# Kangoo API for Web-based Applications

Cristian Garcia

Master Project Report

Software Component Technology Group
Department of Computer Science
ETH Zurich

http://sct.inf.ethz.ch/

WS 2006/07

**Supervised by:**
Dominik Grolimund
Prof. Dr. Peter Müller

**Software Component Technology Group**
inf | Informatik
Computer Science

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Abstract

Kangoo is a global distributed system for the Internet. The goal of this master thesis project was to develop a Kangoo web API, which will allow third party developers to create web applications that interact with the Kangoo distributed storage system. The Kangoo web API interacts with Kangoo and returns metadata contained in the Kangoo metadata server. It also provides several search functionalities working with the Kangoo search server. A Java library specifically developed for Java client applications facilitate the development of such applications. A sample web application was created to demonstrate the various functionalities provided by the Kangoo web API. JML contracts were used inside the Kangoo web API to assert its correctness.

# Contents

# Chapter 1

# Introduction

The goal of this master thesis was to develop an API for web-based applications for the Kangoo distributed storage system. The idea behind the creation of such an API is to offer third party developers the opportunity to create web applications able to interact with Kangoo (see figure 1.2). Using this API third party developer will also be able to combine data contained in Kangoo with other data retrieved using other industry APIs including the Google Map API. Once this web application will be created, Kangoo users will be able to access their data from anywhere through the Internet using a simple browser. At the moment they can access data stored in Kangoo only using the Kangoo client application. This is a big restriction for the spreading of Kangoo, because Kangoo users must use their private workstation or install the client application to show to his friends the photos of their last holiday.

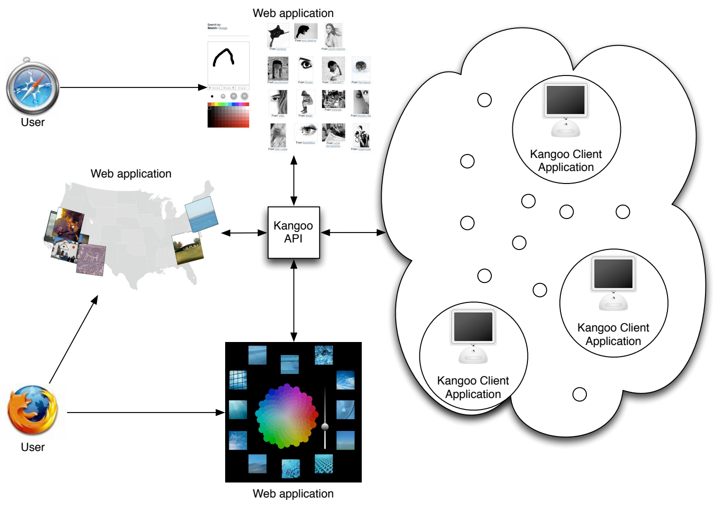

Web application
User
Web application
Kangoo API
Kangoo Client Application
Kangoo Client Application
Kangoo Client Application
User
Web application

Figure 1.1: *The Kangoo API provides access to data contained in Kangoo. Third party web applications can use it to interact with Kangoo.*

To reach our goal I developed the public API, which provides the ability to browse and search for files in the public Kangoo area. This API is able to interact with the centralized Kangoo search server, offering to the web application developer several types of search. The API is also able to interact with the centralized Kangoo metadata server returning details and information about the desired file. Using the public API, developers can implement really interesting application using the metadata returned.

In the future, the private API will be developed. This API will interact with the Kangoo grid storage system, the core of Kangoo, providing to the web applications the possibility of accessing the data contained in the Kangoo private area and downloading the content.

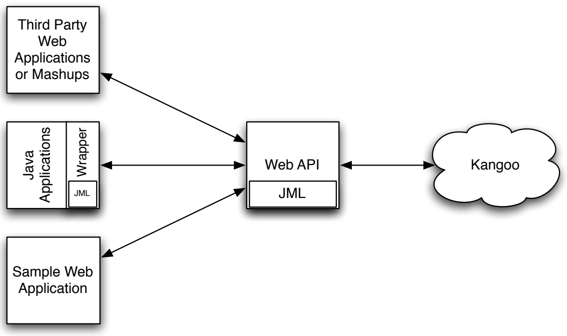


Figure 1.2: *Four components composing the master thesis core part: Web API, JML contracts, sample web application and Java wrapper.*

In this report I will describe the five core tasks composing the master thesis and discuss design and implementation used to complete them. The main task was the development of the Kangoo API. For the second task, I introduced JML contracts in the API to assert the implementation correctness. The third and respective fourth tasks were the development of a sample application, which demonstrates the API functionalities and the development of a Java wrapper, which is a Java library for normal Java applications. The last task composing the core parts of the master thesis was to write an API documentation to better support developers interested in using the Kangoo API.

# Chapter 2

# Kangoo

## 2.1 Introduction

Kangoo is a global, distributed storage system for the Internet [6]. It harnesses the idle disk space and bandwidth of participating computer to provide its users with a free, unlimited, personal online storage. Kangoo users can securely store private files online, share files with other friends, search and browse the Kangoo public area for files that other Kangoo users have published. Using Kangoo people could create groups composed by users sharing common interests.

## 2.2 Architecture

The main Kangoo components are:

- Grid storage system built on top of the user computers

- Search server

- Metadata server

The grid storage system is the main building block of Kangoo and is used to store files over the distributed storage system. Both metadata and search server are centralized servers, mainly for performances reasons, since it is important for a search server to return the desired information in the shortest time.

**The grid storage** is the base of Kangoo. It is used to store and download the files from the distributed storage system.

**The search server** contains index for public files published in Kangoo. It provides several search functionalities. Users can ask it for the most popular files (called top files), recently introduced files or for so-called pearls. Pearls are newly inserted files that gained quickly an important reputation. However, they are too young to be inserted in the top files list but probably with some time they will enter in this list. The search server also provides the ability to search for related tags and special search functions for Kangoo users and groups. Users can also ask for top archives to retrieve the top files for a specific time period.

**The metadata server** contains encrypted metadata associated to all files contained in Kangoo. Metadata are encrypted to protect them from illegal accesses. The metadata server contains classical information such file name, description, tags and date. It is also able to provide, for example, image file thumbnails and small image previews. For all file types it returns statistical information like the number of views for a specific file or the number of users that have marked a file as their favorite.

# Chapter 3

# Related Work

## 3.1 Industry Examples

Already several other online storage services provide an API for the development of web applications. Probably the most famous is Flickr. Flickr, a service provided by Yahoo!, is a web application for the purpose of storing and sharing photos online. It provides the ability to create online communities and to share photos with their members. Flickr provides both a public API as well as a private API [3]. With these APIs it offers all the core functionalities required to browse, search, upload and download images, as well as the opportunity to retrieve details and information about a photo including associated tags and comments. The Flickr API has been developed using several types of communication protocols; in fact it offers an XML-RPC, SOAP and REST API interface. The responses are all in XML format.

Yahoo! also offers for all its other services APIs for web-based applications [2]. For example both Web Search Services and Maps provide APIs based on REST communication protocol. But Yahoo! is not the only online service provider that offers API for web application. eBay provides an API based on SOAP and REST communication protocol. Google provides APIs for its online services [1]. Two of them are Google Calendar, which has an API based on REST, and the Search Service which is now based on AJAX to replace their previous SOAP API.

## 3.2 Communication Protocols

As already seen in section 3.1 there are three major communication protocols used by APIs designed for web-based applications: XML-RPC, SOAP and REST. In the following sections I will introduce these three protocols providing a short description and their main advantages and disadvantages.

#### XML-RPC

XML-RPC is a remote procedure call based on XML that uses HTTP as transport [8]. The main advantage of XML is its human readability but this is at the expense of a lengthy syntax. Using HTTP it works fine with the firewalls.

#### SOAP

Simple Object Access Protocol (SOAP) is a protocol for exchanging XML messages normally based on HTTP and is the successor of XML-RPC [12]. It provides also a Web Services Description Language (WSDL) as well the Universal Description, Discovery, and Integration (UDDI). As for XML-RPC it provides lengthy syntax and it exchanges human readable messages. Advantage of SOAP as for XML-PRC is the good work with firewalls since it is based on HTTP.

### REST

Roy Fielding introduced the concept of Representational State Transfer (REST) in his dissertation [5] on 2000. In reality REST is not a communication protocol but an architecture style for networked systems. Basic principles of REST include: subdivions of the functionalities in resources that are uniquely addressable and the use of a communication protocol that must be based on client/server paradigm, stateless, cacheable and layered. HTTP is mainly used as the communication protocol. The main advantages of REST are a small response time and server loading using caching system. Another advantage is better server scalability since it is stateless. Main disadvantages are the lack of REST library and clear examples of REST usage.

Chapter 4

# API Architecture

## 4.1 Introduction

There are three main challenges for the design of the API. The first challenge is the communication with third party web applications: choose the communication protocol and implement the API according to it. Second challenge is the interaction with Kangoo components: how to access the data contained in Kangoo and how to work with the Kangoo servers. The third and last challenge comes from our desire to have complete control over the API usage. We want to be able to control who is using the API and how he uses it. The design used to solve this last challenge will be described in section 5.

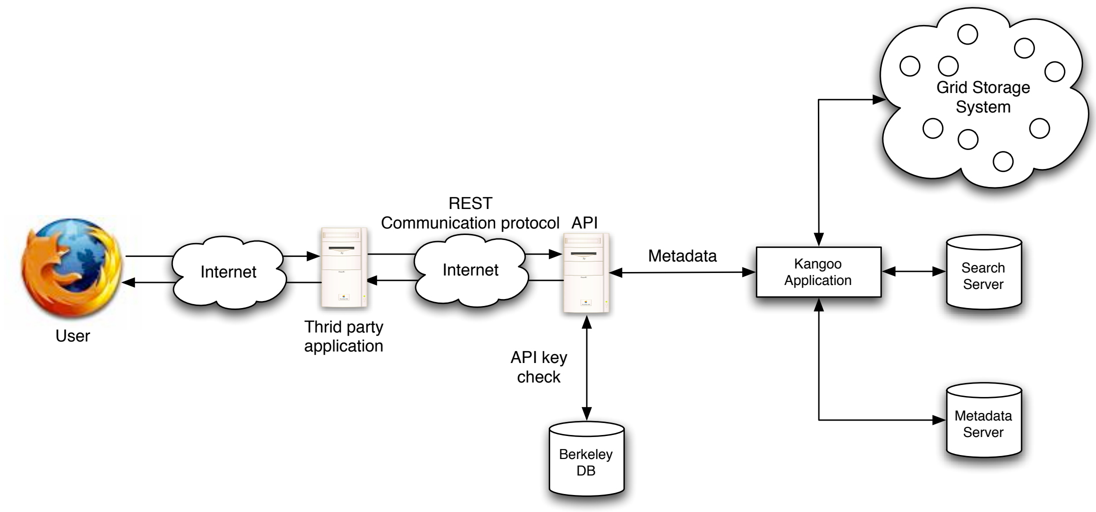


Figure 4.1: *The API architecture. The API is based on REST using HTTP as communication protocol. Interaction with the Kangoo components goes through the Kangoo application instance. Berkeley database is used to store API keys and allows us to control the API usage.*

## 4.2 Communication Protocol

### 4.2.1 REST

The Kangoo API is based on the Representational State Transfer (REST) architecture. REST is not a communication protocol but an architectural style of networked systems. The idea behind REST is that each online service is based on resources and each resource is mapped to a specific URL. This architecture is already used in several APIs as described in chapter 3.

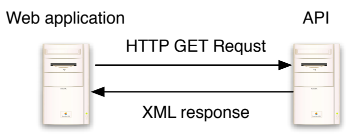


Figure 4.2: *API communication based on REST using HTTP. Method invocation submitted through an HTTP GET request. Response based on XML data.*

REST uses HTTP as the communication protocol. API invocations are based on simple HTTP GET requests to the desired API method URL (see figure 4.2). Parameters for the method invocation are passed either as URL parameters (for example `http://www.kangoo.com/API/items/getDetails?elementKey=2` to request more details on a file) or directly in a URL path (for example `http://www.kangoo.com/API/thumbnail/kangoo/cristian/favorites/kangoo.jpg` to request the thumbnail of an image file). Since parameters are passed in the URL, all of them are strings and they must be parsed by the API method to the correct type format. The main disadvantage of using REST is the care that the developer should take to correctly encode the string parameters in UTF-8 format.

