Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Local Browsing

Information Seeking Tailored to Mobile Devices

Semester Thesis

Sebastian Wendland, Samuel Zihlmann, Dominik Bucher, D-ITET
wendlans@ee.ethz.ch, samuezih@ee.ethz.ch, dobucher@ee.ethz.ch

September 2012

Management of Information Systems / Computer Science Group,
ETH Zurich, 8092 Zurich

Supervisor Dr. Fabio Magagna
Prof. Dr. Juliana Sutanto / Prof. Dr. Bernhard Plattner

With Assistance from Simon Schweingruber, Nokia Switzerland

Contents

Abstract

The motivation for this paper is to improve search on mobile devices. As search usually is performed in the same way as on the desktop, search providers focus on depth, i.e. to find documents about very specific topics. However, search on mobile devices is much more diverse and people often want to browse the web for interesting things around them, for news articles or for content that other people tagged as being valuable.

The paper presents the implementation of a search engine optimized for mobile devices called Local Browsing. The application consists of a backend which is responsible for crawling the web and recommending datasets to users. The crawler fetches keywords, which are selected from web sites by frequency and windowing, further processed, filtered and weighted. The recommender on the other hand provides users with the best possible results based on several user inputs. These inputs include keywords, location, user history, general history and more. The recommendation is done by aggregation and weighting of matching results. The frontend is implemented independently and connects to the backend via the Local Browsing REST API. To find an optimized user interface, typical search use cases on mobile devices are analyzed and compared.

As result we present a running prototype which is compared to different search providers. In comparison it outperforms other search engines for typical mobile searches. For general depth searches a comparison is difficult as our current dataset only covers a tiny bit of all the information available on the web.

Introduction

Mobile internet is gaining on importance. It is expected that in short future more people access the internet from their mobile phones than from desktop computers [1]. Studies have shown that searches on mobile devices vary greatly from those on desktop computers [2] [3]. Whilst we mostly perform depth-searches on the desktop, we tend to do quick surface and transactional searches on a mobile phone [4]. These quick searches include queries for stuff which is physically close to us (e.g. restaurants, bars, theaters, bus stops, and more), which depend on our situation (e.g. train schedules, hotels and taxi telephone numbers when we just arrived at an airport) or just searches to discover fun stuff which is around our current location. Nevertheless the experience in using the mobile internet for search has mainly stayed the same as on the desktop.

To overcome the limitations of current search engines, the Chair of Management of Information Systems at the ETH Zurich has been doing research for several years in analyzing how people seek information on their mobile phones and how this process can be optimized [5] [6] [7] [8]. This thesis covers the implementation and concretization of a system as proposed in [9]. The completed system can be split into two parts: The backend and the user interface.

The logic working in the **background** of a search engine system has a huge impact on our perception of the system (even though as users we only interact with the system through the user interface). Next to general parts like a database, logging system, job system and server administration this backend primarily means the crawler and the recommender. Whilst a crawler collects, filters, assembles and stores data from the internet, a recommender aggregates a set of search results based on various user inputs. Based on the studies in [9] we take the proposed solutions for a recommender further by implementing and adjusting them. In this thesis there is also some research conducted on a specialized keyword crawler that aggregates keywords from websites, which is in stark contrast to the usual full-text indexing and allows for more efficient storage of the web site indexes. It also allows the user to search for specific keywords and the system to recommend such keywords to the user.

The implementation of the **user interface** is done on Windows Phone 7 thanks to assistance on this platform from Nokia Switzerland. The application is tested with different search queries against existing solutions like Google, Bing, local.ch and Nokia Places. The performance is measured in two ways:

- Number of inputs: The optimized application layout should allow a user to find the desired results with as little clicks as possible. If a user has to click, clicks on bigger buttons (the on-screen keyboard features tiny buttons for example) are better.
- Satisfaction and acceptance of the new system: This is measured by counting how many of the desired results the users find, how often users use the application, what their preferred search queries are (if users still type in keywords, as they would with any traditional search engine, the new approach didn't provide any benefits to the user).

