standards.

Finally it is important to mention that several digital publications have been released in file formats that are commonly used for other purposes, such as text (*.txt),* HyperText Markup Language (HTML), and PostScript (*.ps*).

2.2 The ePub standard

The ePub document format represents today the most compatible format for electronic publications, with ePub used by every virtual bookshop and eBook website on the Internet. Thanks to its structure and organization, it allows documents to be universally read and optimized in any device, regardless of screen format or operating system. This format is officially supported by every major e-reader software and hardware, except for Amazon's solution (that will be presented in the next section).

The current official release of ePub format is 2.0.1. However, IDPF has released their draft specification of ePub 3.0[11], which will be formally adopted by the end of 2011. Until then the specifications regarding new media will be refined.

A document published in the ePub standard is a compressed archive with the *.epub* file extension. The ePub document makes use of three different formats, which specify the file contents, aspect, and packaging.

2.2.1 Open Publication Structure

The Open Publication Structure (OPS)[26] is the official specification defined by the IDPF in order to ensure homogeneity with regard to management of electronic publications. The current official release of the Open Publication Structure is version 2.0.1, v1.0.1 (September 2007).

OPS is used in order to manage three kind of entities: style sheets, characters encoding information, and MIME-type[31] declarations for resources such as images and other media types. An important construct to encode OPS publications are "XML Islands".

"XML Islands" are fragments of XML[21] code that are used to declare information and resources inside the document; thanks to these "islands", it is possible to declare images and style sheets by using the appropriate MIME-types, as well as to include information regarding the document itself. These fragments of XML [21] code can be inserted into the document as both inline or out-of-line code.

It is possible to include in a document multiple CSS[7] selectors. These describe the graphical appearance of the document. The CSS[7] used for this is called an OPS Style Sheet and it is composed using CSS 2.0 rules, plus a limited list of custom CSS[7] properties such as *oeb-page-head* and *oeb-column-number*. The CSS[7] mime-type should be *text/css*.

Another important concept of the OPS[26] standard is that of a MIME-type declaration. Each MIME-type declaration explicitly tells how to specify MIME-types for images and how to include resources with a non-standard MIME-type[31].

Lastly OPS[26] defines how to declare the character encoding that should be used within a document. This is necessary in order to support compatibility with multiple languages. The standard encoding set is UTF-8 or UTF-16, although is still possible to produce documents with other encodings.

2.2.2 Open Packaging Format

Open Packaging Format (OPF)[32] (release 2.0.1 v1.0.1) is the name of the specification used to define the organization of contents inside an OPS[26] publication. The aim of OPF [32] is to create a well structured and easy to navigate electronic publication. The main components defined by Open Packaging Format[32] are the *.opf* and the *.ncx* files.

The .opf file is the main descriptor of a document. It has a general parent XML[11] node, called <*package*>, which contains three required elements <*metadata*>, <*manifest*>*,* and <*spine*> plus an optional node, <*guide*>, that can be used to point to particular structural elements of the publication, such as the references.

The <*metadata*> element defines, through a series of specific tags, essential information regarding the document, with the syntax: <**dc:***metadataTag*> Value **</dc:***metadataTag*>. Among the metadata tags are three required ones: title, language, and identifier; in addition, there are a series of optional information such as creator, coverage, type etc. etc.. The majority of metadata tags can be enriched with attributes, which in some cases follow specific rules. An example of such an attribute is the identifier's *id* that has to be equal to the package's *unique-identifier*.

In the <*manifest*> element, all the files contained in the document, including XHTML pages, css, and images, should be declared. Between the beginning and closing tags of the manifest, all the files are contained in <item> tags, each of which should have a unique *id*, the file path (*href*), and *media-type* (the MIME-type[31]).

The reading order of the XHTML documents contained in the ePub file is expressed in the <*spine*> element, which lists the documents with *itemref* tags, in the same linear order as the document chapters or parts follow.

The second file which composes the Open Packaging Format is the .ncx file. This file extension is derived from the name "Navigation Control file for XML". This contains the information that can be used to create a navigation panel for the document, presented as a Table of Contents. The organization if the .ncx file is hierarchical and each subelement is composed of a <*head*> tag, followed by <*docTitle*>, <*docAuthor*>*,* and <*navMap*>. All of these elements are wrapped in a general <*ncx*> tag.

The <*head*> tag contains generic information regarding the document, such as the *maxPageNumber* and *totalPageCount*. Together with the values expressed in <*docTitle*> and <*docAuthor*>, the unique id contained in the <*head*> should match the information stored in the .opf manifest.

The table of contents construction is based upon the <*navMap*> tag, which contains a series of <*navPoint*> elements that represent single entities of the table. Each <*navPoint*> contains two sub-tags, one containing the chapter name and the other a link value, respectively labelled with the tags <*navLabel*> and <*content*>.

2.2.3 OEBPS Container Format

The OEBPS Container Format (OCF)[33] standard defines how the different publications components should be organized in the compressed archive, which is a .zip file[34]. According to the OCF definition, the main ePub directory should contain a folder called "*META-INF*" and a file called "mimetype" as the first file in the folder.

Inside the "*META-INF*" folder there should be an XML definition file called *container.xml*, in which the following tag structure indicates the .opf file and additional, optional, external files that define the content of the publication. This takes the form of:
<rootfiles>
<rootfile full-path="path" media-type="application/oebps-package+xml">
</rootfiles>

The "mimetype" file, which should be uncompressed and unarchived, has to contain the string "*application/epub+zip"*.

2.2.4 ePub 3.0 draft specification

During the first half of February 2011, the IDPF released a series of preliminary draft specifications for the ePub 3.0 standard[11], which will be composed of the following quartet of standards: Content Document 3.0[35], Publications 3.0[36], Open Container Format 3.0[37], and Media Overlays 3.0[38]. It is important to note that these specifications are currently "early drafts", so until the publication of the definitive versions they should be considered as potentially subject to significant changes, in structure and meanings.

According to the draft specification, Content Document 3.0[35] will supersede the Open Publication Structure[15] and introduce several important changes, including HTML5[6], scripting, MathML support, semantic inflection of domain-specific information, and SVG[39] documents. In addition the .ncx navigation file, will be deprecated, and a new navigation format will be defined by EPUB Navigation Documents. In particular a part of Content Document 3.0[35], specifically "XML Islands" will be deprecated.

The Open Packaging Format[32] will be replaced by Publications 3.0[36], which does not introduce significant changes in the document organization structure, but includes a small list of minor new elements such as the d*cterms:modified* property and a metadata "*link*" element. A significant addition in Publications 3.0[36] is the definition of support for Core Media Types, which represents the official introduction of audio and video contents to ePub documents, thanks to the HTML5 [6] <*audio*> and <*video*> tags.

In the OCF 3.0[33] definition, the most important changes concern the "*META-INF*" folder, in which the container.xml is still required, but can be extended with new definition files, such as signatures.xml, encryption.xml, metadata.xml, rights.xml, and manifest.xml. The container zip file will lose its abstraction of a file system, and become more of a regular zip archive.

Finally the new ePub 3.0[21] predicts the introduction of two new specifications: Overview[21] and MediaOverlays[38]. The first one is a description of the ePub 3.0[21] package. The later, MediaOverlays[38], are intended to define how to realize audio and text synchronization.

2.2.5 ePub 2.0/3.0 as the standard for the Leaf platform

The Leaf platform, as mentioned in the second paragraph of section 1.4, uses ePub 2.0 (and ePub 3.0, in the next version) as the standard for contents distributed through it.

The reasons behind the decision of using ePub as a standard format for digital publications are many.

First, the wide adoption of ePub documents is a driving element that convinced the team into this direction. Thanks to the fact of being a standard, only composed by standard-format files, a big amount of nowadays digital publications are created in ePub. Doing so, authors do not have to use proprietary editing tools, but they can take advantage of many different solutions. In addition to this, having documents in ePub let contents owners to distribute these publications on many different platforms. As viewed in table 1, the ePub format is the most compatible in terms of number of distributing platforms, a fact that may convince authors to have at least an ePub version, to be widely distributed.

A second key-element that makes ePub the format for the Leaf platform is its high correlation with web technologies. Being based upon HTML/XML documents, with CSSx styling and JavaScript[8] compatibility, makes the ePub a very versatile format. Thanks to this, contents have the same advantages of web pages, meaning an high level of customization, possibility of interactions embedded in the documents, easy layout and graphic elaboration, and a great capacity of involving multimedia contents.

To not being bind to a proprietary platform, together with the high possibilities given by the web-oriented nature of ePub and its consequential wide adoption, represents a multiple force of this document format that could not be ignored while selecting an initial format for the platform. Considering the well-structured organization of the contents (as shown in the previous subsections) and the

2.3 Existing platforms

The current range of ways to access e-publications requires many different solutions. Hence even though the main goal of Leaf project is to reach the entire market from a different angle, not all of these mechanisms will be (or should be) available in the first prototype.

With regard to books, the main solution today is represented by virtual bookshops hosted on dedicated Internet websites, such as the world-famous Amazon.com[40]. The primary approach used to reach mobile users for such content is through device-specific applications, which offer both a reading experience and content browsing. This functionality is commonly achieved due to access the same data available on the websites, with an intuitive GUI that has been designed for the mobile environment and a pay-per-download philosophy. The most famous example of this is Amazon.com[40] and its *Kindle* application (which is part of the Kindle platform); Kindle is an eBook reader device sold by Amazon.com. Alternatives are Apple's *iBook*[41] and *iBookstore*, Barnes&Noble *Nook*[42] platform and the mobile application *Kobo*[43]. It should be noted that there are currently many websites acting as virtual bookshops, so that books can be licensed from these website and later moved to one or more mobile devices.

With respect to magazines, newspapers, and comics market the situation is a high fragmented market, because every major journal currently offers their own device-centric mobile applications that each have to be downloaded by the user. Each of these applications offers a different way of browsing information. Additionally, the user has to individually download and pay for each issue of a magazine, newspaper, or comic book. Recently Google and Apple mobile platforms, respectively Android and iOS, introduced for publishing companies the possibility of offer time-limited subscriptions to users who often read their contents.

Even though the main goal of the Leaf project is to realize something currently not offered by any on the above mentioned solutions, there are several projects that will be deeply analyzed during the course of the project both to compare with the Leaf project and to learn from these other efforts.

The service called *"PressDisplay.com"*[44] powered by NewspaperDirect Inc. offers to its users the possibility of browsing a very large number of newspapers from almost every country in the world, giving these users the possibility of enjoying contents while offering different payments options. Users can read contents directly on the website, which has an integrated e-reader application with high compatibility with every journal and with a basic set of reading tools. The same company also offers a mobile application for different platforms called "Readers Hub", which aggregates *PressDisplay.com*[44], *Kobo eBooks Reader*[43], and *Zinio magazines*[45] in the same interface. Users can directly access these mobile services and then read the contents offered by each of them, according to their individual commercial strategy. One of the most interesting services offered by *PressDisplay.com*[44] is their web interface, that presents all the latest news from different newspapers in the same page and layout. When the user clicks on the desired article, he or she is immediately redirected to the "reading mode" of the newspaper offering this article.

Another interesting project that will be studied is the *OpenLibrary*[46] project, a website that offers free ebook reading (online) to visitors. Among the most interesting features of this project, the most relevant are the online e-reader application, which efficiently operates on different browsers, and their open space for developers. In this "open space" developers can download open-source code of the platform components and contribute to their further improvement. However, the system currently offers books that are not covered by copyright protection, therefore they can be distributed royalty-free.

Recently, two new services have been introduced in the market, with the aim of distributing electronic publications on a flat-rate model. The first one, called "*24Symbols*"[47], is a spanish start-up that proposes a wide catalogue of regular books with a business model based upon subscriptions. The second one, *"Platify"*[48], is a swedish project with the same final objective on the business side; this project is currently working in the field of academic books, offering royalties-free books. On the technical side, these two services offer only a web-application to read contents, but when their launching phases will be finalized, they will both come back into the market with mobile-based solutions.

With respect to existing Android e-reader applications, the main solutions are the *Aldiko eBook reader*, *Borders eBook, FBReader, Moon+ reader,* and applications such as *Kindle app*[49] and *Kobo*[43].

2.3.1 Evaluation of existing e-readers softwares and features

In this section we discuss the main features of the alternative solutions named in the previous section, and how these features influence the performance of the system.

The first feature to analyze is the way to obtain documents. The most widely adopted solution is to download of the whole document to the reading device, then accessing the document. This approach, used by all the e-reader software, prevents unauthorized redistributions of content by using Digital Rights Management (DRM)[50] protection in those books that are protected by copyrights. Despite DRM's wide usage, this protection mechanism has been subject to heavy criticism by several organizations worldwide. Additionally, DRM protection on books has also been cracked many times, on several different platforms.

No one of the clients currently in the market can be used to access and read online contents without downloading it entirely, but several solutions such as *OpenLibrary*[46] allow their users to read contents online, since they are actually implemented as web services hence they are accessible through the browser. With regard to magazines and journals, since the majority of them are distributed as mobile applications or websites, there are no clients suitable for this analysis, because they don't perform a connection between client and server and the content is part of the application. Despite of this, they can still be analyzed as models of e-reading solutions, especially in terms of user controls and available functions.

The file format support is an important element to evaluate before proceeding with the design of the client software. Even if the Leaf project will mainly work with ePub, because if its wide compatibility with commercial readers (almost every device except Amazon's Kindle [49]), an analysis of the most important file formats for e-publications is important. Excluding the ePub format, readable on every device (but the Amazon Kindle [49]), Text (*txt*) and Adobe's PDF are the most compatible formats, although in reality they were designed for other purposes. A good compatibility range is given by HTML, DjVu [28], and MobiPocket[19] formats, even if they still lack in terms of interactivity with document contents. The Open eBook[24] and FictionBook[51] formats are supported by some important devices such as Apple's products, even though they provide a more basic experience when compared to ePub.

With respect to GUI implementations, there are some aspects to analyze in order to better understand people's needs when reading an e-document. Brightness control, Day/Night modes, custom font size, and auto-adjustment to display size are essential features that every modern e-reader application has, independently of wether it is running on a tablet or ebook reader device. Navigation control is realized with a set of tools, such as table of contents, auto-resume, volume key paging, and bookmarks are implemented into all ebook softwares. Additional features, such as full text search (present in *Aldiko, Apple's iBook*[41] and *FBReader*) or text-to-speech functionalities (in OpenLibrary[46]) are key elements to attract people into this new way of reading. A short list of additional functions offered to readers include the possibility of taking notes, because it is an advanced feature only valid for devices with a touchscreen (or a keyboard). Highlighting and search/share in the text, as well as accessing web links, are present in all the softwares installed on devices able to access the Internet.

Data synchronization is a feature that some clients, such as *Kobo eBook Reader*[43] and *Amazon's Kindle*[49] , include in order to enable the user to have an account accessible through many devices. To do this, the software automatically checks for this user's information on the server and downloads, if necessary, new content via the client application.

Accessing virtual bookstores is an essential feature present in many important applications, such as *Apple iBook*[41], *Amazon's Kindle*[49], *Kobo Reader*[43], *Barnes&Noble Nook*[42], and *Borders' Ebook*. The other clients can access local documents which are stored in the device's memory, or in virtual catalogs in case of a web service (accessible through a browser).

Table 1 lists features that can be found in the major clients, on different platforms. The items have been ordered from the one with the highest number of features (in terms of functions and supported formats, giving priority to the non-proprietary ones). This table does not consider the real application's appreciation in the market, the advantages some companies have on their own formats and the physical devices performance.

Table 1 - Comparison of features in different e-reader clients

*(*Clients with more features are showed in the top)*	Supported Formats (Non-proprietary)	Supported Formats (Proprietary)	Data Synchronization	Day/Night Mode	Custom Font size	Virtual Store
Apple's iBook	txt, ePub, html, Open eBook, pdf	MobiPocket, FictionBook, DjVu, eReader, azw, TomeRaider	YES	YES	YES	YES
Kobo Ebook Reader	txt, ePub, html, Open eBook, DjVu, pdf	MobiPocket, FictionBook, eReader, azw, TomeRaider	YES	YES	YES	YES
Amazon's Kindle 3	txt, html, pdf	MobiPocket, azw	YES	YES	YES	YES
OpenLibrary	txt, ePub, html, DjVu, pdf	azw, DAISY, MobiPocket	YES	YES	YES	YES
pressDisplay.com	txt, ePub, html, Open eBook, pdf	---	YES	YES	YES	YES
Barnes&Noble Nook	txt, ePub, pdf	eReader	YES	YES	YES	YES
Lexcycle Stanza	txt, html, ePub, pdf	rtf, doc, eReader, Lit, azw	YES	YES	YES	NO
Adobe Digital Editions	ePub, pdf	Flash	YES	YES	YES	NO
FBReader	txt, ePub, html, Open eBook	FictionBook, MobiPocket, rtf	NO	YES	YES	NO
Aldiko Ebook Reader	ePub, pdf	---	NO	YES	YES	NO

3. Android OS and SDK environment

3.1 Android OS for mobile devices

In 2005 Google Inc. acquired the Android project, which is a mobile operating system (OS) developed and maintained by the Open Handset Alliance[52]. This alliance included approximately eighty technology firms and had the main goal of developing an open platform for cellular handsets, with the aim of enabling a new ecology of software developers. Currently the development of new versions and the maintenance of the current version's stability is carried out by the Android Open Source Project[53] (ASOP). The Android mobile OS currently represents the most wide-spread mobile phone development platform. According to the statistics presented in subsection 1.4.1, ASOP currently has a market share of 29% (by March the third, 2011). Additionally, there were thirty-three millions Android handsets sold in 2010.