API method responses are based on XML data format (see figure 4.2). I chose XML since web applications are the target applications for the API, and XML can be easily transformed into HMTL using the Extensible Stylesheet Language Transformation (XSLT). Moreover XML is platform independent and human readable. These advantages coupled with the REST architecture, allow a developer to test API method invocation using a simple browser. A developer can see the XML response directly in the browser by using the method URL with the desired parameter as Internet address.

The REST architecture based on HTTP requests could also take advantage of the application server caching system. Methods that always return the same response for a specific request could use the cache system, reducing the response time and interaction with the Kangoo system. An example of such a method is the one responsible for providing the thumbnail and the preview for the image files. Since the thumbnail, as well as the preview of an image never changes, the first time a request for an image is submitted to the API, the application server can cache the API method answer. Cached data can be used for successive method calls for the same image. This can be interesting for example for top files which are requested frequently by different users.

REST will also be useful for the private API. The file browsing of the private data will look clearer and simpler. Since the browsing through user's folders will result in API requests based on the Kangoo folder paths, the path could be used and passed as parameter directly in the URL as described above.

XML-RPC and SOAP have also been considered as possible communication protocols for the API. They both have a good documentation, and several libraries are provided by Apache, for example. But in the case of SOAP, it results in a complexity overhead. Some main SOAP features like the Web Services Description Language (WSDL) and the Universal Description, Discovery, and Integration (UDDI) are not useful and would not be integrated in the API. It also introduces an overhead complexity in the response format. The XML response would be longer and we will not take advantage of the information contained in the extra data introduced by SOAP.

XML-RPC would be a valuable alternative to REST. Its biggest problem, as for SOAP, is the length and complexity of the XML response. Here follow two examples of XML-RPC and REST responses.

```
<?xml version="1.0"?>
<methodResponse>
  <fault>
    <value>
      <struct>
        <member>
          <name>faultCode</name>
          <value><int>4</int></value>
        </member>
        <member>
          <name>faultString</name>
          <value><string>Too many parameters.</string></value>
        </member>
      </struct>
    </value>
  </fault>
</methodResponse>

<?xml version="1.0"?>
<response>
  <fault>
    <faultCode>4</faultCode>
    <faultString>Too many parameters.</faultString>
  </fault>
</response>
```

As we can see the first response returned by XML-RPC is longer and more complex to be transformed using XSLT. Since the variable names could not be accessed directly. The second response returned by REST is shorter and the variable can be easily accessed since they are represented by elements with the variable name. In addition to the simpler XML format, we opt for REST looking also at the development of the private API and seeing in it a more clear way to browse private user files.

### 4.2.2 Application Server

We decide to implement the API methods using the javax.servlet.http.HTTPServlet class [15]. Each method has been implemented in a separate servlet. This design gives us a better control over the API and allows us to disable an API method by simply modifying the configuration file of the application servlet. Servlets run on the Apache Tomcat 5.5 application server [14].

## 4.3 The API and Kangoo

At the moment the API is able to interact with metadata and search servers. No file is downloaded from Kangoo distributed storage, the API returns only metadata contained in the Kangoo

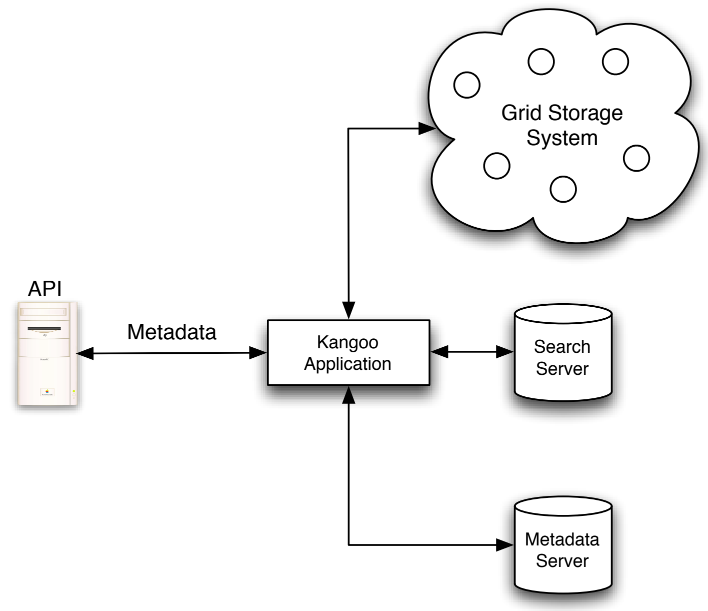


Figure 4.3: *API schema. Interaction with the Kangoo components goes through the Kangoo application. The API returns only metadata contained in Kangoo.*

metadata server.

The API interacts with the Kangoo components using the Kangoo application created during the starting phase of Tomcat (see figure 4.3). This decision has been taken to be able to decrypt the metadata, but also looking at the future extension of the API into the private area of Kangoo. In fact data contained in the private area is encrypted and fragmented before being saved in the distributed storage. So to download a file from the distributed storage we first have to retrieve all the pieces necessary to recreate the file. Once the file is composed, we need to decrypt it. These two processes are already handled by the Kangoo application. Moreover there are also restrictions based on the access rights that the Kangoo application already automatically handles.

# Chapter 5

# Developer Authentication

## 5.1 Introduction

To better control the API usage we introduced API keys. API keys are alphanumeric codes that each method invocation must provide as parameter, otherwise the request will not be handled by the API. The web application developer must request API keys providing personal information like name and email address. He also must provide a motivation with a description of the intended API usage. The Kangoo team must agree to the API key request to allow the developer to use the web API. Using the API key passed at each API method invocation we could monitor the API usage: if a mishandling usage is noticed, for example an excessive number of API invocations in a restricted time period, the API key could temporarily be disabled.

## 5.2 API Key Database

To store the API keys a database has been implemented using the Berkeley Database (see figure 4.3) that stores key/data pairs as byte array. Berkeley DB [4] is a simple and fast database system that works directly with Java objects, and it matches perfectly our requirements to manage the API key database. It is also used by Google for similar purposes.

A database entry could be either active or inactive. This design gives us the possibility to deactivate an entry without deleting it, keeping in the database all the information about the developer that requested it. For example, if we notice the mishandling usage of the API from a developer, we have the opportunity deactivate it and, if it is the case, to reactivate it later once the situation has been clarified.

The API key database provides front- and backend services (see figure 5.1). The frontend service provides a read-only access to the database entries and is used by the API servlets to check the validity of the API key passed to the method invocation. It implements methods to check API key validity, to get data for a specific API key and to get the entry set of the data contained in the database. The backend database service provides read/write access to the database. This second service is used by the API key applications (described in section 5.3) and provides methods to check if a specific API key is contained in the database and to automatically create a unique API key entry returning the string representing the key value. It also provides methods to update data entries and to delete, activate and deactivate a key from the database.

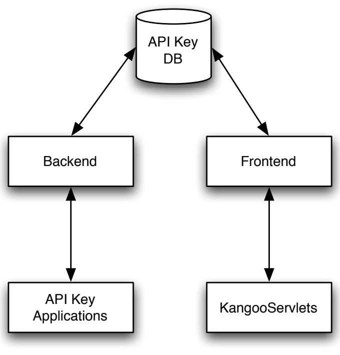


Figure 5.1: *Database structure. Front- and backend services.*

## 5.3 API Key Applications

API key applications are divided into two families. The first family contains only a single application providing a web page that is used by the web application developer to request the API key. The second family is composed by the API key manager applications. Two applications are contained in this second family: the first one is a web service that lists the API key entries contained in the database, and the second one is the updater web application that could be used by the Kangoo team to modify the data contained by a database entry.

The application of the first family composed by an online form is public and accessible from the Internet. Developers can use it to request their key. Once the request is submitted, providing all the mandatory information, the application introduces an inactive entry in the database and sends a notification through email to the Kangoo team. The email contains a link to activate or delete the entry from the database. After having evaluated the request, the Kangoo team could use the link to proceed according to his decision.

The second family of API key applications is protected through the Tomcat authentication system by restricting the access only to users inside the Kangoo team. To access these two pages, composing the second group of applications, a valid username and password must be provided. Through this protection system the pages can be published in the Internet and accessed by the Kangoo team from any location. The manager application returns a table containing all the entries of the databases providing all the information associated with the key. The Updater application outputs a form that allows Kangoo team to update the entry information associated with a key and activate/deactivate it.

# Chapter 6

# API

## 6.1 Introduction

As already introduced in section 4.2, each API method has been implemented in separate servlet based on the javax.servlet.http.HTTPServlet class. Several methods have been developed [7] to provide the best search functionalities and returning all the possible information contained in the metadata server.

Through the API it is possible to get the list of top score, pearl and recent files. API provides also several functionalities for tags associated to items contained in Kangoo. We can get the list of most popular tags or ask, providing a tag text, for related tags. We can also request the list of tags and comments associated to a specific item. For each item it is also possible get all details, like file name or owner username, and statistics, like number of views or number of Kangoo users that had marked it as one of their favorite items.

In a first phase we started developing in the API the methods provided by the first version of the search server. During this phase we implemented the standard search, top, pearls and recent methods. On a second phase we introduced new functionalities looking at other industry APIs evaluated during a preliminary project phase. This phase gave us the most ideas for the API methods; we introduced for example the related tags and top score archives functionalities. On a third phase, started during the development of the first version of the sample application, we introduced new API methods to improve the demo application functionalities. We implemented for example the getDetails method to be able to more quickly request statistics information and details about a specific item.

## 6.2 API Schema

The servlets composing the Kangoo API are divided in two groups (see figure 6.1). The first one, called APIServlet, is composed of servlets that interact only with the metadata server. This first type of servlets extends only the abstract KangooServlet class. This abstract class implements all common methods used by the API servlets. KangooServlet uses the DOM document to produce the XML response. The second group, called SearchServlet, is composed of servlets that interact both with the search and, as well as in the previous case, with the metadata server. Servlets interacting with the search server must implement also the IFetcherListener interface to handle the asynchronous response returned by the search server. Since the API could not provide an asynchronous response, the SearchServlet waits on a Java object until the response from the search server is returned and the results are fetched.

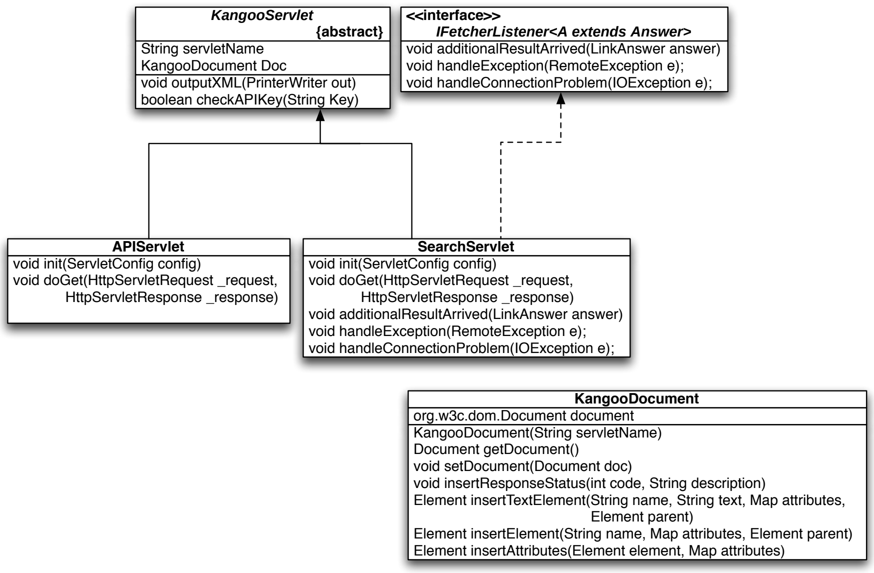

KangooServlet
{abstract}
String servletName
KangooDocument Doc
void outputXML(PrinterWriter out)
boolean checkAPIKey(String Key)
<<interface>>
IFetcherListener<A extends Answer>
void additionalResultArrived(LinkAnswer answer)
void handleException(RemoteException e);
void handleConnectionProblem(IOException e);
APIServlet
void init(ServletConfig config)
void doGet(HttpServletRequest _request, HttpServletResponse _response)
SearchServlet
void init(ServletConfig config)
void doGet(HttpServletRequest _request, HttpServletResponse _response)
void additionalResultArrived(LinkAnswer answer)
void handleException(RemoteException e);
void handleConnectionProblem(IOException e);
KangooDocument
org.w3c.dom.Document document
KangooDocument(String servletName)
Document getDocument()
void setDocument(Document doc)
void insertResponseStatus(int code, String description)
Element insertTextElement(String name, String text, Map attributes, Element parent)
Element insertElement(String name, Map attributes, Element parent)
Element insertAttributes(Element element, Map attributes)

Figure 6.1: *Servlet UML schema.*

## 6.3 Request Handling

The API request handling could be divided in four different phases. In the first one the API key provided as parameter to the method invocation is checked and in case it is not correct a standard error code is returned. Once a request passes the requirements of the first phase, the parameter of the method invocation passed in the URL are parsed and transformed into the desired variable type. If some mandatory parameter is failing or if it could not be correctly parsed, an error code is returned. In case of absence or error of optional parameters, the default values, specified in the API documentation, are used.