The result of this thesis is a fully working system for search tailored to mobile phones which is further extendable to conduct more research, tests and optimizations. The phone application will be made available for download from the Windows Marketplace in short future.

Problem Definition

Starting from the fact that text input is rather clumsy on a mobile phone, a basic requirement for a mobile search engine system is to reduce the amount of necessary text input. Also, people tend to search for different things on their phones than they do on desktop computers. The search is rather transactional, location-based, situation-based or for entertaining purpose, in contrast to an in-depth search on a desktop computer [3] [4] [2] [9].

The solution provided by this thesis is based on keywords. If the user doesn't have to type in his own keywords but instead can select some from a given set, the number of clicks can be greatly reduced. There are two main points where a search engine based on keywords has to excel:

- Providing the user with interesting keywords: When the user starts the application he wants to see all keywords that could possibly interest him so he doesn't have to enter them himself.
- Providing the user with interesting results: Given a set of selected keywords, the results have to be what the user searched for, given his input, location and situation.

Of course, in order to optimize on both these points, a lot of background work has to be done. A specialized crawler has to search for keywords, locations and possibly situation-interesting elements on websites and other data on the internet. This data has to be stored in a meaningful way that also allows to weight different results. Like with any search engine robot, decisions have to be made which information should be stored. Once the data is available, it has to be cleaned and processed periodically, checked for faults and also be represented in a meaningful way for the developers themselves.

As it is impossible to know what users will find interesting without collecting data from users, the backend system also has to provide user or device identification and storage and processing of user generated data. This data includes search histories, locations and browse histories.

Finally, the recommender has to aggregate the data from all the different sources and output a sorted list of results. For this task, four categories are distinguished and aggregated in the end:

- Location based results: Results that are interesting because they are physically close to the user.
- User history based results: Results that a user accessed before.
- General history based results: Results that a majority of users has accessed before and finds interesting.
- Situation based results: Results based on the users situation. As an example think of web sites of cafés in the morning and web sites of cinemas in the evening.

The system can expect the following input from the application: user identification, location, keywords, time. This list can be extended so the resulting system must be modular enough to be extendable in future.

The complete application should try to optimize on the following points:

- Clicks to results: Users should have to press the least amount of clicks to get to the desired result. Clicks themselves can be categorized, whereas a click on a small button is less convenient than a click on a big button. Text input thus takes a very bad stand as the keyboard buttons are usually very small.
- Location-based results: The mobile search engine should be aware of the position of the user to be able to respond readily to requests for points of interests around the user. This also induces a location-awareness of the search results (i.e. web sites) themselves.
- Situation-based results: Depending on the time of day, the current weather, if the user is at home or not, if he arrived somewhere newly, if he is doing sports, and so on, the search application should adjust the results.
- Browse-factor: The search application should be entertaining and allow the user to easily discover and browse for possibly interesting stuff.

And as for every application that wants to get accepted by users, it should of course also be appealing, fast and fun.

Related Work

There has been done a lot of research on **search engines** and **recommenders**, both at universities and in industry. Also, a lot of work has been done on **keyword** and **location extraction**. For all the topics some of the work that is closely related to this thesis (in the context of mobile search engines and keyword based search) is presented.

The paper [10] describes a way to extract keywords and use them to cluster web sites with the intention to create tag clouds that give the user possibility to navigate within his search. The cloud generation is done via self-organizing maps that take into account the textual content of tagged documents. In [11] keywords are extracted from Wikipedia article plaintexts by applying the Term Frequency-Inverse Document Frequency algorithm, which is further described in [12]. The way the algorithm works is stated as follows by the paper: "*The key assumption is that if a term appears frequently in one document, but not in the overall collection, it is reasonable to consider it particularly descriptive for this document*" [11]. The algorithm was refined by applying it to terms of one, two and three words length which were compared to a general-purpose reference corpus compiled from newspaper texts.