The base upon which the Android OS runs is a register-based virtual machine, capable of using a Just-In-Time[54] compilation (JIT). JIT compiles code just when there is a need for it, hence the portions of code to compile can be arbitrary chosen (for example, only a fragment of code, a file, or an individual function) and after compilation the code is inserted into a cache so that it can be easily re-accessed in the future. This register-based VM is known as the Dalvik VM[55]. Before running Android applications, the OS converts the application into the compact Dalvik executable (*.dex*) format, to improve the execution even when memory and microprocessor resources are limited.

On top of the Virtual Machine, the Android OS is composed of a set of dynamically loadable Java libraries, called the Java Class Libraries. These libraries can be called at run time by Java applications. The main language used in the development of these Java Class Libraries was Java, however some parts are written in C because there was a need to access the underlying hardware resources and the OS. These Java Class Libraries are used to achieve three different, but essential purposes: (1) access to common services such as file and network input and output, (2) provide the features required by Java that a given platform may not support (this is because of Java's independence from any specific platform), and (3) providing the developer with a set of well structured and organized standard code libraries, with functions to perform many common tasks.

The Java Class Libraries are presented to the user with a specific object-oriented Application Framework[56], together with an intuitive GUI. This GUI is standardized in all the devices with the Android OS pre-installed. The Application Framework's[56] main components are two entities called *Activity* and *Service*; these are present in all Android applications. The *Activity* is associated with a single screen and a user interface. An activity might involve for example displaying information or filling a text form. The *Service* component, on the other hand, represents something that the application should run during a long time span, without user interaction or information flow from the application to another application. It is also important to note that a *Service* does **not** have a user interface. The Android Application Framework[56] is composed of many other components, including Tasks, Threads, Loopers [57], and more. Details of all of these components, classes, etc. are available from Android's official developer website [57].

The highest level in Android's software stack is represented by applications. An Android application can be downloaded from Google's official marketplace (called Android Market) or from a third party website. Each application always comes as an Android Package (*.apk*) file, which is a variation of a standard java JAR file. The development and structure of an Android application will be further analyzed in the next sections.

3.2 The Android SDK

The Android Software Development Kit (SDK)[19] provided by the Android project is the official set of tools that developers can use to create Android applications.

3.2.1 Technical overview on the Android SDK

The Android SDK[19] relies on two main software repositories: *Android Repository* and *Third party Add-ons*. Using these sources the developer can access a set of "*basic tools"* (SDK Tools, SDK Platform-tools, and SDK Platform) which are required for application development, *"recommended"* (documentation, samples, and Microsoft Windows USB-drivers), and "*supplementary"* (Google APi and additional SDK platforms).

Any Java-ready Integrated Development Environment (IDE) can be used to create Android applications. However, the officially recommended platform is the Eclipse IDE (specifically the Galileo, Helios, or Ganymede versions) which can integrate the Android Development Tools (ADT) plugin, an extension that facilitates Android applications development within the aforementioned Eclipse versions. Together with the ADT plugin, the Android Virtual Device (AVD) manager for Eclipse IDE can be used to easily keep the installed Android SDK[19] version and its components up-to-date and making it easy to run applications on both virtual or real devices.

3.2.2 Android project's structure, resources, and R.java

Every Android project[19] should follow the same standard structure, i.e., it should be organized into several folders: *"src", "gen", "assets", "Android x.x", "res";* in addition some files such as the AndroidManifest.xml and default.properties should be present.

The *"src"* (stands for *"sources"*) folder will contain all of the Java sources packages, as well as the AndroidManifest.xml file. We will examine details of this directory in section 3.2.4.

The*"gen"* folder contains the Generated Java Files, which are automatically produced by the SDK[19] engine. One of the files in this folder is the R.java file, which provides the "*index*" of an Android project. This automatically generated file assigns to every resource in the project (strings, layout objects, and other elements contained in the *"res"* folder) a specific constant value, creating a mapping between object identifiers (ids) and these *public static final int* numbers. It is important to note that every object's id in the project is defined by the R.java file. As this file is automatically generated it should not be manually modified. Several classes appear in the R.java file to indicate the behavior of the different resources' types in the application, these include strings and drawables.

The folder called *“assets”*[58] contains raw files to be used by the project. For example, such a file might be used to define special fonts. These files will be used by the project *only* as input streams, because the “*assets”*[58] folder can not be accessed to perform other kinds of activities. Despite this limitation, this folder can be organized into subfolders, to give the degree of flexibility required by the application to the files that will be stored in this folder.

Inside the *“Android x.x”* folder we can find the current Android version, declared by the *“x.x”* statement. This defines the minimum version of the “android.jar” supported by this specific application.

Finally, we have the resources folder[59], *“res”,* in which media files such as images and functional elements (such as layout definitions and standard strings are stored). This folder is divided into several sub-folders, depending on the resource type that the folder contains; functional images (such as icons and layout elements) are stored in the three folders called *“drawable-hdpi”, “drawable-mdpi”* and *“drawable-ldpi”*, depending on the resolution (high, medium, or low) of these images. These resolution formats are illustrated in table 1 which shows the size of icons and some common tabs.

Table 2 - Resolution formats scheme

Element	hdpi	mdpi	ldpi
Launcher & Menu icons	72 x 72 px	48 x 48 px	36 x 36 px
Tab, Dialog, List View	48 x 48 px	32 x 32 px	24 x 24 px

An essential resource sub-folder is *“layout”*[60], in which layout definitions are defined in a series of .xml files, normally containing a file main.xml and other files for different parts of the application. The layout management and the .xml layout files will be discussed in section 3.2.3.

The *“values”* directory is another sub-folder of *“res”* and it stores xml value declarations, such as the (required) file *strings.xml*. In this file all the strings and string-arrays that will appear in the application’s GUI should be stored, with a specific <*string name=”stringName”*> tag to wrap the string’s value. Similarly, string-arrays should be declared with a <*string-arrays*> tag containing a list of <*item*> tags to contain array values. Even if the value of “Text” elements will be hard coded or set with XML, creating *strings.xml* entries is encouraged because these entries can easily be utilized when translating the interface to another language. An example *strings.xml* file is shown in Figure 1.

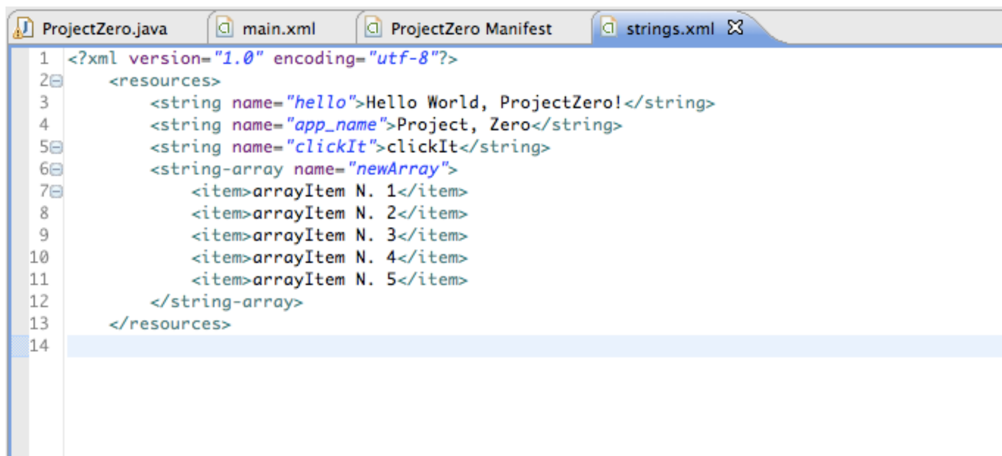

Figure 1 - Sample *strings.xml* file

The *“res”* folder may also include an optional folder called *“raw”*, in which the developer can store raw files that need to be accessed other then as a input stream. (Note that raw files that are only accessed as an input string can already be accessed from the *“assets”* folder).

3.2.3 Android SDK Layouts

The “*layout”*[60] sub-folder of resources folder in a typical Android project is intended to store a series of .xml files that define the application’s layout. A layout specification for an Android application is composed by one or more XML files that can be accessed and referenced by the Java classes of the project’s source code. Normally an Android application binds a layout definition to each application’s activities and the main activity is related to the *main.xml* file.

The main structure of a XML Android layout[60] is organized around the “View” concept. A *“View”* is a generic Java object and class that represents a visual entity in the application’s GUI. Many Java classes are able to extends the *“View”*[61] class; for example *Buttons* and *TextViews*, that are elaborated in the Java source code. The visual behavior of each of these elements is defined in the XML layout[60] files.

A *“View”*[61] can be referred to by XML code, using its unique identifier (*id*). The view contains the main graphical features, such as layout positioning, text attributes, and dimensions. The possibility of inserting text in the elements to predict a text value, such as button’s label. Depending on the XML tag, that should be nested in the view’s container, a list of attributes to define the object’s visual appearance can be used to define the object’s appearance. The complete list of these attributes and their respective values can be found at the official Android Developer Website at *http://developer.android.com/guide/topics/ui/declaring-layout.html#attributes*.

Among the list of Java Views[61] that can be modeled with XML, we can identify several functional categories: *Layouts, Buttons, TextViews, WebViews, EditTexts*, and many more. These categories include elements that can be inserted in the application’s layout, with specific functions. While *Buttons* and *EditText* are elements that give the user interactive objects, *TextViews, WebViews* and *Layouts* form the structure of the application. It is important to note that some XML attributes, such as

android:layout_width, *android:layout_weight* should be declared in order to create the graphic instance of the element; even if the attribute is not explicitly required, the *android:id* attribute should be declared in order to access the element from the Java source code.

Special attention should be given when talking about *"Layout"* objects. These objects are the main containers of the application's visual structure. The main layout types are *"LinearLayout"* (a linear structure to define an ordered disposition of the elements), *"RelativeLayout"* (gives the possibility of relative positioning), and *"AbsoluteLayout"* (to use to enable the developer to specify absolute positioning). A particular layout style is the *"TabsLayout"*, which is composed of several kinds of minor layouts and allow the creation of a structure with different tabs (an example of a TabsLayout is shown in Figure 2). All the elements that appear inside the layout should be nested inside the layout XML tag.

```xml
<?xml version="1.0" encoding="utf-8"?>
<TabHost xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/tabhost"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:padding="5dp">
        <TabWidget
            android:id="@android:id/tabs"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content" />
        <FrameLayout
            android:id="@android:id/tabcontent"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:padding="5dp">

        <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
                        android:id="@+id/tabcontent"
                        android:orientation="vertical"
                        android:background="#ffffff"
                        android:layout_width="match_parent"
                        android:layout_height="match_parent"
                        android:padding="20dp">

                         <TextView
                            android:id="@+id/listitle"
                            android:layout_width="wrap_content"
                            android:layout_height="wrap_content"
                            android:text="@string/listitlefruit"
                            android:textSize="12dp"
                            android:layout_alignParentTop="true"
                            android:layout_centerHorizontal="true"
                         />

            </RelativeLayout>
        </FrameLayout>
    </LinearLayout>
</TabHost>
```

Figure 2 - Sample of a *TabsLayout*

3.2.4 Android SDK source code

The core functionalities and declarations of an Android project are defined in source code packages, inside the *"src"* folder of the application package. Just as in a regular Java project, each source package has several .java files with different classes and interfaces.

The main structure of a .java file inside an Android source package is the same as in regular Java files. The first part of the document is reserved for the import of libraries that will be used in the class and then the class declaration should be written, with eventual extensions and the implementation of other classes.

Inside the class the first part is reserved for variables and constant declarations, furthermore we find all the methods that will compose the class, constructed using the standard Java syntax to declare visibility and parameters.

An essential method which should be present in every Android application's main class is the *"public void onCreate(Bundle savedInstanceState)"* method, that will include the application's starting activity.

To set the layout of the XML definition that will be used to render the Java class, the method to use is *"setContentView(R.layout.xmlName)*. The method's parameter directly points to the *"layout"*[60] folder, thanks to R.java mapping.

Views can be retrieved from the layout files with the method *findViewById (R.id.viewId)*. Each view must be set as an instance of a View[61] object declared in the objects and variables declaration part. Every view, depending on its type, has a set of predefined methods to invoke functions related to its specific purpose. One of the most relevant method that is offered by the majority of View[61] objects is the *setOnClickListener(View v)*. This method allows the creation of listener instances to bind, as an example, the pressing of a button to a specific action.

Activities can be created and launched in different ways. The main approach is the creation and usage of entities called *"intents"*. An *"Intent"* is an object that includes the definition of the main actions to perform with the activity that required the intent's initialization.

3.2.5 Android SDK project Manifest

Every Android project includes an XML file, the so called *"AndroidManifest.xml"*[62], which keeps all the information regarding the application in a single place. This file must have exactly this name, in every project, and it has to be located in the root folder of the application package.

The Manifest[62] file is divided into two parts, one dedicated to the declaration of the project's features and the other lists the activities of the application and the application 's properties.

The declarations part of the Manifest[62] file includes several specific attributes, such as *"android:versionCode", "androidVersionName", "package"* are used to specify this information. When a device wants to install an application, it will first look at the values of these attributes to check for compatibility and to ensure that the system meets the minimum requirements in order to run the software.

Another important source of information in the application declarations part is represented by multiple tags that the developer can insert to better specify important features of the software. Among these tags *<uses-sdk>* states the minimum, maximum, and target SDK[19] versions required to compile the software, *<supports-screens>* indicates that the application supports large sized screens. Additional similar tags include *<meta-data>, <permissions>,* and many more. These tags have tag-specific attributes; hence the developer does **not** have the possibility to define her/his own customized tags in the Manifest file[62].

The *<application>* tag begins the second part of the Manifest[62] file, and includes a series of *<activity>* tags that are used to specify the attributes and behaviors of each activity that comprises the application. Inside the *<activity>* tags we can find the tag *<intent-filter>* that contains *<action>* and *<category>* tags; these tags are used to specify through their attribute *"android:name"* what kind of activity we are declaring and what its purpose is. Essential attributes of the *<activity>* tag are *"android:name", "android:label",* and *"android:theme"* in which the developer specifies the corresponding information. The most important attributes for the *<application>* tag are *"android:icon"*, which should point to the location of an icon and *"android:label"* that must contain the value *"@string/app_name"*, a value contained in the *"strings.xml"* file that indicates the desired application name. An example *AndroidManifest.xml*[62] file is shown in Figure 3.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3        package="com.project.zero"
4        android:versionCode="1"
5        android:versionName="1.0">
6        <supports-screens
7          android:anyDensity="true"
8          android:largeScreens="true"
9        />
10     <application android:icon="@drawable/icon" android:label="@string/app_name" android:debuggable="true">
11         <activity android:name=".ProjectZero"
12                   android:label="@string/app_name">
13             <intent-filter>
14                 <action android:name="android.intent.action.MAIN" />
15                 <category android:name="android.intent.category.LAUNCHER" />
16             </intent-filter>
17         </activity>
18
19     </application>
20
21
22 </manifest>
```

Figure 3 - Sample AndroidManifest.xml file

Modifying the permissions within the Manifest[62] file allows the developer to restrict the application accessibility to other components or services, for example to prevent the application to access the network or the telephone's functionalities. An example of how to use the permissions in the Manifest[62] file is shown in Figure 4.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.me.app.myapp" >
    <permission android:name="com.me.app.myapp.permission.DEADLY_ACTIVITY"
        android:label="@string/permlab_deadlyActivity"
        android:description="@string/permdesc_deadlyActivity"
        android:permissionGroup="android.permission-group.COST_MONEY"
        android:protectionLevel="dangerous" />
    ...
</manifest>
```

Figure 4 - A sample of *<permission>* tag
Source: *http://developer.android.com/guide/topics/security/security.html*

3.2.6 Code compilation and the .apk packaging format

Once the code is ready to be compiled, the developer can start this process by invoking the Just-In-Time[54] compilation procedure. The standard procedure is based on four essential steps: (*1) packaging the resources, (2) javac compilation, (3) Dalvik*[55] *bytecode translation,* and *(4) creation of an unsigned apk file*. These steps, which transform the raw application package into a software ready to run on Android machines, can be executed using the IDE or, in the command line environment, with a simple set of dedicated commands to perform each of these steps individually. Further steps to sign the code with a key and publish it are optional, and these steps only need to be executed just if the developer wants to distribute the application to others.