At this point we enter in the third phase. During this phase the interactions with the Kangoo components are performed. If the method invoked must interact with the search server, we submit the request to retrieve the item ids matching the search request and use them to get the desired information from the metadata server. Otherwise we use the item id passed as parameter to do it. All methods invoked during this phase have contracts specified using JML. On the fourth phase the response will be processed and created.

## 6.4 Response Creation

To produce the XML response we use the Document Object Model (DOM) [11]. Using DOM we have the opportunity to represent the response data as a tree composed by objects. To simplify the document construction a KangooDocument class has been created (see figure 6.1). This class implements useful methods to insert new XML tags and tag attributes inside the document hiding the DOM complexity. It implements also other methods to create and insert the complete status response information in the document response. It also manages the DOM document creation and initialization.

To better support the web application developer who wants to use the Kangoo API, the API documentation includes the response structure for each method [7]. Using this structure the developer can parse and retrieve the desired information. Inside the documentation, there is also an example showing a demo interaction with the API: how to submit an API invocation and how to retrieve the response data.

## 6.5 Error Handling

All API errors are reported to the caller using the status element tag contained in all method responses. The status response is composed by an integer error code and an error description. The API documentation lists all possible errors returned by each method invocation. Through this way we also notify the user in case of invalid API key, missing or invalid mandatory parameters. The 0 error code is returned in case of a correct method.

In addition, an email notification system has been developed to signal to the Kangoo team severe failures of the API. Possible causes of such a failure are: the crash of the search server resulting in no answer to the API requests, or the forced logout of the Kangoo application instance. In both cases the failure will be notified per email to the Kangoo team that could investigate the causes, solve the problem and restart the search server or the Tomcat application server to force the API login. The API is also able to notice unexpected search server exceptions. Also in this situation the Kangoo team is notified per email.

# Chapter 7

# JML

We introduced JML [9] contracts in all the components related to the API. At the moment JML contracts are inserted in the code only as comments since the standard Java generic classes (as for example Map and Set) are not supported by the current JML specification, and we are therefore unable to compile the API code with JMLC. Generic classes are widely used in the search server as well as in the API and in its related projects.

To solve this problem we considered two possible solutions. In the first one we could remove the generics from the projects. The main advantage is the immediate support of JML in the API, but losing all the advantages offered by the usage of generic classes. This first solution would have also required a lot of changes not only in the web API project, but also in Kangoo projects used by the API.

A second possible solution is to leave the JML contracts in the code as simple comments without compiling the code with JMLC. The contracts will then be compiled once the JML specification will be updated to support the standard generic classes. This update will probably be available in the next months. The main advantage of this solution is the possibility to continue using the generics inside the API, taking advantage of their functionalities and avoid the requirement to introduce important changes in both Kangoo API project as well in some other Kangoo projects used by the API. But until the JML specifications update the API will not be able to use JML checks. We preferred the second solution, since the update will probably be available soon.

As already introduced in section 6.3, JML contracts have been implemented for all the internal API methods that are used after the initial parameter parsing. The current design has been adopted to better support the API user, returning all the required information in case of wrong parameters. The parameters passed in the URL are checked and parsed by a special class that is responsible to report all severe errors. Postconditions are also stated for the methods responsible to parse the parameters.

JML contracts have also been introduced in two other projects related to the API. The first one is the project implementing the API key database, and the second one is the API's Java client wrapper project. In both projects contracts are specified in all classes and methods. Since they are two standalone projects, we were able to better take advantage of JML usage.

Due to this problem we were not able to take advantage of the JML checks. They would be really useful to solve dummy bugs that required some time to be debugged and detected. But they have been already useful to think at better checks to introduce during the parameter parsing phase.

# Chapter 8

# API Sample Application

## 8.1 Introduction

To better demonstrate all the functionalities of the API a sample application has been developed. As already done for the API methods, the sample application uses the javax.servlet.http.HTTPServlet and runs on the Apache Tomcat application server.

Two versions of the sample application have been developed. The goal of the first sample application was to provide a preliminary demonstration of the API usage. This first sample application provides a simple graphic interface and allows specifying all the desired parameters for API invocation through an initial page containing an HTML form. It allows us to improve the search server functionalities. In fact it provides us some new ideas to improve the amount of information retrievable from the API. For example, it suggest to us the implementation of the getRelatedTags method to provide extra details about a string search. To improve the sample application functionalities, we introduce an EOF flag in search responses to use in the next/previous page navigation.

During the last month of the master thesis, a second sample application has been developed. This application has more elaborate graphics and could be used as the main Kangoo web application. Four different web applications, each one developed in a separate servlet class, compose the second version of the sample application: initial, archive, popular tags and preview page. The first page is responsible to interact with the main functionalities of the search server. It is the main and initial page of the sample application and displays the top, pearl and recent files, as well as giving the possibility to search for files (see figure 8.2). As initial page it contains links to the other components.

The archive application is responsible for the interaction with the archives contained in the search server. It provides the lists of all archives according to the time scale selected by the user using a calendar layout (see figure 8.4). The third application composing the demo provides the list of all tags contained in Kangoo search server. The tags are shown with different character sizes according to their score (see figure 8.5). The last component is responsible to output a preview page for the selected file containing details like the file owner's name, file path, views counter and other statistical information (see figure 8.7). In case the file requested is an image, the web application provides a small preview. Otherwise, only a standard Kangoo image is provided. In the next sections I will discuss the layout and request handling of this sample application second version.

## 8.2 Layout and User Interaction

After the development of a first sample application we decided to create a new version with a completely new layout that allows a better and more intuitive user interaction. In the first version the user had to specify the application parameters mainly using the check boxes and pulldown menu (see figure 8.1). The interaction with this configuration did not seem to be intuitive enough. Moreover all requests to the sample application must pass through an initial page where an input form is displayed and where the user has the possibility to specify his request.

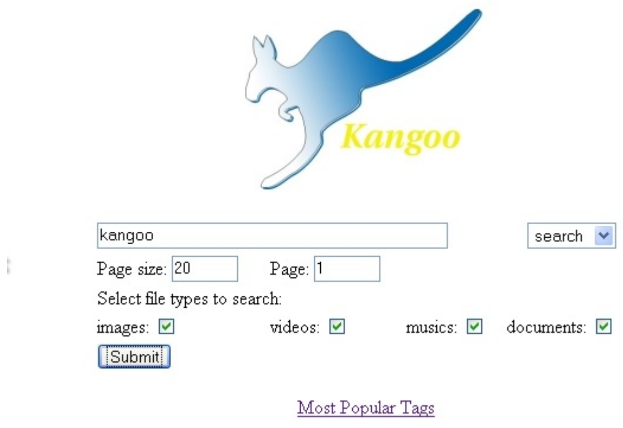


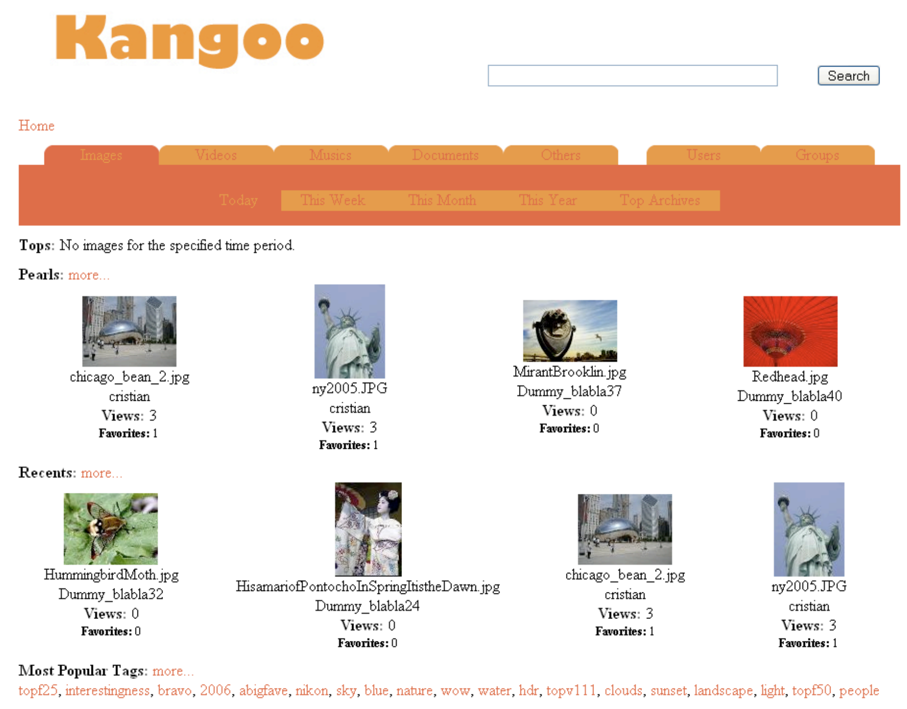
Figure 8.1: *The initial page of the first sample application. The user decides what to display using check boxes and pull down menus.*

The new application is based on a tabbed layout that uses HTML links to interact with the user (see figure 8.2). With this new layout, user can modify his request directly from the response page without going through the initial page. The layout provides two navigation menus that are extended to three for the archive web application. The selected tab is displayed using different colors to show the user his previous decisions. The upper navigation menu allows the user to specify the file type desired for the current request while the lower menu allows the user to specify the time period.

All the parameters needed by the application to execute the user's request are passed directly in the URL and are easily inserted in the HTML links contained in the pages. From any position the user has the possibility to submit a search request using the search form displayed in the top part of the page. Also the search request depends on the file type and time period selected using the tabs.

The tabbed layout is used for all four web applications. In the initial page application the user has the possibility to see a small preview with the first four items for each category: top, pearls, recent items. He can get more items for the desired category by simply clicking on the corresponding link (see figure 8.3). In the initial page, once it shows the list of items for the selected category, the user can navigate and request the previous and next item page.