The document [13] elaborates on the way of how to link web sites to locations (i.e. by extraction of a location from the unstructured data) and also on how to use the data to create link distance between topical pages. The whole crawler can be used to do focused crawling, geoparsing, indexing and weighting. The paper [8] describes other approaches for location based services – basically presenting SALT, "*an engine that receives web sites as input and equips them with location-tags*" [8]. SALT extracts location up to street level and allows for user feedback to refine results.

As for recommending data entries to users, the paper [14] presents an efficient ranking algorithm for recommender systems based on a random walk model on hypergraphs. The recommender algorithm is capable of capturing the difference between popular and niche objects. Furthermore, a recommendation list update algorithm is presented which significantly reduces computational complexity. The whole algorithm doesn't require any parameter tuning and thus gets easy to

implement. PageRank is described in [15] and is probably the most famous crawler/recommender algorithm. Based on the link structure web sites are ranked (the more often a web site is referenced, the higher its rank) which allows for a highly representative rank of importance of web sites. The paper [16] presents a content-based recommendation method that provides results based on a user's recent interest in a web site. The recommendation is done via keyword contexts which are a set of discriminating words that occur together often (those keywords could capture more semantic information than a single keyword). The retrieval is done via information retrieval techniques and association mining techniques.

There are several studies related to the **user interface**. We mainly focus on the studies about tag clouds and navigation tiles as those play a central role in the Local Browsing approach.

Popular approaches for an improved search experience are so called tag clouds. A tag cloud is a collection of words that are more or less coupled to a given topic. A possible way to collect such tags is word counting, where words are sorted according to their frequency on the web site. Often, the size of tags in the tag cloud tells a user how often a given keyword appears in a web site. Tag clouds have been used for desktop web sites for quite some time, even though mostly not for search but for categorization or providing a user with further interesting topics [17].

Recent applications transferred these tag clouds to mobile devices where they provide a way for the user to search for topics by refinement of keywords [11]. The author of the paper uses tag clouds as spatial location descriptors which associate pictures, web sites, posts and articles with keywords and generate tag clouds where more frequent keywords are displayed in bigger font. The tags are generated from unstructured textual web content by applying the term frequency-inverse document frequency algorithm: the more often a term appears in a document, but not in the overall collection, the more particularly descriptive for this document it is.

Other approaches to the search on mobile devices problem include systems where users are able to ask each other questions [18]. In this paper map based interfaces are compared with text based interfaces. For both of those interfaces users could view other queries by users, see or help improve their results and post their own queries. The conclusion was that a hybrid interface that allows users to seamlessly switch between maps and lists is accepted very well.

The FaThumb approach brings facet navigation (searching along tags and tag groups) to mobile devices [19]. FaThumb is a keypad-driven, compact query interface for browsing and searching large data sets. Facets are hierarchically sorted, within this hierarchy users select tags they like and get presented all the entries that contain all the tags.

The TapGlance paper [17] describes a new UI that defines a keyset that allows fast navigation within all applications. The UI is developed because users only have limited time to devote to learning any given application and would like it if all applications behaved in the same way. The importance of such a shared design guideline is also shown by the effort Smartphone OS manufacturers spend on clear and generally usable design guidelines.

Other studies compare different tag cloud layouts [20], integrate the search experience within the browser (in form of temporal and location information clues) or by extending the amount with which users can interact with the search engine (by letting the users show queries from other users, by letting users filter different queries, presenting them on a map, and so on) [21].

Existing Solutions

In this section, existing search engines on mobile devices are presented. They are compared using the framework developed in [5]. The following selection of search engines consists of the most popular search engines for mobile devices, as well as a selection of search engines that focus on local search. As we test the application for locations in Switzerland, we choose local.ch, as this is the most famous local search provider here.

Comparison Framework

As described in [5], the following framework can be used to classify search engines. Classification is done on datasets, user input and the corresponding output. Table 1: Framework for search engine classification describes the different possibilities.