Once the code is compiled, the application is packed into an *.apk* file, which contains a standard Android Package. This file is a compressed archive that can be opened with common archive tools, such as WinZip, 7-Zip, Ark, or WinRar. The *.apk* file format is the standard applications file format that Android equipped devices can open and execute.

4. Client application components

The development of the client application that will allow users to access the Leaf platform from their devices, is the main focus of this masters thesis project. This application, initially developed for Android systems, will be developed based upon several components, which will take care of different tasks, using specific controllers to handle information exchange.

As with every Android application, the components of the Leaf client are represented by "Activities", an abstraction of the different activities that user performs during the usage of the application. Every activity is implemented as a Java class, associated with a unique XML document that models its appearance. The main "Activities" composing the Leaf project's client application are described in figure 5.

The first activity's description, given in section 4.1, explains how the users are able to log in the platform in a secure and persistent way, thanks to the LDAP[63] interaction with the database that stores users' information. Persistency is based upon a special Cookie that is issued only to successfully recognized users.

The list of accessible resources is obtained by connecting to the Leaf platform server, currently hosted at KTH, where data are stored in a protected area. The Android activity called "*libraryActivity*" is responsible of connecting the client with this protected area and showing the user all the available contents. In a future version, a user's "personal space" will be introduced, giving users a private bookshelf hosted on the Leaf server.
Details of the current "*libraryActivity*" these will be discussed in the subsection 4.2.1.

The application's core, depicted in section 4.3, is represented by an e-Reading tool that presents e-publications to the user, interpreting HTML[6], CSS3[7], and scripting contents according to the ePub specifications. The user can navigate through the entire document via a set of commands to control the document's presentation and navigation, as well as returning to the previous activity and starting to read other publications.

Through all the activities, the application offers a GUI that presents contents in an innovative way, specifically designed to facilitate accessibility and usability. The design of this GUI, together with the specifications of its elements, will be discussed in section 4.4.

The following sections will analyze the current strategies and functions that the client application implements, giving a detailed overview of the implementation of the Android activities composing the client application

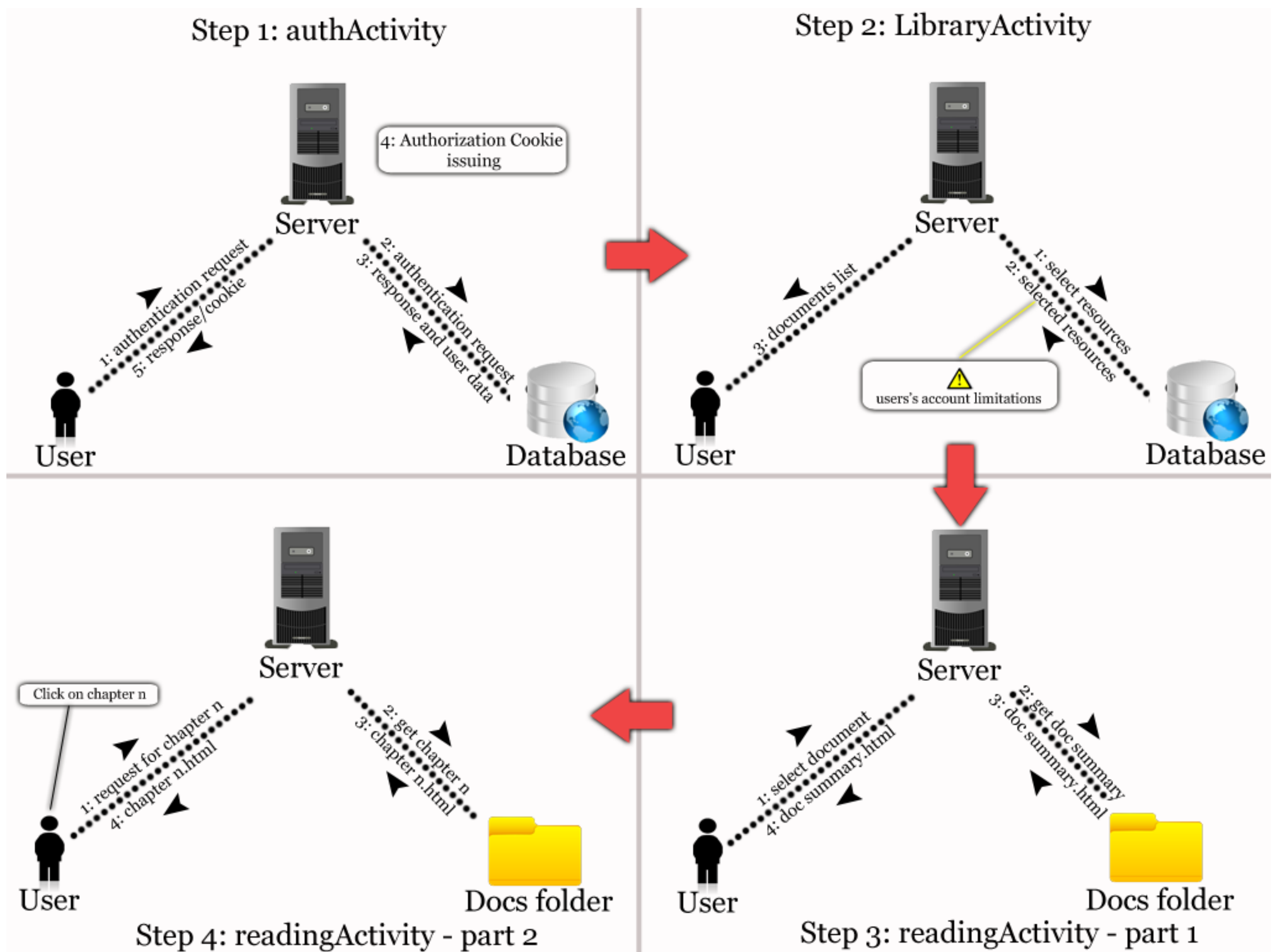

Figure 5 - Overall functioning of client application

4.1 Connection to the server and authentication

Since the client application was initially developed for mobile devices and it has to be able to present contents that might be updated daily (or perhaps even more frequently), the client application must be able to retrieve resources from the Leaf server platform. In order to achieve this, a crucial part of the development of the application was designing a means to automatically establish a secure connection to the server and *partially* download the desired data. Details of this operation will depend upon the user-side commercial policies (these policies are being defined in companion projects by George Khalil and Diego Botero, see [L2] and [L3]).

The main protocols that will be used are HTTP for the actual connection to the server and for data exchange and the Lightweight Directory Access Protocol (LDAP) [63] for authentication and directory browsing. The implementation of the authentication mechanism has been designed using LDAP's Bind operation together with a particular authentication strategy that relies on customized cookies. Details of this will be explained in subsection 4.1.1.

Currently the transport-layer protocol that is used in the client-server connection is a standard implementation of TCP[64]. The team is currently evaluating the usage of SCTP [65] streams, in order to avoid head-of-line blocking and to improve performance, especially when transferring audio and video data.

The design and development of the final version of the client application will build upon the current prototype, using a UML formulation to formalize the design description.

4.1.1 Authentication via authActivity.class to manage users

Information about the users are stored by the server-side application as database entities inside a dedicated table, with relevant subscriber/payment details, current documents, and other information (including publication-oriented data) related to each user.

When the user launches the client application, the first visible activity is the authentication activity (implemented by *authActivity.class*). This class presents a simple login form, into which the user enter his/her credentials to access the Leaf platform. Once the users send their credentials (username and password), these credentials are sent to the server, which compares these credentials against the user's information that is stored in the LDAP[63] system. If the credentials match then the user is allowed to proceed with the next activities, otherwise the users are prompted to re-enter their credentials .

The Java activity that executes this task is called *authActivity.class*. In the first part of this class, when the variables are declared, we find the instantiate a new HttpClient object, obtained from the *authManager class*. Details of this class, its usage and the way it is implemented, can be found on section 4.2.2.

The second part of *authActivity.class* collects the user's credentials from the form, from two "EditText" fields. The username and passwords strings are then stored in a private List, which is used to create a HttpPost object. This object is then assigned the correct connection url (which points at loginApp.php, see subsection 5.3.1) and sent in encoded form to the PHP page that performs the authentication using LDAP. This last process is performed by the HttpClient object (the one coming from authManager), which sends the Http POST request due to its "*execute*" method.

After sending the Http POST request, another Java variable stores the response entity sent by the server. Depending on the HTTP status code associated with this response, the *authActivity.class* decides whether to transition the user to the next activity or to ask again for the user's access credentials. In the case of a positive response, the java CookieStore is updated with the latest cookie issued by the server. For more details on the cookies handling, see section 5.3.2.

Another essential part of user management is the synchronization of each user's data. This should be sufficiently efficient to, as an example, store information regarding the user's bookmarks, in such a way that the user can open the same book on another device *without* losing their bookmark(s). With regard to the future development of an integrated advertising system, it will be necessary to synchronize relevant user data with the server, in order to better understand each user's behavior. This level of synchronization could be realized by one of several different schemes, such as SyncML.

Figure 6 depicts how the user authentication procedure works.

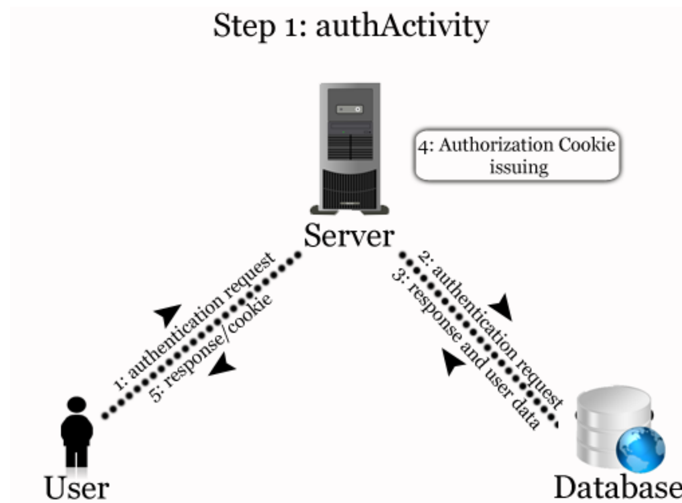

Figure 6 - User authentication

4.1.2 authManager.class

The Java class called "*authManager.class*" is an auxiliary class that is used to maintain the authentication session while the end-user is using the application. Thanks to this class, the user is able to continuously use the application without losing her/his authentication

The content of this class is a simple construction of a DefaultHttpClient[66] object that is used by all the other Java classes of the application. This strategy allows the different classes to always rely on the same object, avoiding the need to create multiple connections to the server. In addition, by using a common object such as the authManager, there is no need to pass data between different activities, decreasing the time need to launch each activity.

The DefaultHttpClient object contained in the authManager activity is instantiated with port 80 for HTTP connections and port 443 for HTTP Secure connections (it is expected that in the future all connections will utilize HTTPS). The number of connections is limited to two simultaneous connections most of the time and maximum of twenty by using a ThreadSafeClientConnManager. In future implementations we expect that, these constraints will be changed to allow a larger number of connections.

4.2 Data presentation

This section gives details on how the data are presented to the user, explaining how classes such as *libraryActivity.class* or *dbConnector.class* work. Despite the fact that code details are not revealed for commercial reasons, in the following subsections we illustrate structural details on how the retrieval of data, and its presentation to the user, have been developed.

4.2.1 Data synchronization and libraryActivity.class

A certain amount of data synchronization is a crucial part of the project. Since the client application will download only requested resources, the application should be synchronized with the server in such a way that the application can present updates to the user, who can decide whether to open a magazine's latest issue or open a book recently uploaded into the server.

During the development of the client application, a key activity was the so called "libraryActivity", which presents to the user the documents that are currently stored by the server. The *libraryActivity.class* file acts as a "bridge" between the retrieving of data from the server (see 4.2.3) and the reading activity (see 4.1.2).

The main component of the *libraryActivity.class* is the method "buildMenu", which takes the outputs of the database connection made by *dbConnector.class* (see 4.2.2) and creates a series of Android "button" elements, one for each document in the server (in a later stage this will happen only for the content present in the user's home page, such as favorites or suggested readings). After this, it binds an "onClickListener" method to these buttons, so when a user clicks on a specific button she/he will be re-directed to the reading activity, passing the document's ID as a parameter. In the future, these simple buttons will be replaced with a default sized image showing the document's cover. The *libraryActivity.class* is depicted by figure 7.

It is important to note that, at this stage of the project, the "libraryActivity" shows to every user all the documents currently in the server (and thus all the documents in the database). In the next phase of development, when user policies have been integrated, the documents presented to the user will only be the documents for which the current user has access. This access, which depends on the account status, will show to the user only the content that will be part of his/her homepage, such as favorite or suggested.

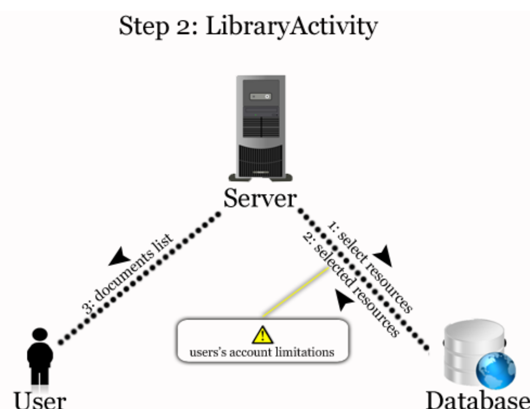

Figure 7 - Basic functioning of libraryActivity

4.2.2 Retrieving information with dbConnector.class

The “libraryActivity” enables the user to select which document to read, retrieves data from the server via a database query launched by *dbConnector.class*. This query asks the database for the contents that are actually stored in the server, and then passes the relevant documents’ data to the “libraryActivity”, which will use each of the publication’s titles to create “button” elements (see subsection 4.2.1).

Since the database is implemented using a PostgreSQL database, the *dbConnector.class* uses the “JDBC for PostgreSQL” driver to connect the Java application to the database. Once the driver is registered and the connection established, a “select” query is sent to the database, asking for data regarding all the documents currently registered in the “books” table (for details regarding the database structure and its tables, refer to Sebastian Galiano’s masters thesis[L3]). Each entry in the “books” table is then stored in a Java private map that will be returned to the “libraryActivity”.

When user policies are integrated, the query that will be sent to the database will be modified to be a query in which only resources associated with the user account’s type will be requested.

A relevant development of this Java utility will consider the security risks of the database queries being made by the client side. To avoid misuses or attacks, in next development phases we will adopt a more complex architecture, which will probably transfer the database interaction responsibility to a server-side script. The *dbConnector.class* will evolve into a Java class that interacts with a server side script, passing to this script a request and the user’s identifier; however, the information returned is expected to be in the same form as currently - thus the rest of the implementation will not be affected by this change.

4.3 E-Reader software

The core functionality of the client application is interpreting HTML5[6], CSS3[7], and scripting code to present the user with document pages to read. These pages can include embedded rich media and interactive elements. In order to perform these tasks, the client application uses a browser-like window element to show contents retrieved from the server.

The Android View[61] component called WebView offers a good solution to create a browser window, based on the well-know layout engine Webkit[67]. The decision to present contents through a webkit-based window is based upon the necessity of interpreting resources that are normally optimized for website rendering. The alternative for the development team could have been to integrate contents in more “classical” layouts part of the Android environment (such as embedding documents in RelativeLayout objects), but the rendering of documents conforming to the ePub 2.0/3.0 specifications would anyway require the use of browser-like surfaces.

Another important design decision that was made regarding document handling is the granularity of the application. The application currently operates on chapters as the smallest part of a document, in case of a book or articles in the case of a magazine. This decision was made due to the need to elaborate the different files within the same document. This also takes advantage of the reference that each chapter has in the summary creation process. In a later phase of the project, this decision may be revised to enable new navigation strategies.

4.3.1 Webkit engine

The Webkit[67] software is a widely used layout engine, which currently provides the base for Web browsers such as Google’s Chrome and Apple’s Safari. It was developed by an aggregation of many industrial actors, including KDE, Apple, RIM, Samsung, and Google. The software is distributed with a GNU lesser public license for its core components, while the other parts are licensed under a BSD license.

There are four main components to the Webkit engine[67]: WebCore, JavaScriptCore, Drosera, and SunSpider.

WebCore[67] is the main functional component of the engine and it is mainly used for layout rendering. It is composed of a collection of libraries that perform the construction of a page through Document Object Model (DOM) analysis and HTML interpretation. As with every modern layout engine, WebCore is able to render any standard markup and formatting language as visual information.