To ask for the top score archive application the user must click on the Top Archive link contained in the main time period navigation menu. As said before, the archive application provides a third navigation menu that allows the user to specify the top score archive time period. The time period specified determines the calendar layout (see figure 8.4). Possible calendar styles are: day of the month style, where the user can navigate using links through year and month and for each year/month pair the corresponding days are shown; week of the year style, where user



**Tops**: No images for the specified time period.

**Pearls**: more...

chicago_bean_2.jpg
cristian
Views: 3
**Favorites:** 1

ny2005.JPG
cristian
Views: 3
**Favorites:** 1

MirantBrooklin.jpg
Dummy_blabla37
Views: 0
**Favorites:** 0

Redhead.jpg
Dummy_blabla40
Views: 0
**Favorites:** 0

**Recents**: more...

HummingbirdMoth.jpg
Dummy_blabla32
Views: 0
**Favorites:** 0

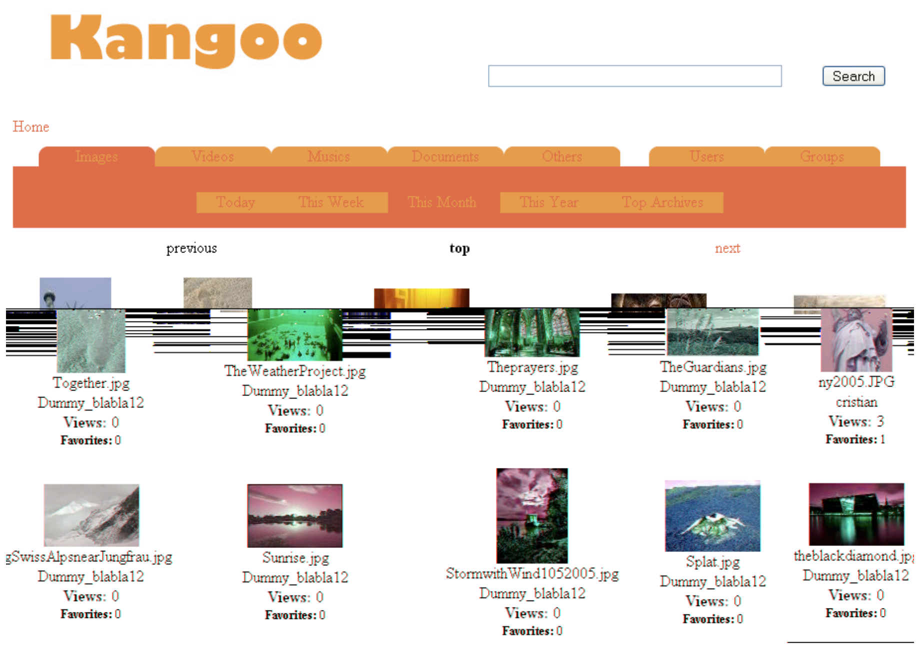
HisamariofPontochoInSpringItistheDawn.jpg
Dummy_blabla24
Views: 0
**Favorites:** 0

chicago_bean_2.jpg
cristian
Views: 3
**Favorites:** 1

ny2005.JPG
cristian
Views: 3
**Favorites:** 1

**Most Popular Tags**: more...
topf25, interestingness, bravo, 2006, abigfave, nikon, sky, blue, nature, wow, water, hdr, topv111, clouds, sunset, landscape, light, topf50, people

Figure 8.2: *The initial page of the sample application. The user has the possibility to specify his decision by clicking on the navigation menus.*



previous **top** next

Together.jpg
Dummy_blabla12
Views: 0
**Favorites:** 0

TheWeatherProject.jpg
Dummy_blabla12
Views: 0
**Favorites:** 0

Theprayers.jpg
Dummy_blabla12
Views: 0
**Favorites:** 0

TheGuardians.jpg
Dummy_blabla12
Views: 0
**Favorites:** 0

ny2005.JPG
cristian
Views: 3
**Favorites:** 1

gSwissAlpsnearJungfrau.jpg
Dummy_blabla12
Views: 0
**Favorites:** 0

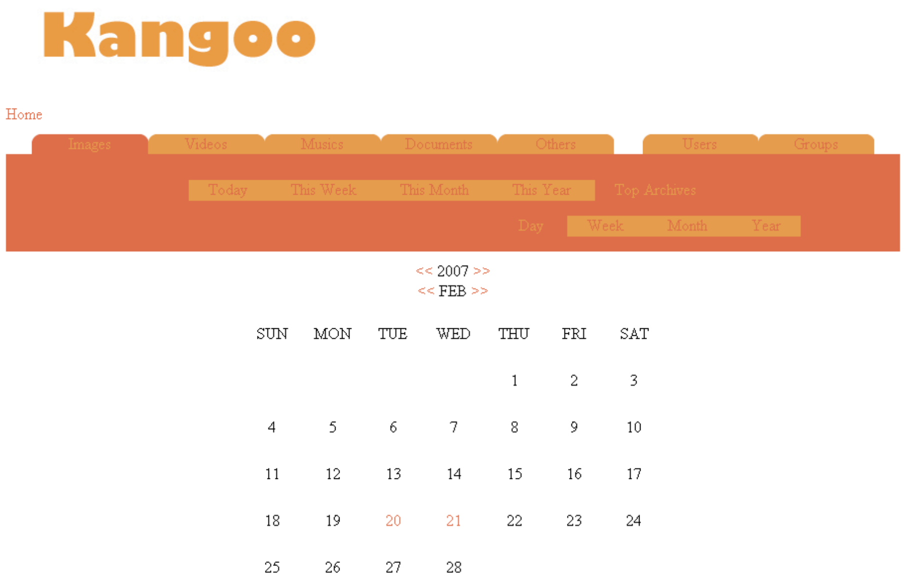
Sunrise.jpg
Dummy_blabla12
Views: 0
**Favorites:** 0

StormwithWind1052005.jpg
Dummy_blabla12
Views: 0
**Favorites:** 0

Splat.jpg
Dummy_blabla12
Views: 0
**Favorites:** 0

theblackdiamond.jp
Dummy_blabla12
Views: 0
**Favorites:** 0

Figure 8.3: *Images contained in the top list of the current month.*



| SUN | MON | TUE | WED | THU | FRI | SAT |
|---|---|---|---|---|---|---|
| | | | | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | | | |

Figure 8.4: *The top score archive application with day of month application layout.*

can navigate through the years selecting the link corresponding to the desired week; month of the year style, same layout as for the week of the year style, where user can navigate through the years, but in this case the user can select the desired month clicking on the corresponding link; and the final year list style, where all the year archives contained in the search server database are listed.

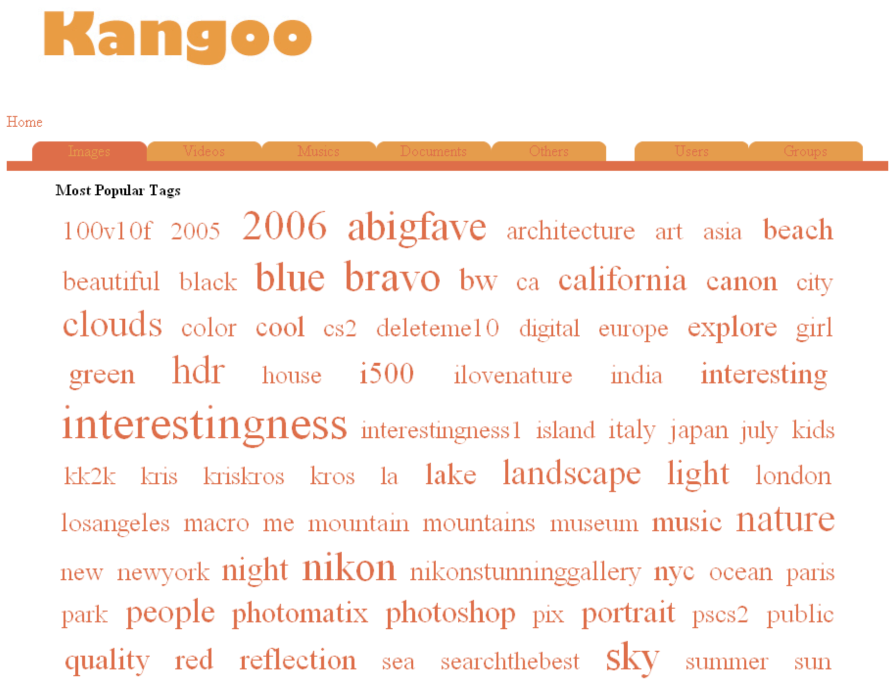
On the most popular tag web application, the time period navigation menu is hidden since the API method to retrieve the most popular tag list doesn't support it. So on this page the user can only select the desired file type (see figure 8.5). Also on the preview web application the time period navigation menu is not displayed. The file type navigation menu is present but it contains no links, it shows only the current file type. This layout is presented because the page is used only to display details of the selected file. It displays statistical information, file details (like file name and owner name) and the list of tags and comments associated with the file (see figure 8.7).

The layout of the sample application is created using XSLT. XSLT is also used to insert the data contained in the XML responses retrieved from the search and metadata server. HTML code uses also CSS. In CSS we specified the colors of the layout for the dynamically selected tabs. The layout could be displayed in three different colors.

## 8.3 Request Handling Process

The request process can be divided in three phases (see figure 8.6): parameter collection and Kangoo API methods invocation; API responses parsing, error checking and responses collection in single DOM document; and XSLT transformation of the created DOM document.

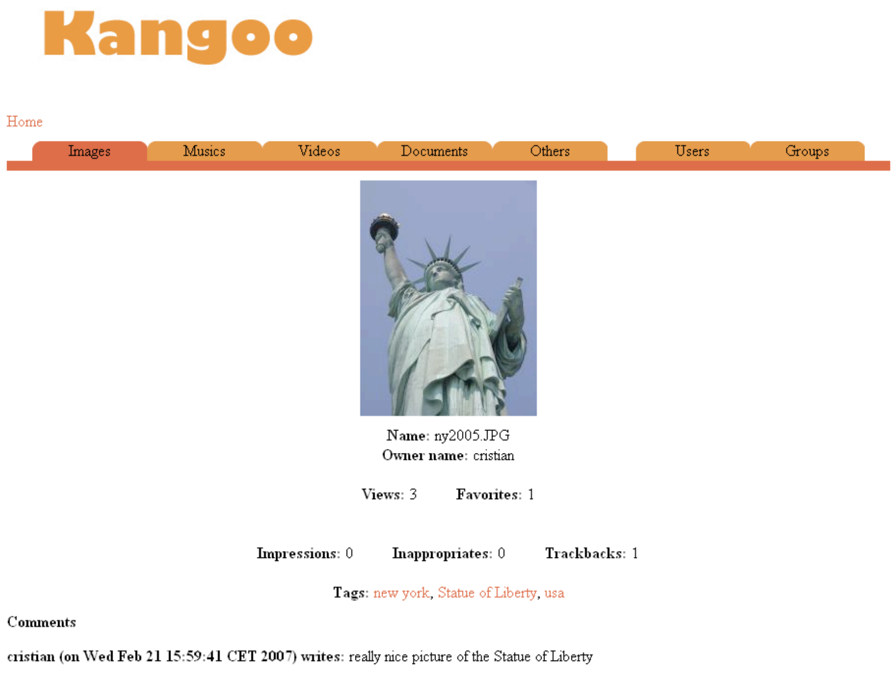
API method invocations are based on the parameters passed in the web application URL. They are set dynamically in the HTML links contained in the page returned by the web application using the XSLT. The links are created using the user's previous choices. Using the parameters passed in the request the corresponding API methods will be invoked. No parameters are mandatory; in case of absence of a parameter the default value is used. Eventual IOExceptions thrown by the connection with the Kangoo API will result in an error message to the user that is displayed using a HTML error page.



Most Popular Tags

100v10f 2005 2006 abigfave architecture art asia beach beautiful black blue bravo bw ca california canon city clouds color cool cs2 deleteme10 digital europe explore girl green hdr house i500 ilovenature india interesting interestingness interestingness1 island italy japan july kids kk2k kris kriskros kros la lake landscape light london losangeles macro me mountain mountains museum music nature new newyork night nikon nikonstunninggallery nyc ocean paris park people photomatix photoshop pix portrait pscs2 public quality red reflection sea searchthebest sky summer sun

Figure 8.5: *The list of the most popular tags for the other document file types.*



**Name:** ny2005.JPG
**Owner name:** cristian

**Views:** 3 **Favorites:** 1

**Impressions:** 0 **Inappropriates:** 0 **Trackbacks:** 1

**Tags:** new york, Statue of Liberty, usa

**Comments**

**cristian (on Wed Feb 21 15:59:41 CET 2007) writes:** really nice picture of the Statue of Liberty

Figure 8.6: *Preview page provides details about the file, statistical information and list all the tags and comments associated with the file.*

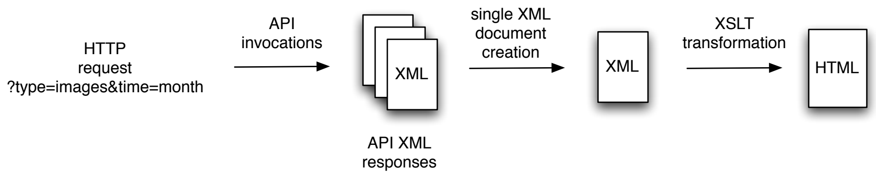


Figure 8.7: *Sample application request handling.*

The goal of the second phase is the creation of a single DOM document that will be transformed in the third phase to produce the final HTML page. To produce it we first parse all the API responses into separate DOM documents. We check the responses to verify possible API error codes that will result, if they are considered severe failure (for example search server unavailability), in an HTML error page. During these checks, no notification is returned to the Kangoo team since for such severe failures already the API will send an email to the team. After the error checks the DOM documents are inserted in a single one that is returned to be transformed.