Resources	• Data Collection (e.g. unsupervised, manual) • Semantic Content (e.g. country based, just business or train information) • Information Type (e.g. text, image, video, audio)	
Input	How is the query entered? • Keyword • Image • Audio	Other passively used information? • Position • Country • Language
Output	How are hits displayed? • List • Map • E-Mail • SMS	How are hits ordered? • PageRank • Distance • User Ranking
	Visible	Non-Visible

Table 1: Framework for search engine classification

Google Mobile Search

Google Mobile Search is probably the most widely used search engine on mobile devices nowadays. The user interface of the mobile web site is a slightly modified version of Google Desktop Search. It features additional buttons to directly search for restaurants, cafés and bars. These buttons are a simple redirection to Google Places which searches for objects nearby, using the user's location if enabled. The Google Search app features a text input box, as well as an audio input button. On both the web site as in the app, user input is usually done via the classical typed keyword input. This requires a lot of on-screen keyboard typing.

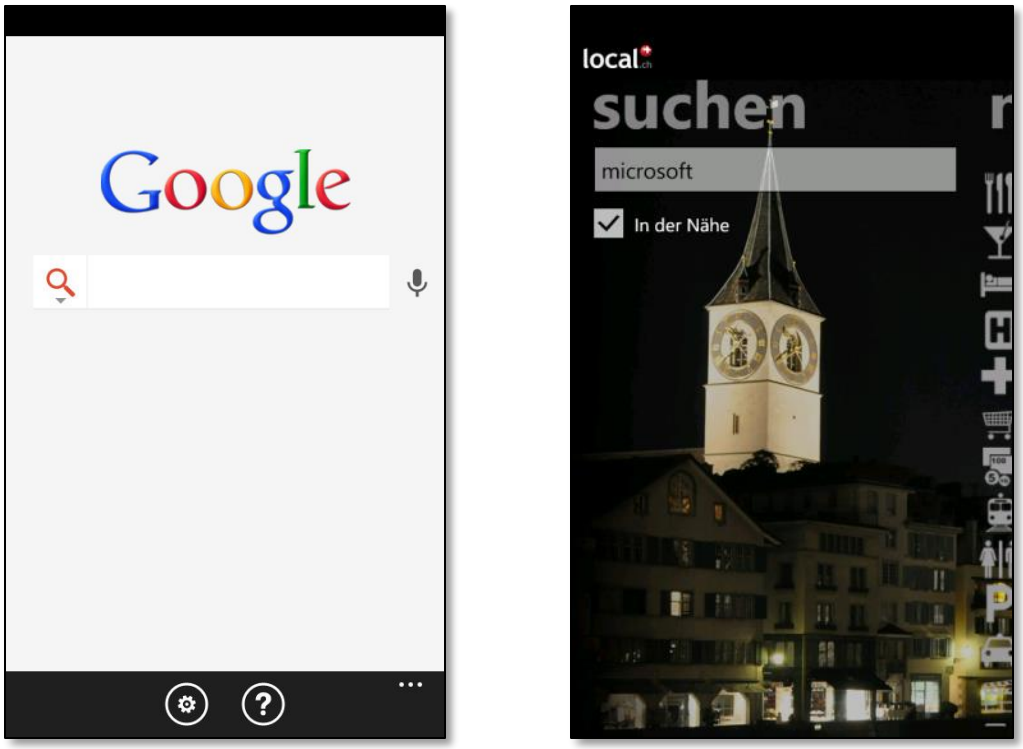

Figure 1: Google and local.ch apps on Windows Phone 7

Google uses a lot of different information types, unsupervised as well as manually entered data and semantic information to calculate results. The query is entered usually as keywords (image search and applications like Google Goggles allows searching by uploading pictures and the Google Search application by speech input) with other passive information like location. The output generated is either displayed as a list or on a map, ordered by PageRank and distance to user.

Local.ch

The Local.ch app features again a classical typed keyboard approach. It can be used to search for various things, all of which need to have a location though. To simply see what's around the user's location, a swipe to the right brings up a browsing screen. On this screen one can browse by categories and see the corresponding entries close to one's location.

In the above classification, Local.ch uses manually collected textual and location information (telephone book, yellow pages). The user enters text as query and the application returns a list of results.