The JavaScriptCore[67] is a Javascript[8] engine responsible for interpreting and executing Javascript[8] code. This allows dynamic scripting. Recently the Webkit[67] JavaScriptCore has been rewritten in order to improve its speed and performance.

Drosera[67] was the Javascript debugging software released together with the Webkit[67] engine, but has currently been abandoned because of the inclusion of its functionalities in the Inspector, an analysis tool distributed with Webkit[67].

The last component of Webkit engine is SunSpider[67], a suite that can be used by analysts and developers to perform benchmarking on complex Javascript applications. It is currently widely used by developers in its 0.9.1 version.

4.3.2 readingActivity.class

The main activity of the client application takes place in the class in which the document reading experience occurs. This part has been implemented, as noted above, using the Android SDK component called WebView in the *readingActivity.class* Java class.

In the first part of the file containing this class there is a declaration of objects and variables that will be used, such as the reading window, the navigation buttons to move to other chapters, the button to come back to the summary, and the “information-visualizer” text box. After the initialization phase, the activity retrieves the ID and name of the document that the user wishes to access; these data comes from the “libraryActivity”, see subsection 4.2.1 .

A core part of the *readingActivity.class* is the initialization of settings regarding the WebView, via the “WebSettings” Java class. These settings allow developers to set particular behaviors for WebViews, such as: is zoom supported or is Javascript[8] enabled. The Javascript handling part is implemented by creating a Javascript interface that catches Javascript events during the reading experience and binds these occurrences with the desired Java methods. Details of the Javascript interface handling are given in subsection 4.1.2 .

Another essential part of the *readingActivity.class* is the enabling of external Javascript libraries, such as JQuery[9]. This is achieved by re-implementing the “onPageFinished” method, part of the Android SDK. This method is invoked when a page finishes being loaded into a WebView. In our implementation this appends Javascript code that loads the JQuery[9] library, if it has not already been included by the document. Along with the JQuery library, another Javascript file called *jsTools.js* is imported; details of this file are explained in subsection 4.1.3. Both *jsTools.js* and JQuery library are stored in the application’s assets folder.

The final relevant method in *readingActivity.class* is “setCurrentReadingState”. This method keeps track of which chapter the user is reading and which are the next and previous chapters. This is implemented using the chapter and book classes and will be explained in more detail in subsections 5.2.1 and 5.2.2 .

When all of the above operations are finished, the “readingActivity” shows the summary of the document inside the WebView window (for the summary creation details, see subsection 5.2.4). The user can select a chapter and then start reading it, calling the “pageLoad” event every time a new chapter is requested (as shown in figure 8, step 2).

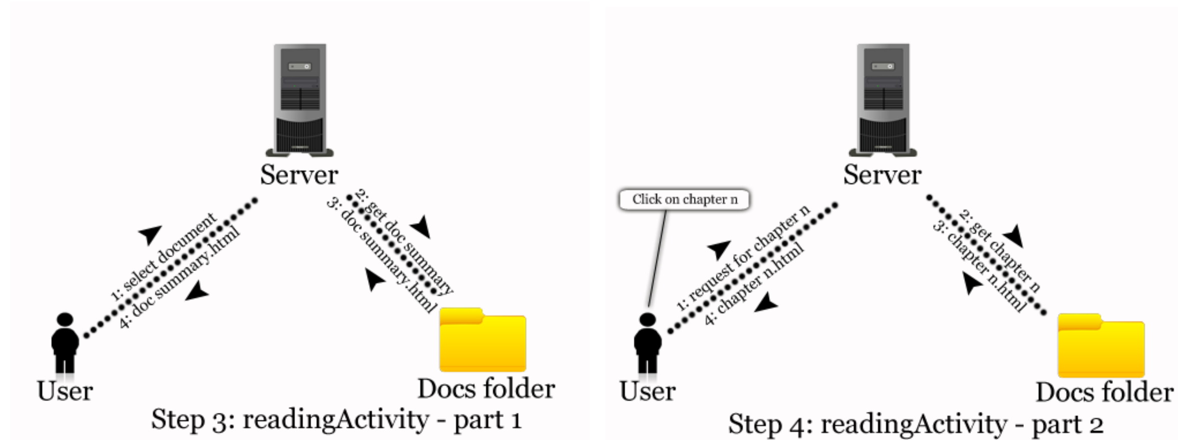

Figure 8 - Basic functioning of readingActivity

4.3.3 Javascript interface

The reading experience takes advantages of Javascript[8] technology, which together with HTML5[6] and CSS3[7] will provide to users a series of interactivity elements. In order to allow writers to insert interactive elements during document creation, it was necessary to create a Javascript interface.

The creation of *javascriptInterface.class* was required in order to bind Javascript events to the propagation of responses in the Java code to these events. In the code of the *javascriptInterface.java* file a set of methods are included, in which different Javascript events are coupled to separate Java methods. An example of this is the Javascript "onTouch" event, that will be converted into a Java "onTouch" event. This provides a means to invoke a particular function depending on the resource that has been touched by the end user. In this way, when the author creates content, she/he is able to determine which kind of reaction should occur when readers touch a particular image, section, or word.

The Javascript interface has been developed to interact even with external Javascript libraries, such as JQuery[9] or Prototype[10].

4.3.4 jsTools.js

Since the client application for the Leaf project includes Javascript[8] controls to handle particular events or to dynamically manipulate the document pages, a Javascript file called *jsTools.js* has been included in the main application's package. Inside *jsTools.js* it is possible to insert standard Javascript instructions, as well as non-standard libraries, in order to manipulate the overall appearance of the document.

The standard usage of *jsTools.js* file is to add an instruction that should be launched when the reader arrives at a certain point in her/his reading. Starting from that point, for instance, the font color could be changed to another value for the rest of the reading experience. The Javascript instruction will be executed using the Javascript interface, and included in *jsTools.js* in order to remain persistent — as *jsTools.js* will be loaded in each page of the document.

4.4 Streaming buffer and cache

A key point in the development of the client application was the creation of a streaming service, in order to avoid the need to download complete documents and to improve the performance of the service, while avoiding misuse of the documents. While streaming of data is widely used for the transmission of multimedia such as video and audio, a system which transfers the document from the server to a dedicated buffer "in portions" rather than as an entire file seemed to be a suitable solution for this project.

In order to stream a document, the architecture must include a buffer on the client side, in which portions of the document can be stored and later accessed by the e-Reader application. This buffer provides temporary storage from which the document fragment can be accessed, while a decryption process prepares the data for use by the e-Reader application. This buffer will be allocated in memory. The application will automatically delete previously accessed data, making it difficult for a malicious user to download entire publications and later redistribute them. This has been considered a primary goal of the Leaf project, because the business policies that will be implemented depend on preventing users from illegally redistributing content.

The design of the streaming functionality will be done in the future after a deeper study and evaluation of alternative strategies, such as the use of the Networking File System to access remote files segments or DCOM approach access sub-parts of a document. For the current prototype we have simply implemented our own FIFO buffer to decouple the downloading of files from the preparation of them for the reader.

An auxiliary cache space is also provided for two different purposes on the client side. The first reason for the cache space, that will be allocated in the client device's memory, is to temporarily store data that has been recently viewed by the user, in order to enable fast recovery of data when the reader wishes to go back and re-access already viewed content (for example, flipping backward one page or returning to the text after following a citation). The second purpose of the cache space is to store, for users accessing the system with an top-class account, entire documents as part of a premium service offered by the Leaf platform. This option will be implemented in later phases of the project, but it is important that the current design anticipates this future extension.

4.5 Graphical User Interface

In order to provide good usability and to offer the user a complete set of innovative tools a crucial part of the project was the design and realization of a suitable Graphical User Interface (GUI) [68] [69].

The design of the application follows the GUI design guidelines given on the official Android developers website[70]. This design can be divided in four functional parts: Icons, Widgets, Activities and Tasks, and Menus. The design of the Leaf client application follows the principles determined by the Android user interface team, in order to provide homogeneity with the Android OS and to improve the user experience by ensuring that it is consistent with the expectations of an Android user.

A key element that influenced the design of the interface, was the target device that the Leaf project had as our primary target, since the Android OS can be run on various types of mobile hardware platforms, such as telephones, smartphones, and tablets. The first version of the client application has been realized with a layout designed for both tablets and smartphones, covering a range of devices with a diagonal screen size ranging between 4 and 10 inches. Details of the parameters and the values that have been selected for them can be found in subsection 4.5.1.

Following this, subsections 4.5.2 and 4.5.3 will focus on the methods used to handle document navigation and the transitions between different parts of a publication. Finally, subsection 4.5.4 will introduce the user controls that are currently present in the first Leaf client.

4.5.1 Support for multiple screen sizes and resolutions

Including support for multiple monitors sizes, as already stated above, is a key element in every Android application, because of the ubiquitous nature of this (mobile) operating system. This is often considered to be Android's best aspect, because it enables many different devices to utilize the same platform. On the other hand, this is also Android's weakest point, due to the difficulties of adapting to so many different interfaces, in terms of both their technical and physical characteristics.

In the prototype Leaf client software, the team decided to fully support the application on tablets, i.e. devices with a diagonal screen size ranging between 5.3 and 11 inches. Additionally, the application is also designed to be compatible with visualization on smaller screen sizes, such as smartphones. The smallest tested screen size had a diagonal screen size of 3.2 inches.

In order to achieve this level of compatibility with so many different screen sizes, the application has been developed in accordance with Android GUI team directives [70]. According to these directions, the application considers two important variables to handle in the application's manifest[62]: *screen size* and *screen density*.

Based upon the *screen size* variable, we were able to select the standard screen size to support in the Leaf client. This variable has been set to *"large"*, which covers a range of devices from four up to seven inches, or to *"normal"*, for screens larger than three inches but smaller than four and a half inches. Figure 9 shows the different screen size formats supported by Android, as well as how the set of these that the Leaf client can utilize. It is important to note that starting from Android 3.0, the way of handling screen sizes has changed to another metric which relies on screen width rather then diagonal size.

Another important aspect of the graphic appearance of the application is the *screen density*. This parameter indicates the pixel-density, that is the number of pixels within a physical area of the screen. Taken together with the screen size this parameter limits the size of the smallest features that the application can present. Currently the possible values for this parameter are four: *small (at least 426dp x 320dp), normal (at least 470dp x 320dp), large (at least 640dp x 480dp), and xlarge (at least 960dp x 720dp)*. The application currently supports both normal and large densities. It is important to note that *"dp"* is a virtual pixel unit used as a standard when defining the basic metrics in an Android application. The unit *"dp"* represents a physical pixel on a 160dpi screen.

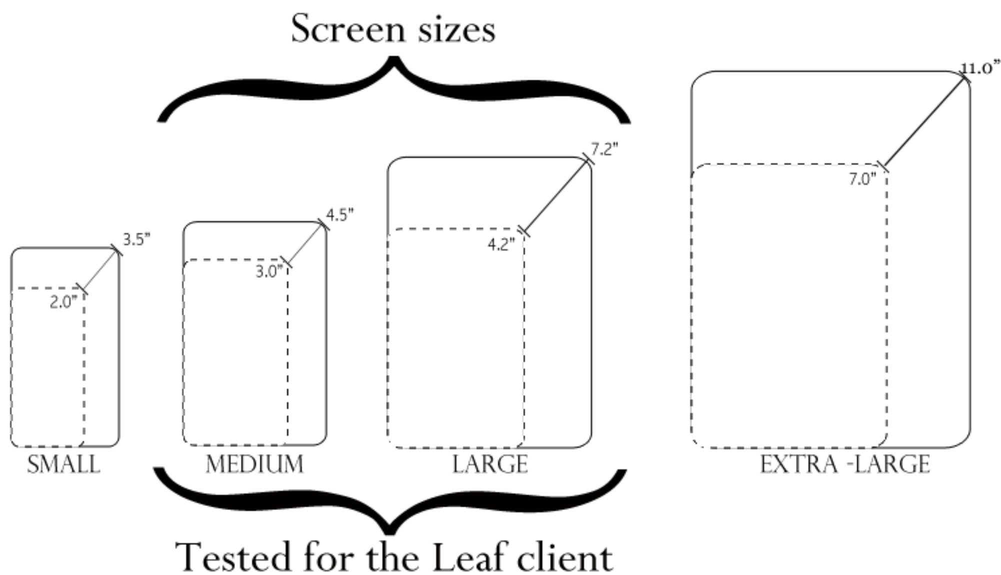

Figure 9 - Screen sizes supported in the Leaf client

4.5.2 Page number handling vs navigation controls

The management of a document's page is a complex part of the client application development, because in a digital publication elements such as display size and font properties may change the number of a page, i.e., pages do not have a fixed page number but rather are numbered based upon how they are rendered on the screen. Even if a page numbering system is not necessary in an application with digital bookmarks and augmented navigation, the principle of offering users a "soft migration" towards new means of entertainment requires including a mechanism to deal with virtual pages.

In order to realize an efficient page-based computing system, during the uploading of documents on the server side of the platform a count is made of all the characters and words that compose a chapter or section of the document itself. The the uploading process saves, together with the document's data, a map in which every chapter's starting point is known in terms of the byte offset into the document file. This data can subsequently be used by the client application to automatically compute the number of pages in a chapter in relation to the display dimensions and, in the case of an increase or decrease in font size, re-compute this number.

However, the need for a page numbering system was been evaluated by the Leaf Project team, together with the resources needed to develop such a system. The conclusion of this evaluation is that at this stage of the project there does not seems to be sufficient reason to include a page numbering system, due the complexity of its implementation and its limited improvement of the user experience.

In order to give the users control over the navigation within the document, the client application includes a "chapters" navigation system. When reading a particular chapter, the user has the possibility to "jump" to the next or previous chapter. This is done using the "*playOrder*" property in each chapter inside an epub document. This

information is stored in a map containing all the information regarding all the chapters. In this way, the buttons that will allow this behavior can point to “next/previous chapter” simply by incrementing or decrementing the index in the current “playOrder”.

During the future development of the Leaf client application, the team will design and implement an alternative navigation system, based on a navigation bar that the user can scroll to move backward and forward in the document. This system, together with multiple digital bookmarks, could provide a good way for users to efficiently follow digital contents.

4.5.3 Transitions

A component to which users are very sensitive, in terms of their reading experience, is the transition between “pages”. Given that the meaning and the importance of a page dramatically changes when considering digital publications, users want to feel a change from one surface to another while reading, as shown by all the major projects in this field, which decided to implement features such as the “leafing page” animation.

The solution adopted until now by every e-reader application is to use animated transitions to give the feeling of changing the page. These animated transitions usually simulate the page “leafing” movement in ways that strongly remind the reader of the action of turning a page. These transitions are achieved, in the majority of the cases, with JavaScript[8] specific plugins, such as booklet[71] and jFlip[72]. An alternative for applications that do not support JavaScript is the use of Adobe’s Flash[73] animations between pages, even if Adobe’s Flash is not fully supported in the Leaf client application. However, in the Leaf project client application this aspect is still not implemented, so the reading experience currently relies on the basic “scrolling down” mechanism, inherited from Webkit[67].

In order to enable transitional effects, the first step will be to introduce an abstract division of each chapter into “pages”. This could be realized by calculating how many characters compose a “page”, considering also the font size and screen dimensions. An alternative method could be calculating the pages in terms of space occupied by spaces, characters, words and images. Once the chapter has been divided into pages, the transition effect between pages can easily be realized using one of the aforementioned JavaScript plugins.

The Leaf team will decide, before starting the development of the application’s next version, which level of usability should be achieved. Depending on this, a suitable transitional effect will be selected and implemented. It might even be possible to implementing more than one effect, then enable users to select their favorite effect (perhaps with different effects for different kinds of documents).

4.5.4 User controls

The first version of the Leaf end-user application contains some controls that the reader may use to better control their reading experience.

Controlling the size of the font is a very user-specific interaction, as it enable each reader to enlarge or reduce the font size used to render the content, according to their needs. To do this, the Leaf client provides the pinch-to-zoom functionality, a tool that is

quite common in current mobile devices application. This functionality allows users to zoom-in or zoom-out by simply "pinching" on the screen (i.e. using two fingers to expand or contract the zoom factor). An increase in the zoom factor (or a decrease) is followed by a subsequent optimization of the text placement according to the new font size. One of the most notable advantages of this approach is that of one ensure that the multimedia content immersed in the text also changes together with the font, then all of the contents will keep the same proportions.

Another reading experience control that has been implemented in the first Leaf client is the *"day/night mode"*. When the reader clicks on the *"day/night mode"* button, this event triggers a change in the font and background colors, switching from *day mode* (black text on a white background) to *night mode* (white text on black background) and vice-versa. Nowadays this function is also very popular in many e-reading applications, because it helps users to obtain the best layout according to their reading desires, in terms of maintaining readability of the contents. This is an example of the importance of CSS3[7] interactions in the application, because the function has been developed just by letting the application change *on-the-fly* some basic values in the CSS of the current content. In future versions of this application, the development team envisions the possibility of having more than just these two modes for this functionality.