During the third phase the API response information is displayed through an XSLT transformation. Using XSLT we can easily transform XML data into the HTML format. And since XSLT has a good efficiency we do not remove any unused data returned by the API methods DOM response, but simply leave it in the DOM document and filtered using XSLT. The pages are outputted according to the layout described in section 8.2.

# Chapter 9

# API Java Wrapper

## 9.1 Introduction

The Kangoo API has been developed for web applications and works using XML based communication protocol. XML data is not so comfortable to use in normal Java applications. To facilitate the implementation of Java client applications that interact with the public Kangoo API we developed a Java wrapper. The Java wrapper is a Java library that hides the API communication and the API response parsing, and it allows interacting with the API using Java objects. API method invocation is hidden behind a simple local method invocation and the API responses are contained in Java objects.

## 9.2 Structure

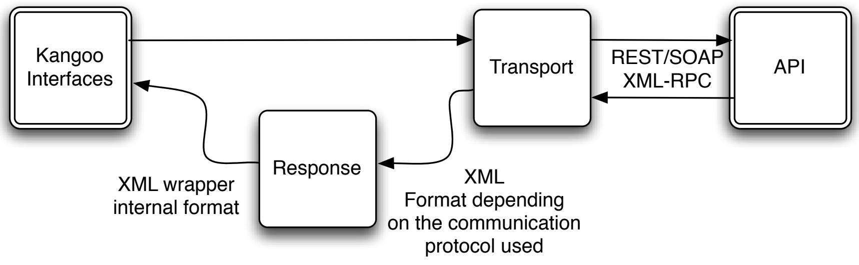

Kangoo Interfaces
Transport
REST/SOAP XML-RPC
API
Response
XML wrapper internal format
XML Format depending on the communication protocol used

Figure 9.1: *Wrapper/API communication.*

The Java wrapper has been designed to handle future extension in the Kangoo API's communication protocols. New communication protocols can be added by simply implementing a pair of Transport/Response classes. Transport class is used by the Kangoo wrapper interfaces to communicate with the API (see figure 9.1). Each API interaction goes through the Transport *get* method that returns an API response contained in a class that extends the abstract Response class. *get* method takes as parameter the URL path to invoke the desired API method and a map containing the API method's parameter pairs (name/value). Using this information, it submits the request to the API according to the communication protocol it implements. In the REST case it encodes the parameters according to the UTF-8 format directly in the URL path and submits the HTTP GET request.

The goal of the communication protocol response class is to extract from the communication protocol dependent XML API response the data and transform it in the wrapper internal XML format. The Kangoo interfaces can request the response in the wrapper internal XML format invoking the *getPayload* method. Using this format they are able to parse the returned data and use it to create the Java objects returned as response.

API methods are implemented in several Kangoo interfaces, one for each package contained in the API: archives, comments, favorites, groups, items, tags, tests, and users. Each interface provides all methods implemented in the corresponding API package. Interfaces invoke API methods using a transport object and could be requested using the central Kangoo class. The Kangoo class is used to create the interfaces with the desired transport protocol.

## 9.3 Error Handling

The Java wrapper error handling is based on exceptions. The wrapper could throw two types of exceptions. The first possible exception, IOException, is used to notify the caller in case of communication problems with the API. In such a case, the IOException thrown by the HTTP connection is forwarded to the wrapper caller.

A second type of exception thrown by methods contained in the Kangoo interfaces is of type KangooException. They are thrown automatically each time an API response contains an error code with value different from 0. The KangooException is composed by the error code and its description.

# Chapter 10

# Conclusions

## 10.1 Achievements

In this report I have shown the approaches taken to develop the public Kangoo API and the other two related projects. During the master thesis project, I completely achieved four of the five core parts defined in the master thesis description. The only incomplete part is the specification of contracts using JML, since I was not able to compile the projects due to the problem encountered using the Java standard generic classes.

I implemented a well designed API that can be extended in the future to also access data contained in the Kangoo private area. To demonstrate the API functionalities I implemented a sample application that could be used as the official Kangoo web application. It provides all the Kangoo search server functionalities and access to the details about files contained in the public Kangoo area. A Java wrapper has also been developed to allow the implementation of Java client applications able to interact with Kangoo.

As introduced above the only partially failed project core part is the integration of JML contracts in the projects composing the master thesis. As already stated in chapter 7 JML contracts are present on the code but only as simple Java comments. We will be able to compile the projects with JMLC once the new version of JML's specification will be published.

Due to the time lost trying to solve JML compilation problems I was not able to start working on the API extension to provide access to the user's private data contained in Kangoo. Once taken the decision to leave JML contracts as simple Java contracts the remaining time was not enough to introduce the important changes on Kangoo projects required to be able to work with the private Kangoo area. I decided so to concentrate my efforts on the development of a new sample application that provides a more intuitive user interaction.

## 10.2 Future Work

The main future step is the development of the private Kangoo API. The main aspects of this enhancement are the interaction with Kangoo grid storage system and user authentication mechanism (between user and sample application, sample application/third party application and the web API and finally also between web API and Kangoo storage system). These aspects have been marginally studied and possible implementation have been already discussed with the supervisor.

Surely the API will be modified according to the evolution of Kangoo. The collection of API methods could be enhanced with Kangoo search server functionalities. Also the amount of information returned by the API methods could be enhanced if new ones will be introduced in the

Kangoo metadata server. Returned information could for example be personalized according to the file type. It will be interesting to be able to provide a frame as video thumbnail or preview, or to associate to a music file information like title, author and distribution year. Also geographical data could be associated to files contained in Kangoo and returned with new API methods.

It will also be interesting to utilize the Kangoo web API for the development of mashup web applications. Mashup web applications are hybrid applications, which combine content from more than one source using their public APIs. A possible example of an API, which could be combined with Kangoo API, is the Google Map API. Using it, it will be possible to allow users to browse image files according to their geographical position using maps.

Once the Kangoo private API is implemented, the sample application could then be updated. This will also probably result in layout changes. Once published a first Kangoo beta version, it will also be interesting to publish the Kangoo API to get some interesting feedback from the developers that will use it. It will also be possible to detect unknown bugs.

## 10.3 Personal Opinion

This master gave me the opportunity to work on a really interesting project based on networked system and to deepen my knowledge in web applications, Java, servlets, XSLT, HTML, REST and of course JML. This has also been the first time I worked on a big real-world project. I get the chance to see how a big project is structured, how different components composing such a project interact and how new components could be implemented reusing existing parts.

During this mater thesis Kangoo was under a continuous evolution. Always new functionalities have been introduced and so always new methods could be implemented in the API. This parallel evolution of Kangoo gave me also the chance to provide inputs for possible enhancements of Kangoo that can be reused also in the API. I could also work on the design of online GUI development that result to be really interesting.

After preliminary studies, I expected to be able to start developing the private API but unfortunately a wrong estimation of possible problems leaves me no more time to work on it. This was a little bit disappointing since, as already introduced in the previous section, it would be probably really interesting to work on it.

# Appendix A

# API Documentation

## A.1 Introduction

The Kangoo Web API is composed of a set of callable methods and it is based on REST communication protocol. Method invocations are based on HTTP GET requests. The responses are contained in XML formatted data. Each method is mapped to a specific URL.

The next section provides an example for calling API methods using Java. Explanations are provided for how methods are invoked and how responses could be parsed in a DOM document.

Section 3 lists all methods implemented in the API. For each method, a small description is provided explaining its functionality and its response data. Additionally, for each method, the URL, arguments, responses and error codes are also listed.

## A.2 Java Example

The following Java code explains how to invoke the test.echo method. Method arguments are passed directly in the URL.

API invocation can be divided in two phases. In the first phase, a URL connection is established with the method address. During the second phase, the response is parsed and inserted in a DOM document.

Data contained in the response, as well as the parameter passed in the URL, are encoded using UTF-8.

```
//
String urlString =
  www.kangoo.com/api/test/echo?apiKey=0123456789abcdef&test_param=0";
try {
  URL url = new URL(urlString);
  URLConnection conn = url.openConnection();
  conn.setDoOutput(true);
  Document doc = parseResponse(conn);
} catch (Exception e) {
  //
}
```

```
//

protected Document parseResponse(URLConnection conn) throws IOException{
  // create a dom doc using the returned response
  Document doc = null;
  DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
  DocumentBuilder db;
  try {
    db = dbf.newDocumentBuilder();
    doc = db.parse(conn.getInputStream());
  } catch (Exception e) {
    //...
  }

  return doc;
}
```

## A.3 API Methods

### A.3.1 Archives

**getTopScoreArchives**

*Description*
This method returns the lists of available Kangoo archives. The response is divided according to the type. The archive information contained in the response could be used to invoke the items.getTop method.

*URL:* /api/archives/getTopScoreArchives

*Arguments*
**apiKey** (required)
Your API application key
**types** (optional, default value 1111111)
Types parameter is the binary representation of the item types desired and has the format groups|users|others|documents|musics|videos|images. To request items of type images and users the value to use is 0100001.

*Example Response*

```
<response method="com.wuala.webapi.archives.getTopScoreArchives">
  <archives count="18">
    <images count="9">
      <archive timePeriod="1" week="9" year="2007"/>
      <archive timePeriod="1" week="8" year="2007"/>
      <archive timePeriod="3" year="2007"/>
      <archive day="21" month="2" timePeriod="0" year="2007"/>
      <archive day="23" month="2" timePeriod="0" year="2007"/>
      <archive day="25" month="2" timePeriod="0" year="2007"/>
      <archive day="27" month="2" timePeriod="0" year="2007"/>
      <archive day="20" month="2" timePeriod="0" year="2007"/>
      <archive month="2" timePeriod="2" year="2007"/>
    </images>
    <videos count="9">
```

```
      <archive timePeriod="3" year="2007"/>
      <archive month="2" timePeriod="2" year="2007"/>
      <archive timePeriod="1" week="8" year="2007"/>
      <archive timePeriod="1" week="9" year="2007"/>
      <archive day="23" month="2" timePeriod="0" year="2007"/>
      <archive day="21" month="2" timePeriod="0" year="2007"/>
      <archive day="27" month="2" timePeriod="0" year="2007"/>
      <archive day="25" month="2" timePeriod="0" year="2007"/>
      <archive day="20" month="2" timePeriod="0" year="2007"/>
    </videos>
    </archives>
    <status>
        <id>0</id>
        <description>Operation succesfull</description>
    </status>
</response>
```

*Error Codes*
10 - API key is not active
11 - API key is not valid
100 - Kangoo API currently not available
101 - Search API currently not available
111111 - [DEBUG] Unexpected exception!!!

## A.3.2 Comments

### getItemList

*Description* This method returns the list of comments associated with the desired item. The response is composed of a list of comment elements containing the text and information about the creation time and author.

*URL:* /api/comments/getItemList

*Arguments*
**apiKey** (required)
Your API application key
**elementKey** (required)
Item element key
**ownerKey** (required)
Item owner key *Example Response*

```
<response method="com.kangoo.webapi.comments.getItemList">
  <comments count="1" file="/Matthias Hobi/Pictures/allalinpass.JPG">
       <comment author="Luzius" date="Wed Oct 18 01:31:47 CEST 2006">
      Nette Aussicht, fast so schön wie in Graubünden. ;)
        </comment>
    </comments>
    <status>
        <id>0</id>
        <description>Operation Succesfull</description>
    </status>
</response>
```

*Error Codes*
1 - Item not found
4 - Invalid arguments. [supplementary information]
10 - API key is not active
11 - API key is not valid
111111 - [DEBUG] Unexpected exception!!!

### A.3.3 Favorites

**getPublicList**

*Description*
This method returns a list of the users favorite public items.