Browser (Embedded Search Provider)

By entering keywords into the navigation bar, the browser takes you to the Bing search application. Using a typed keywords based approach, Bing allows you to search for apps, web content, local content and images. The categories can be chosen with a simple click or swipe and display various additional information like the location on a map.

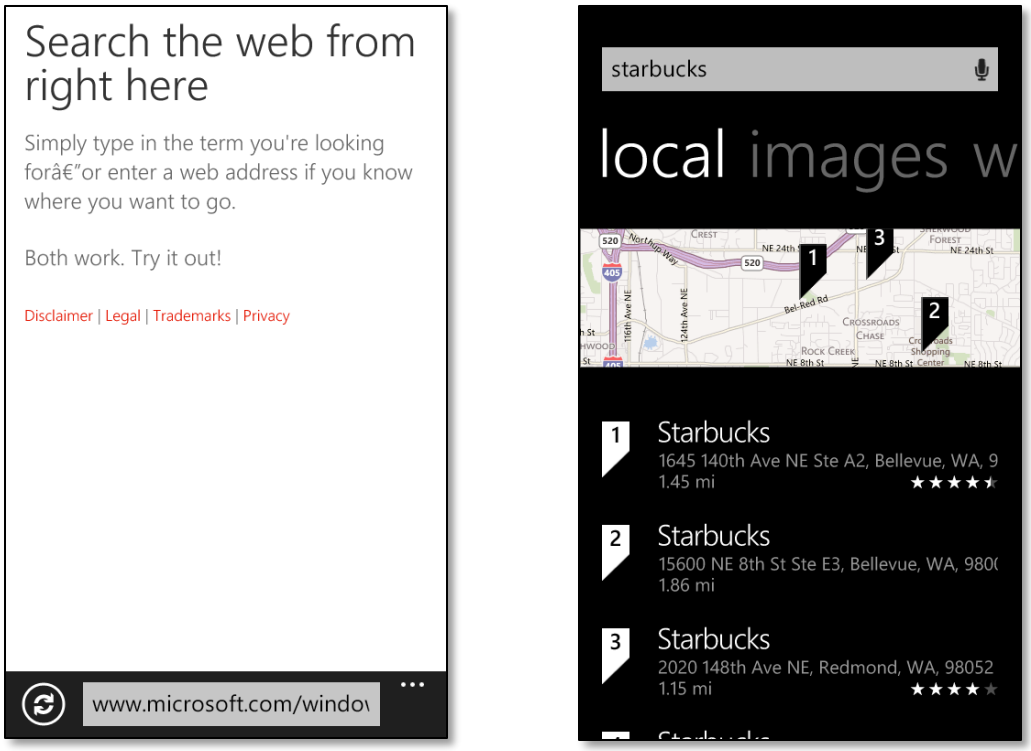

Figure 2 Embedded Bing search in Windows Phone 7, combined with Bing Maps

The integrated Bing Search uses unsupervised as well as manually collected data, presents results on lists or maps, ordered by PageRank and distance. User input is handled via text and audio, with additional information from location.

Nokia Maps

Nokia Maps is an application used for location based search. It does so by typed keywords and by browsing by category. The results from the Nokia Maps Database are shown on a map around the user's location. Clicks on entries show additional information, like web site, telephone number and more.

Nokia Maps uses manually entered information and semantic data. The data is textual and image data. User input consists of keywords, the output is presented on a map, sorted by distance.

Figure 3 Nokia Maps

The Local Browsing Solution

The following section describes the Local Browsing solution in detail. In a first part, the user interface frontend is described, in a second part the server backend. The project consists of two theses, where one (for one person) was the development of the user frontend and the other (for two persons) was the development of the backend server.

System overview

The whole system can be divided into two main parts: The server backend which handles all kind of calculations and storing of the necessary data and on the other hand there is the frontend, an application on a phone which provides a user interface to interact with the calculated data from the server and also collects new data as shown in Figure 4.