A final user control that is implemented in the Leaf client software is the *"back"* function, which enables the user to always return back to the previously visualized chapter, window, or menu. This utility, associated with the generic "Back" button (which is present in each Android device) was implemented by simply catching the pressing of this button and binding it with the universal *"go one step back"* function.

5. Server side applications

The server side of the platform includes several components that are used to upload new publications on the server, as well as the Cookie handling for user authentication. These components have been implemented, during this initial phase of the project, in parallel with the infrastructure development [L3].

A first important part of the server side is composed of a web server & interface which together with Java utilities process the documents which are uploaded. These components and their implementation details will be analyzed in the next sections. Figure 9 gives an overall view of the server side mechanisms.

The second part of this chapter illustrates how user authentication is supported by a specific PHP document and a Cookie management system.

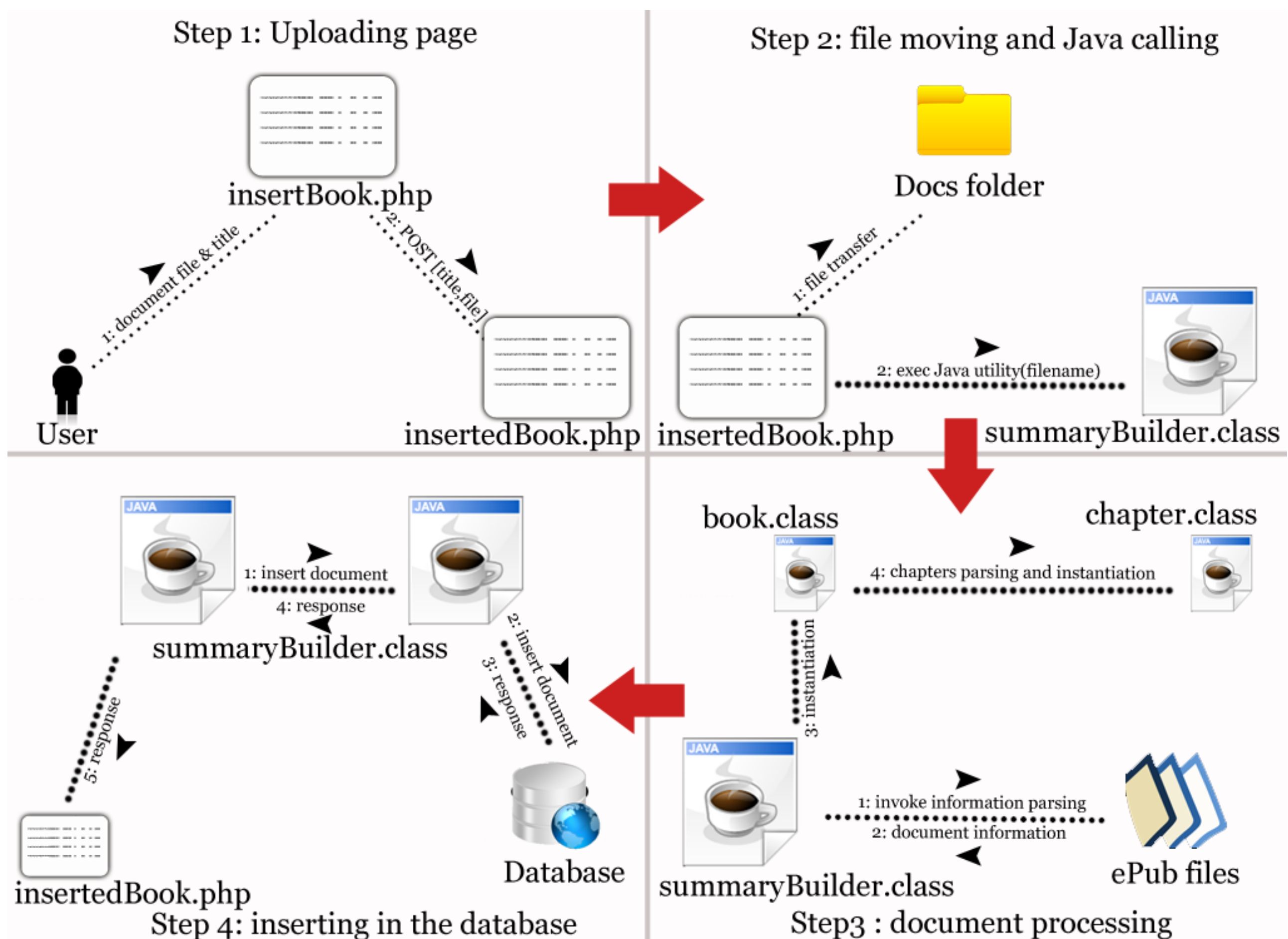

Figure 10 - Server side applications when documents are uploaded

5.1 Web interface

The uploading process to enroll new documents into the platform occurs via two webpages: *insertBook.html* and *insertedBook.php*. In a later stage of the project, these pages will be incorporated into a web platform through which users and the project's staff can perform different actions on both the documents and metadata within the platform.

The first page, *insertBook.html*, is a simple HTML document which presents a HTML form into which the new publication's title and filename are entered. The form's "action" attribute is set to *insertedBook.php*, so that the posted data will be sent to that page. In addition, the attribute "method" is set to "POST", to indicate that data will be sent to the receiving page via an StdIn (Standard Input) approach, which is preferable to the "GET" method, according to HMTL specifications [6] when uploading files to a server. The last attribute, "enctype", is set to "multipart/form-data" MIME[31] type, to indicate that the file will be transferred to the server via a HMTL form.

The second page, *insertedBook.php*, is the receiver for the new publication. It is composed of two sub-parts, the physical uploading of the document and the "exec" execution to complete the enrollment of the document.

The uploading of the document to the server is done through a series of operations. First, the document's name is retrieved from the "POST[]" array and stored into a php variable. Then the page checks whether the file extension is ".epub", and if so, then it changes this file name extension into ".zip". The second step in the process is to check for the file's size and format, in order to check that it is one of the MIME-types that have been declared acceptable to be accepted into the Leaf infrastructure. If all of these operations were successful, then the page checks to ensure that there are no other files in the server with the same name, and if so, then it performs the actual transfer of the file into a folder called "test", in the Apache "htdocs" folder. It is important to note that the file will be transferred as a zip archive, in order for the end user application to browse the document using A Virtual File System (AVFSD) technology [L3]. In the next development phase the responsibility of generating file names will be transferred to the server, so the server will automatically append the generated file name to the list of files to be uploaded by the user.

The second part of *insertedBook.php* is driven by the invocation of the exec command, which is used in PHP to execute an external program using a UNIX shell. The invocation of "exec" calls the the *summaryBuilder* Java class using the document's filename and its title (defined in *insertBook.html*) as parameters, and appending the '#' symbol after the filename to ensure AVFSD compatibility. The description of *summaryBuilder.class* and the other Java classes is given in section 5.2 .

Implementation details of the code, for both *insertBook.html* and *insertedBook.php* pages, can be found in the appendix A starting at page 60 .

5.2 Java utilities

The server side of the Leaf project contains a series of Java utilities to perform a set of essential operations when uploading a new document to the server. These utilities are implemented by four different Java classes: *chapter.class*, *book.class*, *dbInformation.class*, and *summaryBuilder.class*. It is important to note that these classes have been compiled on the developer's machine and then moved to the server, so the original ".java" source files have never been uploaded on the server[1].

5.2.1 chapter.class

According to the object-oriented programming paradigm, every actual object should be represented in the programming environment as an object, in order to better act as its abstraction, with specific operations and properties. The first object type that has been defined in the Leaf project is "chapter", which is the representation of a document 's chapter. In standard epub documents, the standard chapter or section is represented by an HTML/XML document.

The *chapter.class* includes an object constructor, six methods, and a series of attributes: "name", "link", and "playOrder". The first attribute contains the chapter's name or title, the second the position and name of the file, and the third one contains the position in which the chapter appears within the entire document. The values of these attributes are stored into "String" variables, except for "playOrder" that is saved as an "chapter_index" variable as it is an index into a list of chapters.

The main constructor for chapter objects is the method "*chapter(String name, String link, int playOrder)*", in which the values of object's variables "name", "link", and "playOrder" are set to be the three parameters needed to launch this method.

The methods in *chapter.class* can be divided into "setters" and "getters". The setters methods ("setName", "setLink", and "setPlayOrder") can be used by other Java utilities to change the values of these three variables. On the other hand, the "getters" methods "getName", "getLink", and "getPlayOrder" can be used to retrieve these values for a specific chapter object.

5.2.2 book.class

The generic document that can be uploaded and read in the Leaf platform is represented by the Java object "book". Similar to the "chapter" object, the book has several variables to represent different attributes of a document, these are: title, ID, pages, and chaptersList. The first attribute, "title", is represented by a Java String and it set by the user who uploaded the document. This attribute was passed to Java via the "exec" command, in *insertedBook.php* page. The ID of the book is stored in a String variable "id" and it is inherited from the database "id" attribute that is used when inserting a new book row into the table of books in the database. The third attribute, pages, is a Java int variable that contains the number of pages currently in the document; the concept of pages is a complex side of the project (subsection 4.5.2 for details). The last attribute for a book is

[1] This was important as the operator of the service could be a cloud provider or some other form of outsourcing, hence protecting the confidentiality of the source code.

“chaptersList”, a list of all the chapters representing the e-publication; it is implemented as a Java ArrayList containing a series of “Chapter” objects, defined in subsection 5.2.1.

The constructor method for a book object is called “book(String title, String id)” and it creates new book objects, setting the values of variables “title” and “id” as passed in as parameters. The “pages” and the “chaptersList” are initialized to “0” and “null” (respectively) when creating a new book. Later the Java utility in charge of creating new books will automatically update these two variables when inserting all of the chapters into the document.

The setter methods (“setTitle”, “setId”, “setPages”, and “setChapters”) are used to update the value of a book’s variables. The “setChapters” method takes a String pointing to the location of the epub’s “.ncx” file, which contains all the document’s chapters and their links. The method then analyzes the “.ncx” file, parsing all the information related to chapters, retrieving for each element the values of the fields “playOrder” and “src”, that are later stored in the current chapter object.

Getter methods allow the retrieval of information regarding a book object, so other Java utilities can get information about a book, such as title (getTitle), id (getId), pages (getPages), and chapters list (getChaptersList).

5.2.3 dbInformation.class

The main purpose of *dbInformation* Java class is to insert into the database data regarding the document that is being uploaded and to return to the Java class which invoked it a response message.

The constructor method creates a “dbInformation” object which only has the variable “info”, which is a Java HashMap containing pairs of metadata information and labels, such as (“title”, *titlevalue*) or (“filename”, *filenamevalue*). The information contained in this map are: author, title, language, description, publisher, filename, navigation file, cover. The initialization of the “info” map is performed by the class *summaryBuilder.java*. It is important to note that the id is not present in the map, because it is automatically assigned by the database engine due to an auto_increment setting of the id field. It is important to note that this method for assigning IDs will be used only for the first phase. In the future the ID of the documents will be assigned based on the ISBN number of these content, if any, or it will be automatically registered with a new ISBN in case of a new original document. More metadata information will be added in next development phase.

Once the data is retrieved from the map, the dbInformation class initializes the JDBC driver to start a connection with the database, and if this connection is successful, then it inserts the information in a new row in the “books” table. Exception catching and displaying of the reasons for failures have been implemented to avoid data loss and unexpected malfunctions during the document uploading process. If the information is correctly inserted into the database, then the response String is set to “Ok” and the uploading process continues.

Details regarding the database structure, features, and implementation can be found in Sebastian Galiano’s masters thesis [L3].

5.2.4 summaryBuilder.class

The main Java utility on the server side of the platform is the summaryBuilder class, which acts as a controller that triggers actions and creates objects, plus a series of methods to realize the automatic creation of document summaries.

Among the variables of *summaryBuilder.class* we find the title and filename, which come in as String parameters from the PHP "exec" command, invoked by *insertedBook.php* (see section 5.1). In addition, some support variables are used to store temporary file paths, the ".ncx" file location, and the objects to instantiate, such as the current book object. A notable variable is the Java map "metadataMap" that contains all the information that the application is able to retrieve from the document. Later this map will be passed to *dbInformation.class* to fill-in the database entry.

The class has a "main" method that is executed and a series of methods that are launched by the "main" method to perform the document information retrieval and index creation. The necessity of automatic summary creation comes from the absence in .epub documents of a standard form of a table of contents.

The first method that is launched by the main method is called "obtainOPF" and it is used to get the location of the OPF file[32] inside the epub file. This method takes a string as a parameter, in order to know the location of the document inside the server and it automatically searches for the *container.xml* file, inside "META-INF" folder, according to epub standard guidelines. Once the *container.xml* file is found, the method instantiates a series of objects to start perform an XML parse of the file, searching for the "rootfile" element and storing its "full-path" attribute's value into a String variable called OPFpath.

The OPFpath variable is used as the main parameter for the invocation of the "infoParser" method, which needs to know where ".opf" file is in order to start retrieving as much information as possible regarding the document. Similar to the "obtainOPF" method, the "infoParser" method instantiates a series of auxiliary objects in order to parse the ".opf" file, such as a DocumentBuilderFactory, DocumentBuilder, and a Document element. The parsing then continues by identifying certain elements inside the ".opf" file, thanks to the Dublin Core Metadata fields[25] such as "**dc**:title", "**dc**:creator", "**dc**:language", "**dc**:publisher", and many more. Every value identified through the parsing process is stored into the "metadataMap", using a standard default string when these fields were not declared during the document's creation.

The main operation performed in the summaryBuilder.class is the automatic creation of the book summary, since the epub guidelines are not specific about the presence of an index document, leaving it up to applications developers to create their own summary pages. The "createHTMLsummary" method takes the chapters list and the document's folder name as parameters to automatically document. Using a FileOutputStream object, the method starts by creating an HTML document in which it appends a series of "<a>" tags, each one representing a chapter of the book, according to the content of current document's chapters list (see "setChapters" in subsection 5.2.2). When the document is ready, it is moved into the "test" folder in the server.

The last operation performed in the *summaryBuilder.class* consists of calling *dbInformation.class* with the method "fillDBwithInfo". The "fillDBwithInfo" method stores the information contained inside "metadataMap" to the database. The filename of

the summary previously created, which is the same as the original epub, will be stored in the database in the field “filename”.

5.3 User authentication interface and Cookies handling

The user authentication is an essential part of the Leaf platform, since the client application should be able to send the authentication challenge to the server, retrieve a response, and keep the user logged for the entire duration of the reading experience. The server side of this mechanism is illustrated in subsection 5.3.1, while subsection 5.3.2 better explains how the cookies are handled by the server.

5.3.1 User authentication interface with loginApp.php

The server-side of the user authentication is performed by a PHP page called *loginApp.php*. This page is accessed by the end-user application, specifically by *authActivity*, to provide the server with the credentials of the user wishing to access the Leaf platform.

The first part of loginApp.php receives the HTTP POST data and stores them in PHP variables. These variables are then sent to the LDAP[63] connection part. This second part, developed by Sebastian Galiano [L3], configures the LDAP connection parameters and sends the user credentials to the LDAP configuration, which validates these credentials.

In case of a negative reply from the LDAP module, the HTTP response that is sent back to the application client is a code “301” message that is interpreted by the application, which will subsequently ask the user to provide a new set of credentials, since the credentials that were provided are not correct.

If the response is a positive message, HTTP code “200”, then an authorization-granted cookie is issued. This cookie is then stored in the client application’s CookieStore, in the authManager DefaultHttpClient object (see section 4.1.2).

The creation of the Cookie is performed by *loginApp.php* itself. The data that compose this cookie, called “accessCookie”, are: name, domain, expiry time, path, remoteIP, Group, UserName, UserSurname, and UserEmail. Part of these are initialized by the php page, while others (the user-specific ones) are returned from the LDAP query.

After the cookie has been created, it is encoded using MD5 [74] encryption and sent to the client application.

Details related to the cookies’ creation, handling and checking are given in the next subsection. A sample of the “accessCookie” is presented in Appendix B, on page 62.

5.3.2 Cookies handling and the auth_memCookie module

The usage of cookies to control users access and sessions in the Leaf platform is done using an open source module, *auth_MemCookie*[75], that allows the protection of certain web pages within the Apache Web Server environment.

The *auth_MemCookie* module allows the restriction of the visibility of specific pages to users who have been granted a cookie in *loginApp.php*. The *auth_MemCookie* configuration is included in the Apache web server's *default-server.conf* file. Inside this configuration file, we limit the access of certain locations (in this case the folder containing all the publications) to only clients who can present the "accessCookie". If the Apache server is ask to serve a page to a client which does not have a valid accessCookie, then the client is redirected to a customized error page with a "401" HTTP error code (i.e. "401 Unauthorized").