*URL:* /api/favorites/getPublicList

*Arguments*
**apiKey** (required)
Your API application key
**username** (required)
Name of user to fetch the favorites list for

*Example Response*

```
<response method="com.kangoo.webapi.favorites.getPublicList" username="Cristian">
  <favorites count="3">
    <favorite  element_key="34_528_41" owner_key="34_528_12">
      <name>design2.jpg</name>
      <url>/kangoo/Cristian/test/design2.jpg</url>
      <path>/Cristian/test/design2.jpg</path>
      <owner_name>Cristian</owner_name>
    </favorite>
    <favorite  element_key="662_11" owner_key="34_528_12">
      <name>kangooprojects.pdf</name>
      <url>/kangoo/Cristian/test/kangooprojects.pdf</url>
      <path>/Cristian/test/kangooprojects.pdf</path>
      <owner_name>Cristian</owner_name>
    </favorite>
    <favorite  element_key="34_528_22" owner_key="34_528_12">
      <name>xbox_optic.jpg</name>
      <url>/kangoo/Cristian/test/xbox_optic.jpg</url>
      <path>/Cristian/test/xbox_optic.jpg</path>
      <owner_name>Cristian</owner_name>
    </favorite>
  </favorites>
  <status>
    <id>0</id>
    <description>OperationSuccessfull</description>
  </status>
</response>
```

*Error Codes*
2 - User not found
10 - API key is not active

11 - API key is not valid
111111 - [DEBUG] Unexpected exception!!!

### A.3.4 Groups

**searchGroupName**

*Description*
This method returns the list of groups matching the requested search. The relevance attribute contains the item score for the requested search.

*URL:* /api/groups/searchGroupName

*Arguments*
**apiKey** (required)
Your API application key
**groupName** (required)
Group name to search for
**like** (optional, default: false)
Specify if also similar group names must be returned in the response. If set to false, only exact matches are returned
**groupNumber** (optional, default 20)
Number of groups names returned in a page
**lastGroupFetched** (optional, default null)
Last group name fetched. Used to request next group names contained in the response. If not passed response start from the first group name available.

*Example Response*

```
<response groupname="web" like="false" method="com.kangoo.webapi.groups.searchGroupName">
  <results count="1" exec_time="16">
    <groups count="1"  estimated_count="1">
      <group element_key="27_1_13" owner_key="27_1_13" relevance="1.79">
        <name>Dominik Fan Club</name>
        <path>/Dominik Fan Club</path>
        <url>/kangoo/Dominik Fan Club</url>
        <views>1</views>
        <favorites>0</favorites>
        <impressions>0</impressions>
        <inappropriates>0</inappropriates>
        <trackbacks>0</trackbacks>
        <owner_name>Dominik Fan Club</owner_name>
      </user>
    </users>
  </results>
  <status>
    <id>0</id>
    <description>Operation succesfull</description>
  </status>
</response>
```

*Error Codes*
10 - API key is not active

11 - API key is not valid
100 - Kangoo API currently not available
101 - Search API currently not available
111111 - [DEBUG] Unexpected exception!!!

### A.3.5 Items

**search**

*Description*
This method returns the list of items matching the specified search. The response is divided according to the type. The number of returned results for each type is specified by the perPage argument. When the last items of a specific type are returned, the eof attribute flag is set to true in the root element of the type. A global eof flag is set to true if no more results are available for any of the types. If more pages are available the eof attribute is omitted.
For each item details and statistics are provided:
Details: file name, file URL, file path, owner name
Statistics: views (the number of users that download the item), favorites (number of users that insert the item in their favorite lists), impressions (the number of times the item has been returned in a search), inappropriates (the number of users signaling the item as inappropriate), trackbacks (the number of links to the item)
The relevance attribute contains the item score for the requested search.

*URL:* /api/items/search

*Arguments*
**apiKey** (required)
Your API application key
**text** (required)
Text to search for
**page** (optional, default: 1)
The page of results to return
**perPage** (optional, default: 20)
Number of items to return per page
**types** (optional, default value 1111111)
Types parameter is the binary representation of the item types desired and has the format groups|users|others|documents|musics|videos|images. To request items of type images and users the value to use is 0100001.
sortBy (optional, default: 0)
Order responses are returned. 0: sort by score, 1: sort by time

*Example Response*

```
<response method="com.kangoo.webapi.items.search" text="kangoo">
  <results eof="true" count=8 exec_time=17 page=2>
    <images eof="true" count=3 estimated_count="3">
      <image  element_key=34_528_41 owner_key=34_528_12 relevance="1.0">
        <name>design2.jpg</name>
        <url>/kangoo/Cristian/test/design2.jpg</url>
        <path>/Cristian/test/design2.jpg</path>
        <views>1</views>
        <favorites>0</favorites>
        <impressions>0</impressions>
        <inappropriates>0</inappropriates>
        <trackbacks>0</trackbacks>
        <owner_name>Cristian</owner_name>
      </image>
      <image  element_key=34_528_22 owner_key=34_528_12 relevance="1.0">
        <name>xbox_optic.jpg</name>
        <url>/kangoo/Cristian/test/xbox_optic.jpg</url>
        <path>/Cristian/test/xbox_optic.jpg</path>
        <views>5</views>
        <favorites>2</favorites>
        <impressions>4</impressions>
        <inappropriates>0</inappropriates>
        <trackbacks>0</trackbacks>
        <owner_name>Cristian</owner_name>
      </image>
      <image  element_key=34_1924_272 owner_key=34_1362_2 relevance="1.0">
        <name>P1000131.JPG</name>
        <url>/kangoo/Dominik Egger/Pictures/greece/P1000131.JPG</url>
        <path>/Dominik Egger/Pictures/greece/P1000131.JPG</path>
        <views>1</views>
        <favorites>0</favorites>
        <impressions>0</impressions>
        <inappropriates>0</inappropriates>
        <trackbacks>0</trackbacks>
        <owner_name>Dominik Egger</owner_name>
      </image>
    </images>
    <videos eof="true" count=2 estimated_count="2">
      <video  element_key=34_1924_434 owner_key=34_1362_2 relevance="1.0">
        <name>Stuntcity.wmv</name>
        <url>/kangoo/Dominik Egger/Video/Stuntcity.wmv</url>
        <path>/Dominik Egger/Video/Stuntcity.wmv</path>
        <views>2</views>
        <favorites>1</favorites>
        <impressions>0</impressions>
        <inappropriates>0</inappropriates>
        <trackbacks>0</trackbacks>
        <owner_name>Dominik Egger</owner_name>
      </video>
      <video  element_key=34_68_446 owner_key=34_66_2 relevance="1.0">
        <name>Küken-Song.mpg</name>
        <url>/kangoo/Dominik/Public/Küken-Song.mpg</url>
        <path>/Dominik/Public/Küken-Song.mpg</path>
        <views>1</views>
        <favorites>0</favorites>
        <impressions>0</impressions>
        <inappropriates>0</inappropriates>
        <trackbacks>0</trackbacks>
        <owner_name>Dominik</owner_name>
      </video>
    </videos>
    <musics eof="true" count=1 estimated_count="2">
      <music  elementKey=34_703_41 owner_key=34_66_2 relevance="1.0">
        <name>01 - Playground love.mp3</name>
        <url>/kangoo/Dominik/Public/01 - Playground love.mp3</url>
        <path>/Dominik/Public/01 - Playground love.mp3</path>
        <views>7</views>
        <favorites>3</favorites>
        <impressions>0</impressions>
        <inappropriates>0</inappropriates>
        <trackbacks>0</trackbacks>
        <owner_name>Dominik</owner_name>
      </music>
    </musics>
    <documents eof="true" count=1 estimated_count="1">
      <document  elementKey=34_662_11 owner_key=34_528_12 relevance="1.0">
        <name>kangooprojects.pdf</name>
        <url>/kangoo/Cristian/test/kangooprojects.pdf</url>
        <path>/Cristian/test/kangooprojects.pdf</path>
        <views>1</views>
        <favorites>0</favorites>
        <impressions>0</impressions>
        <inappropriates>0</inappropriates>
        <trackbacks>0</trackbacks>
        <owner_name>Cristian</owner_name>
      </document>
    </documents>
  </results>
  <status>
    <id>0</id>
    <description>Operation Successfull</description>
  </status>
</response>
```

*Error Codes*
4 - Invalid arguments. [supplementary information]
10 - API key is not active
11 - API key is not valid
100 - Kangoo API currently not available
101 - Search API currently not available
1000 - The requested page is empty
111111 - [DEBUG] Unexpected exception!!!

**getTop**

*Description*
This method returns the list of top items contained in Kangoo. If the year parameter is specified and is bigger than 0, the API retrieves the data from the Kangoo archives (for archives request timePeriod must be: timePeriod <= 3  timePeriod >= 0). Use archive.getTopArchiveList method to get list of available archives.
The response is divided according to the type. The number of returned results for each type is specified by the perPage argument. When the last items of a specific type are returned, the eof attribute flag is set to true in the root element of the type. A global eof flag is set to true if no more results are available for any of the types. If more pages are available the eof attribute is omitted.
For each item details and statistics are provided:
Details: file name, file URL, file path, owner name
Statistics: views (the number of users that download the item), favorites (number of users that

insert the item in their favorite lists), impressions (the number of times the item has been returned in a search), inappropriates (the number of users signaling the item as inappropriate), trackbacks (the number of links to the item)

*URL:* /api/items/getTop

*Arguments*
**apiKey** (required)
Your API application key
**page** (optional, default: 1)
The page of results to return
**perPage** (optional, default: 20)
Number of items to return per page
**types** (optional, default value 1111111)
Types parameter is the binary representation of the item types desired and has the format groups|users|others|documents|musics|videos|images. To request items of type images and users the value to use is 0100001.
**timePeriod** (optional, defaul: All Times (Year if archive request))
Specify the time period to use for the top request. Allowed values:
0 - Day, 1 - Week, 2 - Month, 3 - Year, 4 - All Times
If archive data is requested, allowed values are:
0 - Day, 1 - Week, 2 - Month, 3 - Year
**year** (optional)
Year of desired archive
**month** (optional)
Month of desired archive [1,12]
**week** (optional)
Week of desired archive [1,52]
**day** (optional)
Day of desired archive [1,31]

*Example Response*

```
<response method="com.kangoo.webapi.items.getTop">
  <results eof="true" count=8 exec_time=17 page=2>
    <images eof="true" count=3 estimated_count="100">
      <image  element_key=34_528_41 owner_key=34_528_12>
        <name>design2.jpg</name>
        <url>/kangoo/Cristian/test/design2.jpg</url>
        <path>/Cristian/test/design2.jpg</path>
        <views>1</views>
        <favorites>0</favorites>
        <impressions>0</impressions>
        <inappropriates>0</inappropriates>
        <trackbacks>0</trackbacks>
        <owner_name>Cristian</owner_name>
      </image>
      <image  element_key=34_528_22 owner_key=34_528_12>
        <name>xbox_optic.jpg</name>
        <url>/kangoo/Cristian/test/xbox_optic.jpg</url>
        <path>/Cristian/test/xbox_optic.jpg</path>
        <views>5</views>
        <favorites>2</favorites>
        <impressions>4</impressions>
        <inappropriates>0</inappropriates>
        <trackbacks>0</trackbacks>
        <owner_name>Cristian</owner_name>
      </image>
      <image  element_key=34_1924_272 owner_key=34_1362_2>
        <name>P1000131.JPG</name>
        <url>/kangoo/Dominik Egger/Pictures/greece/P1000131.JPG</url>
        <path>/Dominik Egger/Pictures/greece/P1000131.JPG</path>
        <views>1</views>
        <favorites>0</favorites>
        <impressions>0</impressions>
        <inappropriates>0</inappropriates>
        <trackbacks>0</trackbacks>
        <owner_name>Dominik Egger</owner_name>
      </image>
    </images>
    <videos eof="true" count=2 estimated_count="100">
      <video  element_key=34_1924_434 owner_key=34_1362_2>
        <name>Stuntcity.wmv</name>
        <url>/kangoo/Dominik Egger/Video/Stuntcity.wmv</url>
        <path>/Dominik Egger/Video/Stuntcity.wmv</path>
        <views>2</views>
        <favorites>1</favorites>
        <impressions>0</impressions>
        <inappropriates>0</inappropriates>
        <trackbacks>0</trackbacks>
        <owner_name>Dominik Egger</owner_name>
      </video>
      <video  element_key=34_68_446 owner_key=34_66_2>
        <name>Küken-Song.mpg</name>
        <url>/kangoo/Dominik/Public/Küken-Song.mpg</url>
        <path>/Dominik/Public/Küken-Song.mpg</path>
        <views>1</views>
        <favorites>0</favorites>
        <impressions>0</impressions>
        <inappropriates>0</inappropriates>
        <trackbacks>0</trackbacks>
        <owner_name>Dominik</owner_name>
      </video>
    </videos>
    <musics eof="true" count=1 estimated_count="100">
      <music  elementKey=34_703_41 owner_key=34_66_2>
        <name>01 - Playground love.mp3</name>
        <url>/kangoo/Dominik/Public/01 - Playground love.mp3</url>
        <path>/Dominik/Public/01 - Playground love.mp3</path>
        <views>7</views>
        <favorites>3</favorites>
        <impressions>0</impressions>
        <inappropriates>0</inappropriates>
        <trackbacks>0</trackbacks>
        <owner_name>Dominik</owner_name>
      </music>
    </musics>
    <documents eof="true" count=1 estimated_count="100">
```

```
      <document  elementKey=34_662_11 owner_key=34_528_12>
        <name>kangooprojects.pdf</name>
        <url>/kangoo/Cristian/test/kangooprojects.pdf</url>
        <path>/Cristian/test/kangooprojects.pdf</path>
        <views>1</views>
        <favorites>0</favorites>
        <impressions>0</impressions>
        <inappropriates>0</inappropriates>
        <trackbacks>0</trackbacks>
        <owner_name>Cristian</owner_name>
      </document>
    </documents>
  </results>
  <status>
    <id>0</id>
    <description>Operation Successfull</description>
  </status>
</response>
```

*Error Codes*
4 - Invalid arguments. [supplementary information]
10 - API key is not active
11 - API key is not valid
100 - Kangoo API currently not available
101 - Search API currently not available
1000 - The requested page is empty
111111 - [DEBUG] Unexpected exception!!!