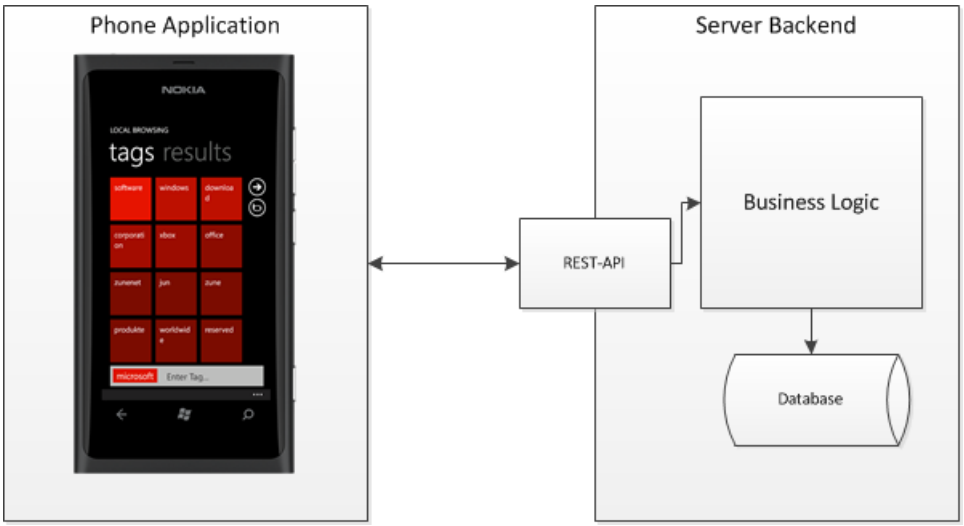

Figure 4: System overview with application and server

The system is designed for the purpose that all heavy computation is done by the backend server. This ensures that the frontend application runs as smooth as possible with its limited resources.

User Interface Frontend

The following section describes the Local Browsing Windows Phone Application. It tries to fulfill the requirements in the problem definition by providing functionality to search via tags, to switch between query inputs and results in a fast way, to give feedback on results and to view web sites.

Schematic Overview

A first observation to make is that you need user generated data to provide the requested features of the problem definition section. Solely collecting web sites is not going to tell when or at which places users like to search for which things. Also, to make the whole system scalable, automated approaches have to be used to ensure users always get the most popular results for any situation.

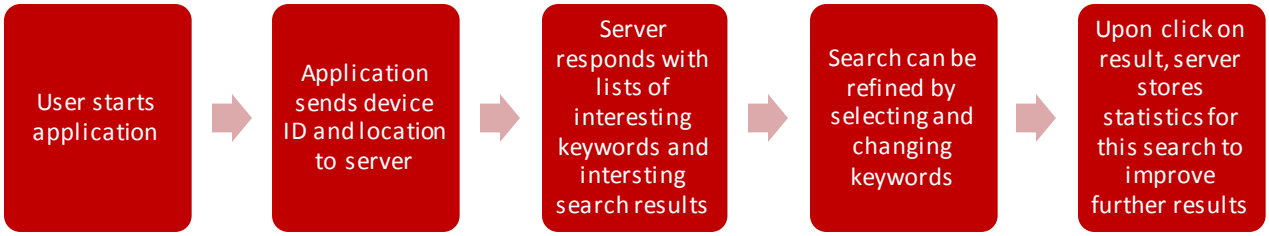

Figure 5: Search process by using Local Browsing

With this in mind and based on [9] the following application flow was designed:

- After the application starts, a device ID is sent to the server. This allows the users to stay anonymously but still giving him personalized results. Of course, in a later step complete user authentication should be considered, in the way many big search providers are doing it.
- The server responds with a list of keywords that could be of interest to the user. They are aggregated by the local browsing recommender which is described later. This recommender recommends keywords and websites because of several different aspects:
    - They are physically close to the user, i.e. their location is within a certain radius to the user.
    - The user used to access this keyword / website before. It is especially valuable, if the user already accessed it in this same location before.
    - A lot of users tend to access this keyword / website. The result is especially valuable if a lot of users access the website at the same location the user is at the moment.
    - The user could be interested in the keyword / website in his current situation.
- The keywords are then displayed in the application. Keywords are represented by buttons, whereas a click on a button has several effects:
    - The keyword is added to the list of currently selected keywords.
    - A new set of keywords is requested and displayed.
    - A set of results is requested and held ready for the user to see.
- An additional input field allows the user to add keywords that can't be found in the list, also to remove them.
- The user can switch between the keywords view and results view at any time to improve his search results.