The *auth_MemCookie* module takes advantage of another Apache module, called *memCached*[76]. This second module is an Apache adaptation of the "Memcached" memory caching system [76] used by many websites and databases to optimize memory management, especially in databases with high volumes of data. The connection between the *auth_MemCookie* module and the *memCached*[76] requires that the user first has to register causing the cookie value to be stored by the *memCached* module — so that it can subsequently be available to be checked when there is a future connection by this client.

The current configuration for *auth_memCookie* enables the two modules to communicate between themselves, and specifies the details regarding the access of users to certain parts of the server. Details of this configuration are given in Appendix C.

In future implementations of the client application, the team will consider the use of *auth_MemCookie* "Groups" functionality, to grant access to a certain set of contents only to authorized groups of users (i.e. premium users).

6. Analysis of the results

This chapter analyzes the results obtained by the development of the first client for the Leaf project, on both functional and performance aspects. The first section of the chapter explains the reasons behind the different decisions made during the application planning, with the help of figure 11. The following sections, 6.2 and 6.3, respectively present to the reader the performances of the application (with further analysis on these data) and the functionalities implemented in the application, compared to table 1 (which appears in subsection 2.3.1).

6.1 Implementation decisions

This section presents a summary of the implementation decisions that have been made during the development of the Leaf platform client application.

Figure 11 describes these decisions using a graph, in which the “building blocks” each represent a critical part of the application that has been developed for the chosen solution (this is shown in the light-green box inside the dark-green block). In contrast, the light-red boxes symbolize the alternative solutions that the team decided to not implement, at least in this phase of the project. Finally, light-yellow boxes depict those elements that the team will develop in a future phase.

The building blocks in this graph are linked with arrows, which abstractly illustrates how the key decisions made during the design and development process had a direct influence on future steps. A good example of the dependency of each step on the previous steps is the implementation of a specific kind of cookie, which is a consequence of the decision to have an authorization strategy *not* based on the basic HTTP authorization. The authorization strategy itself became an implementation decision once the team decided to adopt a streaming service rather than an offline mode.

Is important to note that the graph in figure 11 illustrates only the implementation decisions that were critical for the client application development. For this reason, choices related to the contents (e.g. books or magazines), to the server architecture (e.g. which load balancer solution to implement), and to the business area (e.g. which business model to apply) are not represented in this graph.

Finally, each building block contains an indication of which section of this thesis better explains the specific decision. Reading the appropriate section gives the reader an overview of the starting conditions, reasons, and results of every crucial step that the team made during the development of this application.

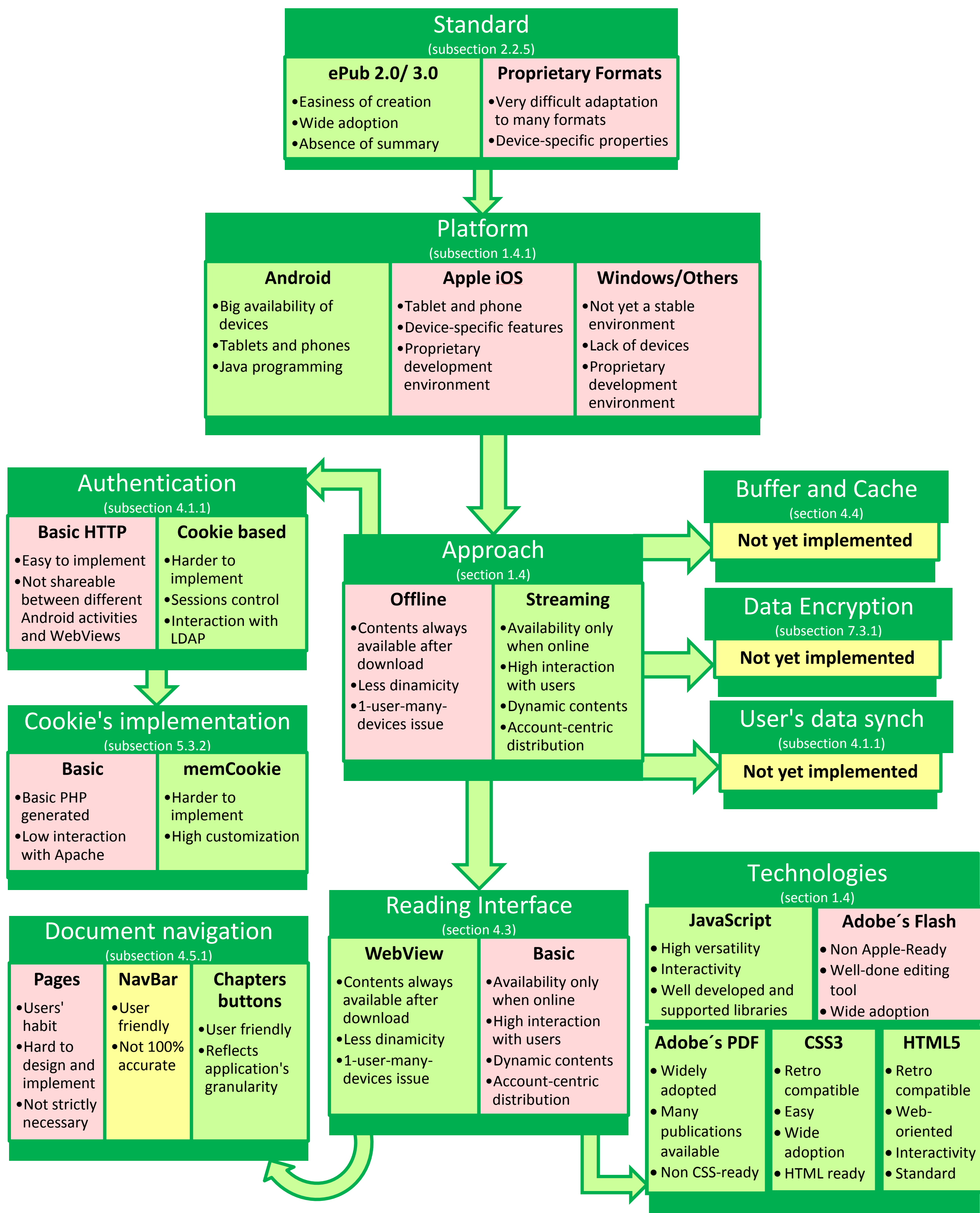

Figure 11 - Decisional path for the design and development of the client application

6.2 Empirical results

During the development of the Leaf client application, the components that the team created have been constantly evaluated in terms of their performance and reliability. The purpose of this section is to show the metrics that characterize the final application, to have a more precise picture of the application's features.

The performance tests have been conducted using Traceview[77], a Java profiler tool that is part of the Android SDK suite. This tool can be used to monitor the timing of several parts of an application. Using Traceview, the developer can visualize as a series of timelines all the processes running between the start and end points, selected by the application's author. For every process, we were able to extract the timing (expressed in milliseconds), dependencies, and the full tree of related processes.

The reliability tests have been performed and elaborated starting from manual human invoked tests performed on several different Android devices, to test error rate, usability, and an evaluation of the quality of the application's different components.

The results presented in this section will be compared, in the next section, with the values obtained from alternative applications' equivalent components.

Table 3 - Application's empirical data on components' speed tests
(note the decimal comma indicates the radix point)

Android Activity	Application's component		Java class	WiFi (msec)	3G (msec)
LeafClient Activity	1	Application started	leafClient.java	137,6	139,6
Authorization Activity	2	Sending credentials and LDAP authorization - *successful*	authActivity.java + authManager.java	875,2	956,2
	3	Sending credentials and LDAP authorization - un*successful*	authActivity.java	810,3	886,4
Library Activity	4	Data retrieving from database	dbConnector.java	187,1	300,8
	5	Buttons dynamic creation	libraryActivity.java	20,5	20,8
Reading Activity	6	HTML summary displaying	readingActivity.java	87,5	107,5
	7	Chapter file loading - *ePub, no images*	readingActivity.java	734,1	942,7
	8	Chapter file loading - *ePub, images*	readingActivity.java	1000,2	1552,8
	9	Background/font colors change	readingActivity.java	3,4	10,1

It is important to specify the environmental variables that characterized the profiling phase, as well as the testing methodology. The values, expressed in milliseconds, represent the average results of five different measurements taken for each application's element, to avoid presenting inconsistent data. This means that for each of the measurement, the team manually executed five times the profiling with Traceview, and out of the five obtained results, we calculated the average value. In this case the choice of using the average value instead of the median value was made because the first one can better represent the overall performance of the application, being an artificial number that represents all the five measurements. With later developments of this project, the team will continue testing the application's performances, aiming to improve the speed and efficiency of the client (adopting an automated way to do this, in order to have more measurements for every single item to test).

The device used to run the tests is a Samsung Galaxy Tab, with Android v. 2.3.3 installed. The conditions of this machine were typical, in terms of memory usage, network adapter traffic consumption, and number of active applications.

To test the change in performance with different types of connectivity, we repeated the measurement using both WiFi and 3G networks. The WiFi connection was to a WiFi 802.11n access point, connected to a 24Mbps ADSL line. The 3G trials have been performed with full connectivity on a HSDPA 7.2Mbps network.

Finally, when measuring the speed of data retrieving from the server, the number of different documents stored in our memory was six, all of them typical-length ePub 2.0 books (specifically 270Kb, 217Kb, 1,5Mb, 274Kb, 1,7Mb, and 541Kb).

6.2.1 Analysis of the empirical results

As can be observed from table 3, the results obtained from the profiling activity are quite comparable, although the WiFi connectivity gives a better time-response on each activity as compared to the 3G connectivity. Despite this, the difference between the two network connections is never sufficiently different that it would justify the introduction of major changes to improve the performance with 3G connectivity.

In the majority of the activities, we can observe that when they are ran over a 3G network, they tend to be slightly slower than when using WiFi link. The relation between the two different situations can be observed in Figure 12, which shows the average relation between measurements, as well as the peaks in which we can notice a remarkable similarity.

Figure 12 shows the trend of the application when executed over both WiFi and 3G connections. The Y-axis indicates the time, expressed in milliseconds, it takes the application's components to run. The X-axis of the graph represents all (nine) of the application's components that have been tested. These components have been put in "calling" order, so the first (measurement number "1") is the first component that is called when the user opens the application. The second measurement, related to sending the LDAP[63] authentication request, is the second activity the user performs and so on until the last one, which represents a typical user interactivity control such as the background/ font colors mode. This order gives an idea of the "standard usage" trends during a regular user experience with the application.

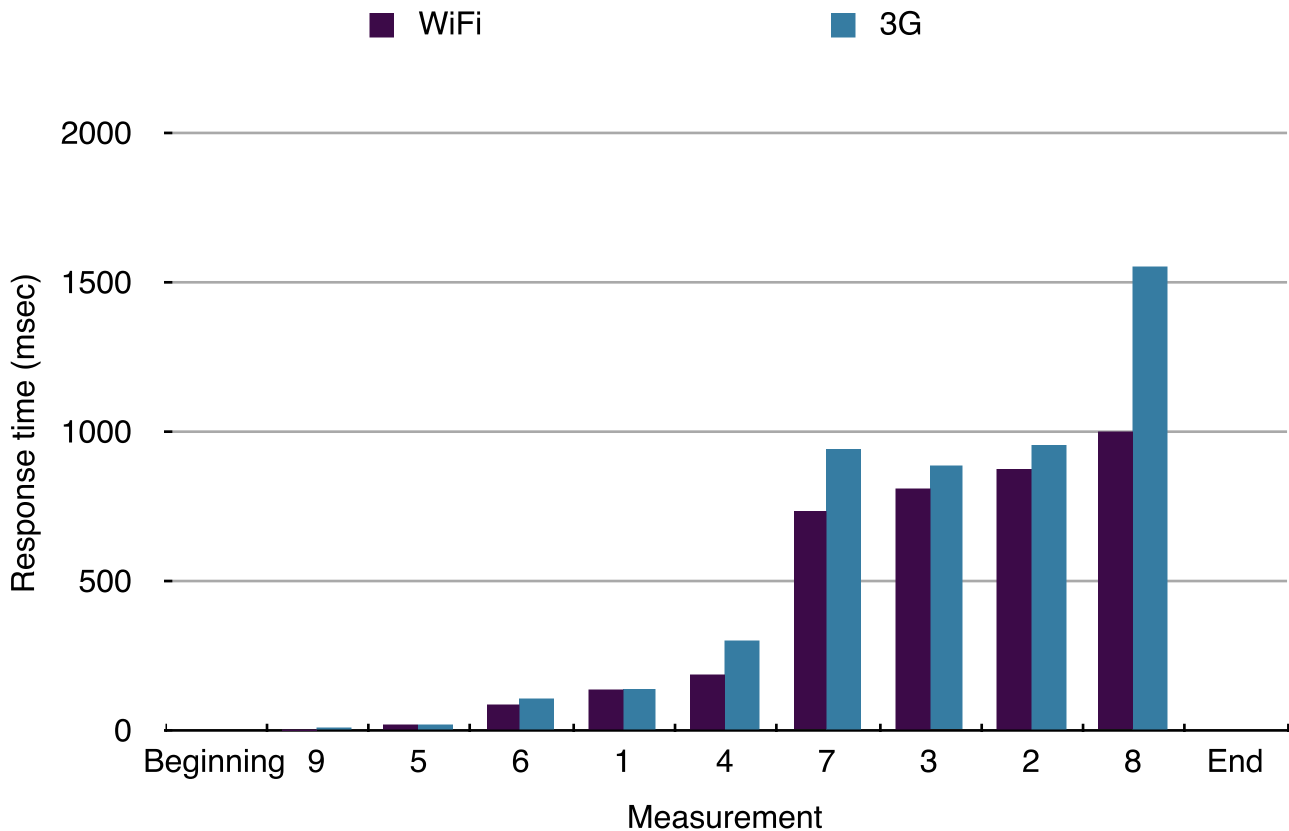

Figure 12 - Relation between WiFi and 3G in the application
(*Results are shown in order of WiFi speed to show how the difference between connections grows depending on the amount of data transferred)*

The results of these measurements confirmed are conform what the team was expecting, that is a fixed ratio of the performance of WiFi and 3G modes, with an average ratio of WiFi(msec)/3G(msec) = 0,87. The values that have not been included in this calculation are the so called "peaks", in which we observe a different behavior — when there is a large amount of data to be transferred and this ratio represents the average difference in throughput for these two types of links.

Measure number "5" shows very similar values for both connection types, due to the fact that they refer to a task which is independent of the connectivity. In fact, the measure presents the speed of automatically creating "buttons" to access the documents, after having already retrieved the titles and references from the database.

On the other hand, measurements "7", "8", and "9" have very different values, demonstrating a significant difference in performance. In the first two cases, when downloading a document with or without images, this appear to be reasonable, since the download is a task that may requires more time than the other components, due to the bigger amount of data involved (this is also demonstrated by the highest values in the graph). Moreover, measurement "9" consists of sending a CSS rule to the document, which involves a re-definition of the document's style, as well as very fast page refresh. This is also a task that involves significantly online operations, so the WiFi performance was expected to be higher than the 3G.

6.3 Comparison with existing results

This section's primary goal is to compare and analyze the measurements obtained from the Leaf end-user application with the results of other alternative software.

A real comparison of measurements with existing applications would have been very interesting. Unfortunately, the software developers of such e-reading tools do not disclose comparable measurements of their applications for marketing reasons. Given this, it is not possible to establish a direct comparison between the Leaf client and other applications, specifically the ones described in subsection 2.3.1.

As a final summary of the Leaf end-user application, we have added the details of our client software to our table 1 (shown on page 13), to directly compare it with the most popular comparable applications currently in the market.

Table 4 - Comparison of features in different e-reader clients, revised

*(*Clients with more features are showed in the top)*	Supported Formats (Non-proprietary)	Supported Formats (Proprietary)	Data Synchronization	Day/Night Mode	Custom Font size	Virtual Store
Apple's iBook	txt, ePub, html, Open eBook, pdf	MobiPocket, FictionBook, DjVu, eReader, azw, TomeRaider	YES	YES	YES	YES
Kobo Ebook Reader	txt, ePub, html, Open eBook, DjVu, pdf	MobiPocket, FictionBook, eReader, azw, TomeRaider	YES	YES	YES	YES
Amazon's Kindle 3	txt, html, pdf	MobiPocket, azw	YES	YES	YES	YES
OpenLibrary	txt, ePub, html, DjVu, pdf	azw, DAISY, MobiPocket	YES	YES	YES	YES
pressDisplay.com	txt, ePub, html, Open eBook, pdf	---	YES	YES	YES	YES
Barnes&Noble Nook	txt, ePub, pdf	eReader	YES	YES	YES	YES
Lexcycle Stanza	txt, html, ePub, pdf	rtf, doc, eReader, Lit, azw	YES	YES	YES	NO
The Leaf client	**txt, html, ePub, pdf**	**---**	**NO**	**YES**	**YES**	**YES/ NO**
Adobe Digital Editions	ePub, pdf	Flash	YES	YES	YES	NO
FBReader	txt, ePub, html, Open eBook	FictionBook, MobiPocket, rtf	NO	YES	YES	NO
Aldiko Ebook Reader	ePub, pdf	---	NO	YES	YES	NO

Table 4 shows the features of the Leaf client application, comparing it with popular applications currently in the market. Despite identifying this row as Leaf, the team has not yet decided upon a final name for the software.