**getRecent**

*Description*
This method returns the list of recently added items contained in Kangoo. The response is divided according to the type. The number of returned results for each type is specified by the perPage argument. When the last items of a specific type are returned, the eof attribute flag is set to true in the root element of the type. A global eof flag is set to true if no more results are available for any of the types. If more pages are available the eof attribute is omitted.
For each item details and statistics are provided:
Details: file name, file URL, file path, owner name
Statistics: views (the number of users that download the item), favorites (number of users that insert the item in their favorite lists), impressions (the number of times the item has been returned in a search), inappropriates (the number of users signaling the item as inappropriate), trackbacks (the number of links to the item)

*URL:* /api/items/getRecent

*Arguments*
**apiKey** (required)
Your API application key
**page** (optional, default: 1)
The page of results to return
**perPage** (optional, default: 20)
Number of items to return per page
**types** (optional, default value 1111111)

Types parameter is the binary representation of the item types desired and has the format groups|users|others|documents|musics|videos|images. To request items of type images and users the value to use is 0100001.

*Example Response*

```
<response method="com.kangoo.webapi.items.getRecent">
  <results eof="true" count=8 exec_time=17 page=2>
    <images eof="true" count=3 estimated_count="100">
      <image  element_key=34_528_41 owner_key=34_528_12>
        <name>design2.jpg</name>
        <url>/kangoo/Cristian/test/design2.jpg</url>
        <path>/Cristian/test/design2.jpg</path>
        <views>1</views>
        <favorites>0</favorites>
        <impressions>0</impressions>
        <inappropriates>0</inappropriates>
        <trackbacks>0</trackbacks>
        <owner_name>Cristian</owner_name>
      </image>
      <image  element_key=34_528_22 owner_key=34_528_12>
        <name>xbox_optic.jpg</name>
        <url>/kangoo/Cristian/test/xbox_optic.jpg</url>
        <path>/Cristian/test/xbox_optic.jpg</path>
        <views>5</views>
        <favorites>2</favorites>
        <impressions>4</impressions>
        <inappropriates>0</inappropriates>
        <trackbacks>0</trackbacks>
        <owner_name>Cristian</owner_name>
      </image>
      <image  element_key=34_1924_272 owner_key=34_1362_2>
        <name>P1000131.JPG</name>
        <url>/kangoo/Dominik Egger/Pictures/greece/P1000131.JPG</url>
        <path>/Dominik Egger/Pictures/greece/P1000131.JPG</path>
        <views>1</views>
        <favorites>0</favorites>
        <impressions>0</impressions>
        <inappropriates>0</inappropriates>
        <trackbacks>0</trackbacks>
        <owner_name>Dominik Egger</owner_name>
      </image>
    </images>
    <videos eof="true" count=2 estimated_count="100">
      <video  element_key=34_1924_434 owner_key=34_1362_2>
        <name>Stuntcity.wmv</name>
        <url>/kangoo/Dominik Egger/Video/Stuntcity.wmv</url>
        <path>/Dominik Egger/Video/Stuntcity.wmv</path>
        <views>2</views>
        <favorites>1</favorites>
        <impressions>0</impressions>
        <inappropriates>0</inappropriates>
        <trackbacks>0</trackbacks>
        <owner_name>Dominik Egger</owner_name>
      </video>
      <video  element_key=34_68_446 owner_key=34_66_2>
        <name>Küken-Song.mpg</name>
        <url>/kangoo/Dominik/Public/Küken-Song.mpg</url>
        <path>/Dominik/Public/Küken-Song.mpg</path>
        <views>1</views>
        <favorites>0</favorites>
        <impressions>0</impressions>
        <inappropriates>0</inappropriates>
        <trackbacks>0</trackbacks>
        <owner_name>Dominik</owner_name>
      </video>
    </videos>
    <musics eof="true" count=1 estimated_count="100">
      <music  elementKey=34_703_41 owner_key=34_66_2>
        <name>01 - Playground love.mp3</name>
        <url>/kangoo/Dominik/Public/01 - Playground love.mp3</url>
        <path>/Dominik/Public/01 - Playground love.mp3</path>
        <views>7</views>
        <favorites>3</favorites>
        <impressions>0</impressions>
        <inappropriates>0</inappropriates>
        <trackbacks>0</trackbacks>
        <owner_name>Dominik</owner_name>
      </music>
    </musics>
    <documents eof="true" count=1 estimated_count="100">
      <document  elementKey=34_662_11 owner_key=34_528_12>
        <name>kangooprojects.pdf</name>
        <url>/kangoo/Cristian/test/kangooprojects.pdf</url>
        <path>/Cristian/test/kangooprojects.pdf</path>
        <views>1</views>
        <favorites>0</favorites>
        <impressions>0</impressions>
        <inappropriates>0</inappropriates>
        <trackbacks>0</trackbacks>
        <owner_name>Cristian</owner_name>
      </document>
    </documents>
  </results>
  <status>
    <id>0</id>
    <description>Operation Successfull</description>
  </status>
</response>
```

*Error Codes*
4 - Invalid arguments. [supplementary information]
10 - API key is not active
11 - API key is not valid
100 - Kangoo API currently not available
101 - Search API currently not available
1000 - The requested page is empty
111111 - [DEBUG] Unexpected exception!!!

**getPearls**

*Description*
This method returns the list of pearl items contained in Kangoo. Pearl items are recently added items that have acquired a good reputation during their short life in Kangoo. Due to their young age they still arent contained in the top item list, but probably in short time they will be inserted in the top item list.
The response is divided according to the type. The number of returned results for each type is specified by the perPage argument. When the last items of a specific type are returned, the eof attribute flag is set to true in the root element of the type. A global eof flag is set to true if no more results are available for any of the types. If more pages are available the eof attribute is omitted.
For each item details and statistics are provided:
Details: file name, file URL, file path, owner name
Statistics: views (the number of users that download the item), favorites (number of users that insert the item in their favorite lists), impressions (the number of times the item has been returned in a search), inappropriates (the number of users signaling the item as inappropriate), trackbacks (the number of links to the item)

*URL:* /api/items/getPearls

*Arguments*
**apiKey** (required)
Your API application key
**page** (optional, default: 1)
The page of results to return
**perPage** (optional, default: 20)
Number of items to return per page
**types** (optional, default value 1111111)
Types parameter is the binary representation of the item types desired and has the format groups|users|others|documents|musics|videos|images. To request items of type images and users the value to use is 0100001.

*Example Response*

```
<response method="com.kangoo.webapi.items.getPearls">
  <results eof="true" count=8 exec_time=17 page=2>
    <images eof="true" count=3 estimated_count="100">
      <image  element_key=34_528_41 owner_key=34_528_12>
        <name>design2.jpg</name>
        <url>/kangoo/Cristian/test/design2.jpg</url>
        <path>/Cristian/test/design2.jpg</path>
        <views>1</views>
        <favorites>0</favorites>
        <impressions>0</impressions>
        <inappropriates>0</inappropriates>
        <trackbacks>0</trackbacks>
        <owner_name>Cristian</owner_name>
      </image>
      <image  element_key=34_528_22 owner_key=34_528_12>
        <name>xbox_optic.jpg</name>
        <url>/kangoo/Cristian/test/xbox_optic.jpg</url>
        <path>/Cristian/test/xbox_optic.jpg</path>
        <views>5</views>
        <favorites>2</favorites>
        <impressions>4</impressions>
        <inappropriates>0</inappropriates>
        <trackbacks>0</trackbacks>
        <owner_name>Cristian</owner_name>
      </image>
      <image  element_key=34_1924_272 owner_key=34_1362_2>
        <name>P1000131.JPG</name>
        <url>/kangoo/Dominik Egger/Pictures/greece/P1000131.JPG</url>
        <path>/Dominik Egger/Pictures/greece/P1000131.JPG</path>
        <views>1</views>
        <favorites>0</favorites>
        <impressions>0</impressions>
        <inappropriates>0</inappropriates>
        <trackbacks>0</trackbacks>
        <owner_name>Dominik Egger</owner_name>
      </image>
    </images>
    <videos eof="true" count=2 estimated_count="100">
      <video  element_key=34_1924_434 owner_key=34_1362_2>
        <name>Stuntcity.wmv</name>
        <url>/kangoo/Dominik Egger/Video/Stuntcity.wmv</url>
        <path>/Dominik Egger/Video/Stuntcity.wmv</path>
        <views>2</views>
        <favorites>1</favorites>
        <impressions>0</impressions>
        <inappropriates>0</inappropriates>
        <trackbacks>0</trackbacks>
        <owner_name>Dominik Egger</owner_name>
      </video>
      <video  element_key=34_68_446 owner_key=34_66_2>
        <name>Küken-Song.mpg</name>
        <url>/kangoo/Dominik/Public/Küken-Song.mpg</url>
        <path>/Dominik/Public/Küken-Song.mpg</path>
        <views>1</views>
        <favorites>0</favorites>
        <impressions>0</impressions>
        <inappropriates>0</inappropriates>
        <trackbacks>0</trackbacks>
        <owner_name>Dominik</owner_name>
      </video>
    </videos>
    <musics eof="true" count=1 estimated_count="100">
      <music  elementKey=34_703_41 owner_key=34_66_2>
        <name>01 - Playground love.mp3</name>
        <url>/kangoo/Dominik/Public/01 - Playground love.mp3</url>
        <path>/Dominik/Public/01 - Playground love.mp3</path>
        <views>7</views>
        <favorites>3</favorites>
        <impressions>0</impressions>
        <inappropriates>0</inappropriates>
        <trackbacks>0</trackbacks>
        <owner_name>Dominik</owner_name>
      </music>
    </musics>
    <documents eof="true" count=1 estimated_count="100">
      <document  elementKey=34_662_11 owner_key=34_528_12>
        <name>kangooprojects.pdf</name>
        <url>/kangoo/Cristian/test/kangooprojects.pdf</url>
        <path>/Cristian/test/kangooprojects.pdf</path>
        <views>1</views>
        <favorites>0</favorites>
        <impressions>0</impressions>
        <inappropriates>0</inappropriates>
        <trackbacks>0</trackbacks>
        <owner_name>Cristian</owner_name>
      </document>
    </documents>
  </results>
  <status>
    <id>0</id>
    <description>Operation Successfull</description>
  </status>
</response>
```

*Error Codes*
4 - Invalid arguments. [supplementary information]
10 - API key is not active
11 - API key is not valid
100 - Kangoo API currently not available
101 - Search API currently not available
1000 - The requested page is empty
111111 - [DEBUG] Unexpected exception!!!