The second and the third columns show supported formats (both proprietary and non-proprietary formats), indicating those ebooks format that have been tested on the application. Adobe's PDF, txt and HTML have been tested and found to work properly in our application, i.e. without any problem during the reading experience. The ePub format, which in this table is more specifically identified as ePub 2.0, is the target format on which the team worked, and currently represents the standard for our distribution platform.

The fourth and the fifth columns show, respectively, user's data synchronization and night/day mode, i.e. the possibility of easily switching from a white background with black font to the opposite combination. User's data synchronization is still not implemented, meaning that bookmarks and other functionalities related to a specific user are not part of the current application. In regards to the "day/night" mode, it is already present in the Leaf client, as described in the third chapter of subsection 4.5.4.

The last two columns state that custom font sizes and virtual stores are part of the Leaf end-user application. The possibility of changing font size is given to the user (as described in subsection 4.5.4, second paragraph) thanks to the "pinch-to-zoom" function, that allows users to easily control text's size. The "virtual store" is function that has been implemented in the application in the form of "virtual library". This is more than a digital shop, as the user can access the collection of documents from the application itself *without* needing to download the publication from a third party and then open it with the Leaf client. Even if accessing the collection of available contents "in-app" and online is a possibility for the user, the team did not develop functionality for purchasing contents. For this reason it can not be considered as proper virtual store, this remains for a subsequent development phase when the accounts policies will be be implemented.

Comparing the client we developed for the Leaf project with the ones already in the market, we can observe that the our first prototype is positioned on top of the second half of the table. Given all the assumptions valid for table 1, this result represents a good base for the next version of the client, which will have to compete with the more well-known applications also on performances and usability.

7. Conclusions

In this chapter the focus is on what has been achieved thus far and what remains for future development phases. The first section will explain which the objectives that have been successfully implemented for both client and server applications, remarking on how the proposals in the original project plan have been respected or changed during the implementation process. Subsequently section 7.2 describes the project plans for the future, focusing on the parts of the project that still need to be implemented and explaining evaluations that will be used as the basis for implementation choices. In the end of the chapter a short discussion explains which limitations the author had to overcome in order to successfully achieve the thesis project's goals.

7.1 Achieved goals

The first phase of the Leaf project involved the development of both client and server prototypes. The following list describes the goals that have been successfully achieved during this first phase:

- Theoretical preparation, Literature Study, Android programming training and study, and evaluation of current solutions in the market. *(weeks 1-5)*

- Development of a standalone e-reading client application's prototype for Android. *(weeks 6-11)*

- Development of server-side strategies and further implementation of these applications in the server *(in collaboration with Sebastian Galiano). (weeks 12-15)*

- Creation of the Leaf project temporary homepage and web pages to upload new documents on the server. *(week 10)*

- Adaption of the client application's prototype to get data from the server side and read documents in a streaming-like approach using basic HTTP. *(weeks 12-15)*

- Integration of multimedia support in the reading application, making it possible to read contents created with multiple technologies, including Javascript[8], JQuery library[9], HTML5[6], and embedded images and videos. *(weeks 12-14)*

- Creation of an authorization mechanism based upon LDAP[63] to check users' credentials, grant them access to the contents they are authorized for, and maintaining users authenticated throughout their reading experience. This component still has to be optimized, as described in subsection 4.1.1. (*weeks 17-24)*

- Adaptation of both client and server applications to run with several types of non-standard epub documents and predicting anomalies in the file types and structures. *(weeks 14-15)*

- Design of GUI elements to integrate in the first client application, such as buttons, images, and logos *(weeks 15-16)*

7.2 Limitations of the current solution

During the development of the client and server applications for the Leaf project there were some time and resource constraints, that led to few limitations in the prototype application.

First of all the development of the ePub 3.0 standard, currently still in progress, represents a changing target that will influence the final application developed for the project, i.e. the client software and parts of the server side utilities. The preliminary indications given by the recently released draft specifications have been used as our main guidance during client application development in order that the application can be easily adapted to the ePub 3.0 standards when they are officially released[35].

In regard to the client application, some of its components have only limited functionality. An example of such limitations is the lack of a well established page numbering system, which is not typically available for digital publications navigation, but the (optional) presence of this feature could positively influence the application's success. In addition, small features such as brightness control and font-size increasing were part of the original design, but they have not been implemented; nevertheless, these features will probably be part of the first iteration during the next phase of the development.

Looking from the network and security point of view, there are still some important features that should be implemented, such as the usage of HTTPS in place of basic HTTP (with mutual authentication of both client and server), use of a more secure database connection, and encryption of all transmitted data. More details regarding these features will be described in section 7.3,

7.2.1 Authorization system's limitations

In subsection 4.1.1, the last paragraph introduced some technical problems related to the authorization system, which is currently based on a combination of two modules, *auth_memCookie*[75] and *memCached*[76]. The setup for these two components on the server, and the way they operate, can be found in section 5.3.

The main limitation regarding this mechanism is related to the connection between these two modules, which is currently not working properly. At the current stage, the *auth_memCookie* module should check, every time a user tries to access the protected area of the server, if the authorization cookie for that user has been created and if it is valid. To do this, the module should connect to the second one, *memCached*, and check if in its memory that such a cookie has been stored and that the presented cookie matches this stored cookie.

After a very deep analysis of the server, Apache, and these two modules, the team was unable to find the reason why this connection is not producing the expected result, that is finding the current user's cookie. The first part of this authorization mechanism, has been successfully tested with different diagnostic tools. This means that when a user logins into the web platform, his/her access is registered by the *auth_memCookie*[75] and a well-formed cookie is correctly stored in the *memCached*[76] for the user session handling. However, when the user later access the server the cookie is not successfully compared.

Unfortunately the documentation and support for this kind of problem, on a such specific issue, is insufficient to allow the team to determine the problem's causes. To overcome this issue, the team has already planned implementing a similar solution, which relies on a different technology, that should allow the same mechanism to keep track of users sessions. Details of this alternative solution are presented in section 7.3.

7.3 Future work

The next iteration of development of the "Leaf Project" end-user application will involve improvements to existing features, as well as the implementation of new utilities and technologies. The team working on the Leaf project is also looking forward to see the future work from the other actors involved in the e-reading market, such as editing platform creators, device manufacturers, an teams working on the development of web and e-publications standards. The following two sections describe work planned for the client application and some though regarding future technologies that may be relevant to the future development of the client application.

7.3.1 Future work on the client application

Among the the points that will be reviewed, restructured, and expanded, first of all we must adapt the application to the final ePub 3.0 structure. Currently the application provides full support for ePub 2.0, but with the official standardization of ePub 3.0 format, the application should be able to handle contents organized according to the new standard.

With regard to the networking and security aspects of the client and server, there are several features to be implemented in the next iteration. First of all, data encryption should be implemented in a secure but light weight way, to avoid misuse of the data transmitted without reducing the speed or the reliability of the wireless connection.

In addition, the usage of HTTPS rather than the basic HTTP would offer the application improved security, especially in terms of data protection, noted that earlier we should perform mutual authentication of the client and server — to prevent impersonation of either. Directly connected to this issue, the authorization mechanism has to be re-designed due to the problems described in subsection 7.2.1. To overcome this problem, the team plans to take advantage of another module for Apache, that should perform the same kind of activity of *auth_memCookie*[75], but in a more reliable and well-documented way. The reason why this module is not currently installed on the server, is that in order to use it the team has to review the Apache web server configuration, a task that will be done in the next phase of this project.

A crucial point to be evolved in the next version of the application is the connection to the database, currently achieved thanks to a specific Java class, *dbConnector.class*, illustrated in section 4.2.4. This solution is considered to be non-optimal, because it consists of having the database information and queries in a Java class, installed in the client device. The solution that the team is planning to adopt is to query a php controller, hosted on the server, which will be in charge of receiving queries from the application and forwarding them to the database, **without** having the database credentials in the end-user environment.

The implementation of a customized buffer and cache is also a key element of the next version of the application. Having these two components under the control of the development team could give the project the tools needed to better adapt the streaming approach.

The navigation system is currently composed the the buttons used to navigate among different chapters. As stated in subsection 4.5.2, this part of the application will be enriched later with the implementation of digital bookmarks (already part of the plan for user's data synchronization) and a document scrollbar to move within the document.

Regarding future utilities that will be provided to users, depending on the commercial requirements of the project there will be a great number of functions that could be added to the application, such as an offline version of the application, social network integration, user's data control, and integration with an advertising system.

The "offline version" functionality will allow users, such as those with a premium account, to download contents into their device, in order to be able to read publications even when offline. This feature must include appropriate data encryption, to avoid the illegal re-distribution of data through other channels. The offline version should be implemented in such a way that it will automatically synchronize the user's data as soon as the device comes back online.

The integration with social network(s) is a desired functionality that will be implemented in two separate parts. Concerning the connection with todays' popular social networks, the Leaf Project team is in contact with the development team of Treadmill, a service which has the aim of creating a network of readers and contents around the most famous social networks via a specific API. Additionally, the Leaf Project client application will include its own "sharing" tools to enable users to interact at different levels with the application itself and with their social networks.

The future developments of the client application will involve, depending on commercial discussions and on the data gathered following the product's launch, its own social network. The team is now in contact with the firm behind "Bananity" [78], a recently launched social network, to investigate the possibility of a collaboration between these two services.

User's data manipulation is a delicate step that will be deeply analyzed before starting any implementation. This manipulation would be based upon retrieving users' data concerning their readings and behaviors, and using this data for commercial or statistical purposes. Despite the great possibilities in this direction given by the usage of mobile devices, such as learning users' favorite places, behaviors, and contents, the legal aspects of utilizing this data (other than to improve the service provided to the users) represents a barrier that should be studied extensively before embarking on any usage of this data.

Finally, an important feature that the team is currently studying is integration with an advertising system. Our current thinking is that this will be implemented by renting phrases, rather than spaces, to advertising companies. In this way the reading experience will not suffer from any disturbance in the already limited reading (display) space or time, but rather the user might use the advertising functionality as a utility — rather than an annoying distraction. This feature will be implemented, on the user side, as a menu that scrolls down when the user clicks on a phrase or word in which he/she is interested, such

as the name of an airline or of a food. A specific API will be developed for this purpose, to provide advertisers with an easy interface to rent "phrases" and link them with their commercial sources.

In the future iterations of the Leaf Project end-user application, there will be improvements to the server-side utilities, such as an improved uploading system, user and account management, and a new web interface. These components will depend significantly on commercial discussions.

The GUI design [50] [51] is another element that will be completely restructured during the next stage of the Leaf project, optimizing it according to the new functions that the team decides to include in the next version of the application. The theme will probably feature a large main window, representing the user's "homepage", with all the resources available for the user, classified and organized into different categories. Along the sides of that window there will be a list of buttons that the user may use to access "Recent history", "Favorite readings", and "Suggested publications". In addition to these categories, a complete set of tools will be provided to enable users to perform activities during their reading, such as adding/removing bookmarks, highlighting text, and changing the document's orientation. Of course in the case of appearance changes, the GUI should be sufficiently advanced that it can react flexibly in order to always offer the best reading experience to the user.

The contextual and functional menus of the application will be re-designed to offer users shortcuts to the technical functions that are normally present in hidden menus and that are not directly related to the reading material, such as access to the "general settings" or to invoke manual synchronization of data.

An advanced graphical feature that will be implemented in later phases of the project will be a means for the user to collect different articles and summaries of the latest relevant data from the user's favorite readings, creating a sort of "customized" journal page from which the user can potentially access interesting information extracted from various sources. This feature represents an example of the benefit that users would have through the Leaf platform.

7.3.2 Future technologies

Some external factors will influence the direction of the project, specifically the different solutions introduced in this market by different companies. Recently, firms such as Koobits[79] or Platify[48] have launched platforms in which, similarly to the Leaf project, users can upload and read different kinds of publications, taking advantage of modern interactivity elements. Looking at the development of these projects will have a double impact on the Leaf: on one hand their work can be observed to gather feedback and indicators of their directions, on the other they can be viewed as potential future competitors.

What will be crucial for the future of the Leaf project is the development of new technologies related to the field of e-reading. In this direction, we are following to the major developments of both hardware and software.

The development of eInk screens surely represents one of the most significant key-features that will affect the e-reading market in the near future. The integration of

electronic ink into mass-production tablets could be a major factor to unify e-readers and tablets.

In addition to hardware innovations, there will be many software technologies that will affect and increase the success of e-reading solutions. Many of these solutions come from the Web sector, and are highly related to e-reading thanks to the ePub structure (specifically the fact of utilizing HTML documents). The most important innovations that will be introduced in this market segment are Google's DART[80] and the Mozilla Web API[81], projects with the aim of enabling browsers to exploit device functionalities. These types of coming products will help content creators add more interactivity in their creations, which is one of the most important motivations behind the Leaf project.

Content creators will also benefit from new editing tools to compose publications and web contents, enabling them to achieve an interactivity level that could lead entertainment and information to a new phase. This area will be open for many new possibilities. In this field, tools like Adobe's Edge[82] and the Sigil editor[83] can play a leading role.

Despite the fact that an editing tool is not currently part of the Leaf platform, future phases of the project will develop such a tool as a natural evolution of the web interface that today enables users to upload their documents. Such an editing tool will most likely be implemented as an online tool and directly integrated into the Leaf website, in order to enable authors to easily create new content.

Depending on the business path(s) that the project will take, as well as the technologies that will be implemented, the Leaf project will try to evolve from a distribution platform to a more comprehensive platform, in which users will enjoy various forms of content on different devices and in various environments, while enabling them to subscribe to specific content creators, editorial teams, and/or distributors. In the same way, authors will be able to collaborate with designers and artists to embed multimedia interactive elements in their creations, periodically publish new contents, and decide in which way(s) to earn money on these contents.

References

1. Michael Hart, Gutenberg Project description [www] http://www.gutenberg.org/wiki/Gutenberg:The_History_and_Philosophy_of_Project_Gutenberg_by_Michael_Hart, Last access on 2011-02-24

2. Newspaper Association of America, Report on total paid circulation in the last 50 years [www] http://www.naa.org/TrendsandNumbers/Total-Paid-Circulation.aspx USA, Last access on 2011-02-24

3. Federazione Italiana Editori Giornali, Italian newspapers paid circulation 2009/10, 2011-01-28 [www] http://www.fieg.it/documenti_item.asp?page=1&doc_id=198 ITA, Last access on 2011-02-24

4. National daily newspaper circulation for January 2011 in UK, 2011-02-11 [www] http://www.guardian.co.uk/media/table/2011/feb/11/abcs-national-newspapers1, Last access on 2011-02-24

5. Android Project, Android OS reference [www] http://developer.android.com/reference/packages.html, Last access on 2011-02-24

6. W3C, HTML5 Specification [www] http://www.w3.org/TR/html5/, Last access on 2011-02-24

7. W3 Schools, Css Reference [www] http://www.w3schools.com/css/css_reference.asp, Last access on 2011-02-24

8. W3 Schools, Javascript Reference [www] http://www.w3schools.com/jsref/default.asp, Last access on 2011-02-24

9. JQuery Project, JQuery definition [www] http://api.jquery.com/category/core/, Last access on 2011-02-24

10. Prototype Core Team, Prototype Javascript Framework Guide [www] http://www.prototypejs.org/learn, Last access on 2011-02-24

11. International Digital Publishing Forum, ePub 3 Overview, 2011-02-15 [www] http://idpf.org/epub/30/spec/epub30-overview.html, Last access on 2011-02-24

12. Apache, Subversion, [www] http://subversion.apache.org/, Last access on 2011-10-09

13. Millennial Mobile Mix, 15th July 2011, [www] http://www.bgr.com/2011/07/15/millennial-android-tops-mobile-os-usage-for-7th-straight-month-iphone-still-top-device/, Last access on 2011-10-09

14. Gartner's official Mobile OS report, April 2011, [www] http://www.gartner.com/it/page.jsp?id=1622614, Last access on 2011-10-09

15. Samsung, Samsung's official website, [www] http://www.smasung.com, Last access on 2011-10-09

16. High Tech Computer Corporation, HTC's official website, [www] http://www.htc.com, Last access on 2011-10-09

17. Sony, Sony's official website, [www] http://www.sonyericsson.com/cws, Last access on 2011-10-09

18. Motorola, Motorola's official website, [www] http://www.motorola.com, Last access on 2011-10-09

19. Android Project, Android SDK definition [www] http://developer.android.com/sdk/index.html, Last access on 2011-02-24

20. Apple's official website, [www] http://www.apple.com, Last access on 2011-10-09

21. W3 Schools, XML definition [www] http://www.w3schools.com/xml/default.asp, Last access on 2011-02-24

22. Adobe Portable Document Format v1.7, November 2006 [www] http://www.adobe.com/content/dam/Adobe/en/devnet/acrobat/pdfs/pdf_reference_1-7.pdf, Last access on 2011-03-05

23. Wikipedia, Softbook [www] http://en.wikipedia.org/wiki/SoftBook, Last access on 2011-02-24

24. International Digital Publishing Forum, Open eBook specifications [www] http://www.openebook.org/oebps/oebps1.2/index.htm, Last access on 2011-02-24

25. Dublin Core Metadata Initiative, Official specifications [www] http://dublincore.org/specifications/, Last access on 2011-02-24

26. International Digital Publishing Forum, Open Publication Structure 2.0.1, 2010-09-04 [www] http://idpf.org/epub/20/spec/OPS_2.0.1_draft.htm, Last access on 2011-02-24

27. Mobile Wiki, PalmDOC format [www] http://wiki.mobileread.com/wiki/PalmDOC, Last access on 2011-03-07

28. DjVu.org, DjVu official documentation [www] http://djvu.org/resources/, Last access on 2011-03-07

29. MobiPocket official reference site, MobiPocket documentation [www] http://www.mobipocket.com/dev/, Last access on 2011-03-07

30. Amazon.com, Amazon's AZW document format [www] http://kindle.s3.amazonaws.com/Kindle_Users_Guide.azw, Last access on 2011-03-07

31. N. Freed and N. Borenstein, Multipurpose Internet Mail Extensions Part One: Format of Internet Message Bodies, November 1996 [www] http://www.ietf.org/rfc/rfc2045.txt?number=2045, Last access on 2011-03-06

32. International Digital Publishing Forum, Open Packaging Format 2.0.1, 2010-09-04 [www] http://idpf.org/epub/20/spec/OPF_2.0.1_draft.htm, Last access on 2011-02-24

33. International Digital Publishing Forum, OEBPS Container Format 2.01. 2010-09-04 [www] http://idpf.org, word document on the main file, Last access on 2011-02-24

34. J. Dlugosz, ZIP2 reference [www] http://www.dlugosz.com/ZIP2/index.html, Last access on 2011-03-07

35. International Digital Publishing Forum, ePub 3 Content Document, 2011-02-15 [www] http://idpf.org/epub/30/spec/epub30-contentdocs.html, Last access on 2011-02-24

36. International Digital Publishing Forum, ePub 3 Publications, 2011-02-15 [www] http://idpf.org/epub/30/spec/epub30-publications.html, Last access on 2011-02-24

37. International Digital Publishing Forum, OEBPS Container Format 3.0, 2011-02-15 [www] http://idpf.org/epub/30/spec/epub30-ocf.html, Last access on 2011-02-24

38. International Digital Publishing Forum, ePub Media Overlays, 2011-02-15 [www] http://idpf.org/epub/30/spec/epub30-mediaoverlays.html, Last access on 2011-02-24

39. Erik Dahlström, SVG [www] http://www.w3.org/TR/SVG11/, Last access on 2011-02-24

40. Amazon Inc., Amazon.com [www] http://www.amazon.com, Last access on 2011-02-24

41. Apple, iBook main page, http://itunes.apple.com/us/app/ibooks/id364709193?mt=8, Last access on 2011-03-05

42. Barnes&Noble, Nook Platform website [www] http://www.barnesandnoble.com/NOOK/index.asp, Last access on 2011-02-24

43. Kobo, KoboBooks.com [www] http://www.kobobooks.com/, Last access on 2011-02-24

44. NewspaperDirect Inc., PressDisplay.com [www] http://www.pressdisplay.com/pressdisplay/viewer.aspx, Last access on 2011-02-24

45. Zinio LLC, Zinio Homepage [www] http://www.zinio.com/, Last access on 2011-03-07

46. The Internet Archive, OpenLibrary Project [www] http://openlibrary.org/, Last access on 2011-02-24

47. 24Symbols, official homepage [www] http://www.24symbols.com, Last access on 2011-10-09

48. Paul Biba, Platify's service analysis [www] http://http://www.teleread.com/paul-biba/new-swedish-ebook-platform-platify-opens/, Last access on 2011-10-09

49. Amazon.com description, Kindle App for Android, [www] http://www.amazon.com/gp/feature.html/ref=red_lnd_shrt_url?ie=UTF8&docId=165849822, Last access on 2011-10-09

50. Wikipedia, Digital Rights Management [www] http://en.wikipedia.org/wiki/Digital_rights_management, Last access on 2011-03-06

51. FictionBook, FictionBook 2.1 Schema [www] http://www.fictionbook.org/index.php/Eng:XML_Schema_Fictionbook_2.1, Lat access on 2011-03-07

52. Open Handset Alliance, Official Website, [www] http://www.openhandsetalliance.com/, Last access on 2011-02-24

53. Android Project, Android Open Source Project official documentation [www] http://source.android.com/, Last access on 2011-02-24

54. Official Java Reference Guide, Just In Time compilation [www] http://java.sun.com/developer/onlineTraining/Programming/JDCBook/perf2.html#jit, Last access on 2011-03-06

55. D. Bornstein, Dalvik Virtual Machine [www] http://www.dalvikvm.com/, Last access on 2011-02-24

56. Android Project, Android Application Framework and .apk file definition [www] http://sites.google.com/site/io/inside-the-android-application-framework, Last access on 2011-02-24

57. Android Project, Android Developer Guide [www] http://developer.android.com/guide/index.html, Last access on 2011-02-24

58. Android Project, Android SDK Assets and Asset manager [www] http://code.google.com/android/reference/android/content/res/AssetManager.html, Last access on 2011-02-24

59. Android Project, Android SDK Resources definition [www] http://code.google.com/android/reference/android/content/res/Resources.html, Last access on 2011-02-24

60. Android Project, Android layouts [www] http://developer.android.com/guide/topics/resources/layout-resource.html, Last access on 2011-10-09

61. Android Project, Android SDK Views and XML [www] http://developer.android.com/reference/android/view/View.html, Last access on 2011-02-24

62. Android Project, Android Manifest file [www] http://developer.android.com/guide/topics/manifest/manifest-intro.html, Last access on 2011-02-24

63. M.Wahl, T. Howes, and S. Kille, Lightweight Directory Access Protocol, http://www.ietf.org/rfc/rfc2251.txt, Last access on 2011-10-09

64. Information Sciences Institute of University of Southern California, RFC 793 - TCP protocol [www] http://tools.ietf.org/html/rfc793, Last access on 2011-10-09

65. R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson, Stream Control Transmission Protocol, [www] http://tools.ietf.org/html/rfc2960, Last access on 2011-10-09

66. Android official homepage, DefaultHttpClient specification [www] http://developer.android.com/reference/org/apache/http/impl/client/DefaultHttpClient.html, Last access on 2011-10-09

67. The Webkit project, Webkit official WiKi [www] http://trac.webkit.org/wiki, Last access on 2011-02-24

68. Tom Tullis and Bill Albert, "Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics", Morgan-Kaufmann, 2008

69. Donald Norman, “The design of future things”, Basic Books, 6 Dec 2007

70. Android Project, GUI design guidelines. [www] http://developer.android.com/guide/practices/ui_guidelines/index.html, Last access on 2011-02-24

71. JQuery Booklet plugin, JQuery Booklet plugin [www] http://builtbywill.com/code/booklet/, Last access on 2011-10-09

72. JQuery official page, JQuery jFlip plugin [www] http://www.jquery.info/spip.php?article78, Last access on 2011-10-09

73. Adobe, Adobe’s Flash [www] http://www.adobe.com/software/flash/about/, Last access on 2011-11-01

74. R. Rivest, The MD5 Message-Digest Algorithm [www] http://tools.ietf.org/html/rfc1321, Last access on 2011-10-09

75. M. Carbonneaux, authMemCookie module [www] http://authmemcookie.sourceforge.net/, Last access on 2011-10-09

76. Memcached.org page, memCached definition [www] http://memcached.org/about, Last access on 2011-10-09

77. Android official homepage, Profiling Android applications with Traceview [www] http://developer.android.com/guide/developing/debugging/debugging-tracing.html, Last access on 2011-10-09

78. Bananity’s team, Bananity’s official homepage, http://www.bananity.com/, Last access on 2011-10-09

79. Koobits Inc., Official Koobits web page, [www] http://www.koobits.com, Last access on 2011-05-18

80. Slashdot.com, Google’s DART launch [www] http://tech.slashdot.org/story/11/09/09/1453224/google-to-introduce-new-programming-language-dart, Last access on 2011-10-09

81. Mozilla project, the Mozilla web API, http://hacks.mozilla.org/2011/08/introducing-webapi/, Last access on 2011-10-09

82. Adobe, Adobe’s Edge website, [www] http://labs.adobe.com/technologies/edge/, Last access on 2011-11-01

83. Sigil project, Sigil - a WYSIWYG ebook editor [www] http://code.google.com/p/sigil/, Last access on 2011-11-01

Other documents in the Leaf project

L1. D. Botero, The Leaf project: business plan, masters thesis project, November 2011

L2. G. Khalil, The Leaf project: commercial plan, masters thesis project, September 2011

L3. S. Galiàno, The Leaf project: infrastructure development, masters thesis project, August 2011

Appendixes

Appendix A - Uploading a book through the web interface

Inserting a new publication inside the Leaf platform library is a simple task that can be performed through the web interface. This possibility is available only to users who have already logged into the website.

The main structure of this operation is composed by two steps, represented by two separate files: *insertBook.html* and *insertedBook.php*.

The first of these file is a HTML document which presents a simple form, in which the user can define the publication's name and point to the real file in his device's storage. All the other information regarding the book, such as author, publisher, date and many more, will be automatically fetched by the server side utilities. Figure 13 shows a screenshot of this page.

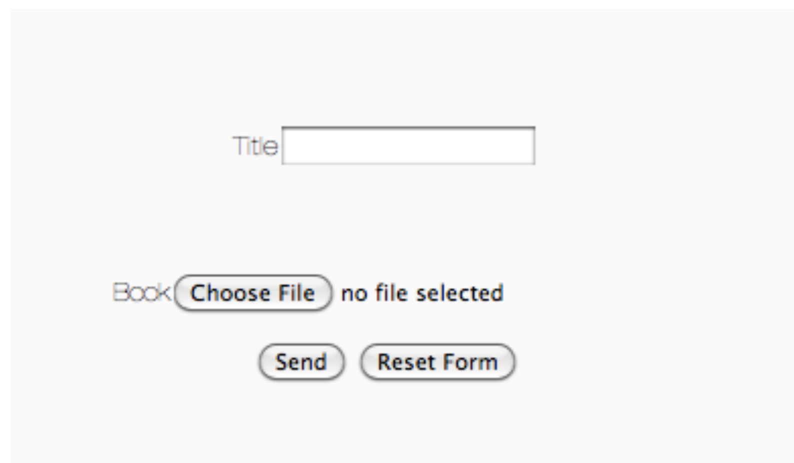

Figure 13 - Screenshot of the document inserting form

The implementation of this page is simple and clean. As observable from the code, the main element is a HTML form which contains one "*<input type='text'>*" tag for inserting the document's title and a "*<input type='file'>*" tag for choosing the file from the local storage. Then the form ends with the *reset* and the *submit* buttons. The interesting part is in the declaration of the form itself, when is crucial to specify the form method (POST) and the MIME-type (*multipart/form-data*), as explained in the second paragraph of section 5.1

The data sent by *insertBook.html* are then received by a PHP page called *insertedBook.php*. The main role of this page is to actually perform the storing of the documents in the server, retrieving the book file from the PHP array "*$_FILES[]*" and the book name from the "*$_POST[]*" array. The most important part in the PHP code is represented by the last line, in which the page calls the first server side utility, *summaryBuilder*. This Java class will then call the other utilities, described in section 5.2.

```
<div id="submain">

    <form action="insertedBook.php" method="post" enctype="multipart/form-data">

            <p style="margin-top:50px;">
            <span>Title</span><input type="text" name="title" id="title" />
             </p>

            <p style="margin-top:50px;">
            <span>Book</span><input type="file" name="book" id="book" />
             </p>


            <p style="margin:0 0 0 0; position:relative;"><span><input type="submit" value="Send" /></span>
              <input type="reset" class="bottoneForm" value="Reset Form" /></p>
        </form>


</div>
```

Figure 14 - Code for *insertingBook.html*

```
$book = $_FILES['book']['name'];  //The book filename, coming from the previous page

if(strpos($book, '.epub') !== false){          //Check if the book is in .epub format, if so it converts it into .zip
 $book = str_replace('.epub','.zip',$book);
 echo "Conversion Done, ".$book." has been uploaded in the server ";
}

if(!$book==''){
do {
 $msg = "";
 if (is_uploaded_file($_FILES['book']['tmp_name'])) {
  // Checking file size
  if ($_FILES['book']['size'] == 0) {
   $msg = "<p>Error</p>";
   break;
  }
  $type = $_FILES['book']['type'];
  if (($type!="application/epub+zip") && ($type!="application/xhtml+xml")&&($type!="application/epub+zip")) {
   $msg = "<p>Wrong Format</p>";
   break;
  }
  if (file_exists('test/'.$_FILES['book']['name'])) {        //Checking if there is already a file with the same name
   $msg = "<p>There is already a file with the same name, please rename the document!</p>";
   break;
  }
  // Moving the file
  if (!move_uploaded_file($_FILES['book']['tmp_name'], 'test/'.$book)) {
   $msg = "<p>Error</p>";
   break;
  }
 }
} while (false);
echo $msg;
} else $book = "";


$title = $_POST['title']; //Retrieving the title from the previous page

$path1 = "test/".$book."#"; //Real path of the file with the # to access the virtual folder

exec('java summaryBuilder ' . $path1 . ' ' . $title . '> /srv/www/htdocs/errors');
```

Figure 15 - Code for *insertedBook.php*

Appendix B - Sample “accessCookie”

The example below shows a sample “accessCookie”. Such a cookie is created by the *auth_memCookie* module running in the Apache web server when the user accesses the Leaf platform via an “in-app” login. This cookie is subsequently stored in the *memCached* memory. This cookie can be checked when the user wishes to access other parts of the server, such as the library or a specific book, to ensure that only authenticated users have access to the platform’s resources.

Cookie Name:	*accessCookie*
Accessible to script	*Yes*
Content:	*6873654d925a4411932581aa417096bd*
Path:	*/test*
Domain:	*“130.237.209.245”*
Created	*Saturday, October 8, 2011 3:55:00 PM*
Expiration	*Sunday, October 9, 2011 9:09:28 PM*

Figure 16 - Data present in the accessCookie

This cookie is directly connected to its instance in the *memCached*, in which there are additional information to keep track of the user’s session. Among these information, we can find: *username, group, remoteIp, email, name, surname*. It is important to note that these information can be automatically fetched from the LDAP[63] interrogation, which can obtain users details. Nonetheless, these information are not currently stored for the Leaf project platform.

Appendix C - Apache configuration for the *auth_memCookie*

The following portion of configuration is an extract of the configuration for the *auth_memCookie* module. After the loading of the module file, the tag *<Location />* contains all the rules to be applied for the main Apache directory. A series of options defines the behavior of the module, such as simulating the basic HTTP authentication, setting the HTTP headers in the HTTP packets, the cookie name, and the address for *memCached*.

In the bottom of the configuration is important to note the last portion, which defines the behavior for the "/test" folder, which in this case represents the location on the server where documents are stored.

```
LoadModule auth_memcookie_module  /usr/lib/apache2/mod_auth_memcookie.so
<IfModule mod_auth_memcookie.c>
<Location />
#     Options  +Indexes
#     Order allow,deny
#    Allow from all
#AllowOverride All
#AcceptPathInfo On
Auth_memCookie_CookieName accessCookie
Auth_memCookie_Memcached_AddrPort "localhost:11211"
Auth_memCookie_SetSessionHTTPHeader "on"
Auth_memCookie_SessionTableSize "40"
Auth_memCookie_Authoritative "on"
Auth_memCookie_SimulateAuthBasic on
Auth_memCookie_MatchIP_Mode "2"
Auth_memCookie_GroupAuthoritative "off"
# must be set without that the refuse authentification
AuthType Cookie

# must be set (apache mandatory) but not used by the module
AuthName "AuthLeafProject"

# to redirect unauthorized user to the login page
ErrorDocument 401 "/error.html"

#Require valid-user

</Location>
</IfModule>

<Location "/test">
    require valid-user
</Location>
```

Figure 17 - Extract of the server's configuration for the *auth_memcookie* module

TRITA-ICT-EX-2011:249

www.kth.se