**getDetails**

*Description*
This method returns details and statistics about the requested item. Data returned contains:
Details: file name, file URL, file path, owner name
Statistics: views (the number of users that download the item), favorites (number of users that insert the item in their favorite lists), impressions (the number of times the item has been returned in a search), inappropriates (the number of users signaling the item as inappropriate), trackbacks (the number of links to the item)

*URL:* /api/items/getDetails

*Arguments*
**apiKey** (required)
Your API application key
**elementKey** (required)
Item element key
**ownerKey** (required)
Item owner key

*Example Response*

```
<response method="com.kangoo.webapi.items.getDetails">
  <details element_key="26_10107_11" owner_key="26_2459_13">
    <name>MirrorWater.jpg</name>
    <url>/kangoo/Dummy_blabla48/flickr/MirrorWater.jpg</url>
    <path>/Dummy_blabla48/flickr/MirrorWater.jpg</path>
    <views>857</views>
    <favorites>0</favorites>
    <impressions>56573</impressions>
    <inappropriates>0</inappropriates>
    <trackbacks>0</trackbacks>
    <owner_name>Dummy_blabla48</owner_name>
  </details>
  <status>
    <id>0</id>
    <description>Operation Succesfull</description>
  </status>
</response>
```

*Error Codes*
1 - Item not found
4 - Invalid arguments. [supplementary information]
10 - API key is not active
11 - API key is not valid
111111 - [DEBUG] Unexpected exception!!!

### A.3.6 Tags

#### getItemList

*Description*
This method returns all the tags associated with a specific item. For each tag the author name is returned.

*URL:* /api/tags/getItemList

*Arguments*
**apiKey** (required)
Your API application key
**elementKey** (required)
Item element key
**ownerKey** (required)
Item owner key

*Example Response*

```
<response method="com.kangoo.webapi.tags.getItemList">
  <item elementKey="2065_53_11" path="/Dummy_raphi/flickr/00020015.jpg" tags="4"
            url="kangoo/Dummy_raphi/flickr/00020015.jpg" userKey="2065_1_11">
    <tag author="Dummy_raphi">japan</tag>
    <tag author="Dummy_raphi">lomo</tag>
    <tag author="Dummy_raphi">okayama</tag>
    <tag author="Dummy_raphi">top20favorites</tag>
  </item>
  <status>
```

```
    <id>0</id>
    <description>Operation succesfull</description>
  </status>
</response>
```

*Error Codes*
1 - Item not found
4 - Invalid arguments. [supplementary information]
10 - API key is not active
11 - API key is not valid
111111 - [DEBUG] Unexpected exception!!!

### getMostPopular

*Description*
This method returns the list of all most popular tags associated to the desired item types is returned. For each tag the corresponding score is returned. The response is divided according to the type.

*URL:* /api/tags/getMostPopular

*Arguments*
**apiKey** (required)
Your API application key
**types** (optional, default value 1111111)
Types parameter is the binary representation of the item types desired and has the format groups|users|others|documents|musics|videos|images. To request items of type images and users the value to use is 0100001.

*Example Response*

```
<response method="com.kangoo.webapi.tags.getMostPopular">
  <tags count="8" exec_time="19" max_score="3">
    <images count="8" max_score="2">
      <tag score="2">top20favorites</tag>
      <tag score="2">sweet</tag>
      <tag score="2">okayama</tag>
      <tag score="2">new york</tag>
      <tag score="1">abigfave</tag>
      <tag score="1">MIT</tag>
      <tag score="1">30secs</tag>
      <tag score="1">ski</tag>
    </images>
    <videos count="1" max_score="3">
      <tag score="3">california</tag>
    </video>
    <musics count="0" max_score="0"/>
  </tags>
  <status>
    <id>0</id>
    <description>Operation succesfull</description>
  </status>
</response>
```

*Error Codes*
10 - API key is not active
11 - API key is not valid
100 - Kangoo API currently not available
101 - Search API currently not available

**getTagGeist**

*Description*
This method returns the list of tags contained in Kangoo specifying the date when they have been inserted. The response is divided according to the type. If year argument is specified and is bigger than 0, the API retrieves the data for the specified past time period. Otherwise the method returns tags considering today as the starting date.

*URL:* /api/tags/getTagGeist

*Arguments*
**apiKey** (required)
Your API application key
**types** (optional, default value 1111111)
Types parameter is the binary representation of the item types desired and has the format groups|users|others|documents|musics|videos|images. To request items of type images and users the value to use is 0100001.
**timePeriod** (optional, defaul: Day)
Specify the time period to use for the top request.
0 - Day, 1 - Week, 2 - Month, 3 - Year
4 - All Times NOT ALLOWED!
**year** (optional)
Year of desired past period
**month** (optional)
Month of desired past period
**week** (optional)
Week of desired past period [1,52]
**day** (optional) Day of desired past period [1,31]

*Example Response*

```
<response method="com.kangoo.webapi.tags.getTagGeist">
  <tags count="9" exec_time="19">
    <images count="8">
      <tag date="Tue Jan 02 14:06:28 CET 2007">montag</tag>
      <tag date="Tue Jan 02 14:06:28 CET 2007">heute</tag>
      <tag date="Mon Jan 01 15:20:54 CET 2007">sonne</tag>
      <tag date="Mon Jan 01 15:20:54 CET 2007">kaffee</tag>
      <tag date="Mon Jan 01 15:17:56 CET 2007">kaffee</tag>
      <tag date="Wed Dec 27 10:54:26 CET 2006">casting</tag>
      <tag date="Wed Dec 27 10:54:25 CET 2006">las vegas</tag>
      <tag date="Wed Dec 27 10:54:25 CET 2006">national park</tag>
    </images>
    <videos count="1">
      <tag date="Wed Dec 27 11:54:35 CET 2006">california</tag>
    </video>
    <musics count="0"/>
  </tags>
  <status>
    <id>0</id>
    <description>Operation succesfull</description>
  </status>
</response>
```

*Error Codes*
10 - API key is not active
11 - API key is not valid
100 - Kangoo API currently not available
101 - Search API currently not available
1000 - The requested page is empty
111111 - [DEBUG] Unexpected exception!!!

**getRelatedTags**

*Description*
This method returns the list of related tags with the associated score for the specified tag. The response is divided according to the type. To pass multiple tags as input set the multipleTagsFlag parameter to true and pass the multiple tags in the argument tag separated by a comma.

*URL:* /api/tags/getRelatedTags

*Arguments*
**apiKey** (required)
Your API application key
**tag** (required)
The tag to fetch related tags for. Single tag or comma separated tags if multipleTagsFlag is passed as parameter (e.g usa,new york,ny)
**types** (optional, default value 1111111)
Types parameter is the binary representation of the item types desired and has the format groups|users|others|documents|musics|videos|images. To request items of type images and users the value to use is 0100001.
**multipleTagsFlag** (optional, default: false)
Passed to signal that tag parameter contains comma separated tags, e.g usa,new york,ny

*Example Response*

```
<response method="com.wuala.webapi.tags.getRelatedTags">
  <tags count="5" exec_time="19" max_score="87">
    <images count="5" max_score="87">
      <tag score="87">california</tag>
      <tag score="78">ca</tag>
      <tag score="73">photoshop</tag>
      <tag score="73">interestingness</tag>
      <tag score="70">us</tag>
    </images>
    <videos count="0" max_score="0"/>
  </tags>
  <status>
    <id>0</id>
    <description>Operation succesfull</description>
  </status>
```

```
</response>
```

*Error Codes*
4 - Invalid arguments. [supplementary information]
10 - API key is not active
11 - API key is not valid
100 - Kangoo API currently not available
101 - Search API currently not available
111111 - [DEBUG] Unexpected exception!!!

### A.3.7 Test

**echo**

*Description*
This method echoes the list of arguments passed.

*URL:* /api/test/echo

*Arguments*
**apiKey** (required)
Your API application key

*Example Response*

```
<response method="com.kangoo.webapi.test.Echo">
  <results>
    <apiKey>1234567890123456</apiKey>
    <test>cristian</test>
  </results>
  <status>
    <id>0</id>
    <description>Operation succesfull</description>
  </status>
</response>
```

*Error Codes*
10 - API key is not active
11 - API key is not

**checkAPIKey**

*Description*
This method returns the status of an API key through the response error code.

*URL:* /api/test/checkAPIKey

*Arguments*
**apiKey** (required)
Your API application key

*Example Response*

```
<response method="com.kangoo.webapi.test.checkAPIKey">
```

```
  <status>
    <id>0</id>
    <description>Operation succesfull</description>
  </status>
</response>
```

*Error Codes*
10 - API key is not active
11 - API key is not

### A.3.8 Thumbnail

**thumbnailURL**

*Description*
This method returns the preview in jpg format for image files. The desired image file is passed directly as path in the URL. This method could be used as input for element img in HTML pages.

*URL: /**

*Arguments*
The path is encoded directly in the URL request
Example: for file with path /Dominik/Public/Pictures/USA/DSC03895.JPG the URL is http://www.kangoo.com/Dominik/Public/Pictures/USA/DSC03895.JPG?thumb=320
**thumb** (required)
Defines preview size. Allowed values: 96 and 320.

*Response*
This method returns the image in jpg format.

*Error Codes*
Error image returned to handle eventual errors (preview not found, thumbnail not found)

### A.3.9 Users

**getIcon**

*Description*
This method returns the preview in jpg format of the icon associated with a user. It could be used as input for element img in HTML pages.

*URL:* /api/users/getIcon

*Arguments*
**username** (required)
Username to fetch image for.
**size** (optional, default: 96)
Defines preview size. Allowed values: 96 and 320.

*Response*
This method returns the image of the icon associate to the user in jpg format.

*Error Codes*

Standard image returned when desired preview not found.

### searchUserName

*Description*
This method returns the list of users matching the requested search. The relevance attribute contains the item score for the requested search.

*URL:* /api/users/searchUserName

*Arguments*
**apiKey** (required)
Your API application key
**userName** (required)
User name to search for
**like** (optional, default: false)
Specify if also similar user names must be returned. If set to true, only exact matches are returned
**userNumber** (optional, default 20)
Number of user names returned in a page
**lastUserFetched** (optional, default null)
Last username fetched. Used to request next usernames contained in the response. If not passed response start from the first username available.

*Example Response*

```
<response like="true" method="com.kangoo.webapi.users.searchUserName" username="Dom">
  <results count="1" exec_time="16">
    <users count="1"  estimated_count="1">
      <user element_key="27_1_13" owner_key="27_1_13" relevance="1.79">
        <name>Dominik</name>
        <path>/Dominik</path>
        <url>/kangoo/Dominik</url>
        <views>1</views>
        <favorites>0</favorites>
        <impressions>0</impressions>
        <inappropriates>0</inappropriates>
        <trackbacks>0</trackbacks>
        <owner_name>Dominik</owner_name>
      </user>
    </users>
  </results>
  <status>
    <id>0</id>
    <description>Operation succesfull</description>
  </status>
</response>
```

*Error Codes*
10 - API key is not active
11 - API key is not valid
100 - Kangoo API currently not available
101 - Search API currently not available
111111 - [DEBUG] Unexpected exception!!!

# Bibliography

[1] Google APIs. http://code.google.com/apis/.

[2] Yahoo! APIs. http://developer.yahoo.com/.

[3] Flickr API Documentation. http://www.flickr.com/services/api/.

[4] Oracle Berkeley DB Java Edition. http://www.oracle.com/database/berkeley-db/je/index.html.

[5] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures.* PhD thesis, University of California, Irvine, 2000.

[6] Dominik Grolimund. *Redundancy Management and Reliability in Kangoo*, 2006. http://dcg.ethz.ch/theses.html.

[7] Dominik Grolimund and Cristian Garcia. *Kangoo Web API Documentation*, February 2007.

[8] XML-RPC Homepage. http://www.xmlrpc.com/.

[9] Gary T. Leavens, Erik Poll, Curtis Clifton, Yoonsik Cheon, Clyde Ruby, David Cok, Peter Muller, Joseph Kiniry, and Patrice Chalin. *JML Reference Manual*, August 2006.

[10] Luzius Meisser. *File Abstraction, Transactions and Caching in Kangoo*, 2006. http://dcg.ethz.ch/theses.html.

[11] Document Object Model. http://java.sun.com/j2se/1.5.0/docs/api/org/w3c/dom/package-summary.html.

[12] W3C Soap Recommendation. http://www.w3.org/tr/2003/rec-soap12-part0-20030624/.

[13] Stefan Schmid. *Implementation of the Kangoo Distributed Hash Table*, 2005. http://dcg.ethz.ch/theses.html.

[14] Apache Tomcat. http://tomcat.apache.org/.

[15] Jason Hunter with William Crawford. *Java Servlet Programming.* O'Really, first edition, 1998.