Figure 6 gives a schematic layout of the main part of the application. On a first page keywords are shown. They can be selected and refined by entering in the refinement box or deleting from there. The keyword boxes might contain additional information such as hints to future keywords, results or number of results. On a second page the results themselves are displayed. The result boxes display different information about the results like title, description, distance to current location and more. Switching between the pages is implemented by the swipe gesture, but other ways might be acceptable as well (e.g. pressing a button). The only requirement for the switching is that it is fast and intuitive.

The following section describes the concrete implementation in detail.

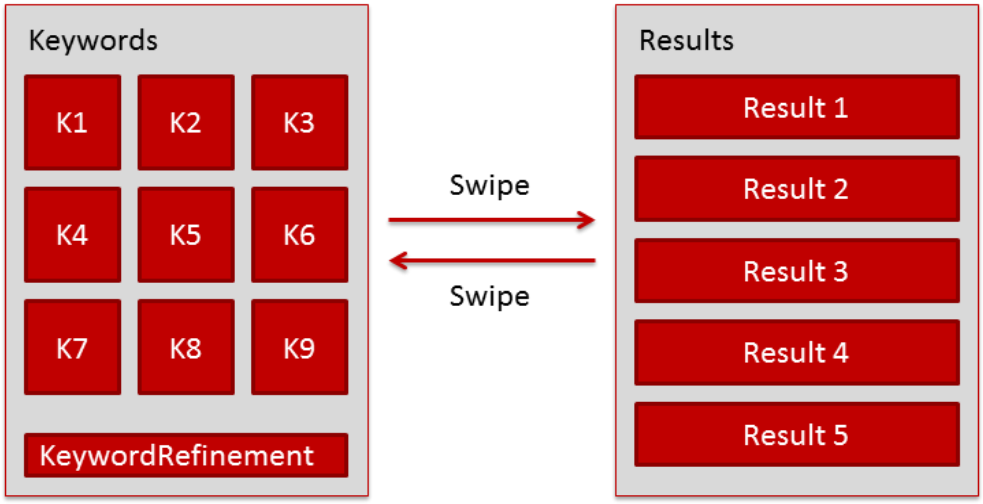

Figure 6: Application UI concept

The Implementation

The application itself features tiles (Windows Metro style buttons) that each display a keyword. The color of the tile is determined by the importance of the keyword. After testing several different layouts we decided upon a layout with 4x3 buttons that shows only the keyword itself. This is integrating well in the Windows Metro design guidelines, gives the keywords enough space to be displayed and the buttons are big enough to be easily accessible. As these tiles are rather big and each of them stands for a whole word, navigation is very fast. On startup a screen like Figure 7 (left side) is shown. Within this screen, the keyword set can be adjusted as liked. Keywords can be added and removed with a single click, and if for any reason no good keyword can be found, a keyword can be entered manually as well. For simplicity the term keyword was replaced with tag in the application. An arrow button gives a different way of switching to the results page and a Bing button allows searching Bing for the currently selected keywords.

With a single swipe, the results for the current keyword set can be viewed. This allows for very fast checking for results and leads to a continuous refinement process. The result screen itself (Figure 7 right side) presents the different results with the most important details (title/domain, description and distance to current location if available) and sorts the results according to their importance. It also allows viewing the result on a map, requesting more details and under certain circumstances editing the result (mainly for improving results by giving users the possibility to give their own input). Those tasks are accessed by a long click which opens a context menu on Windows Phone.

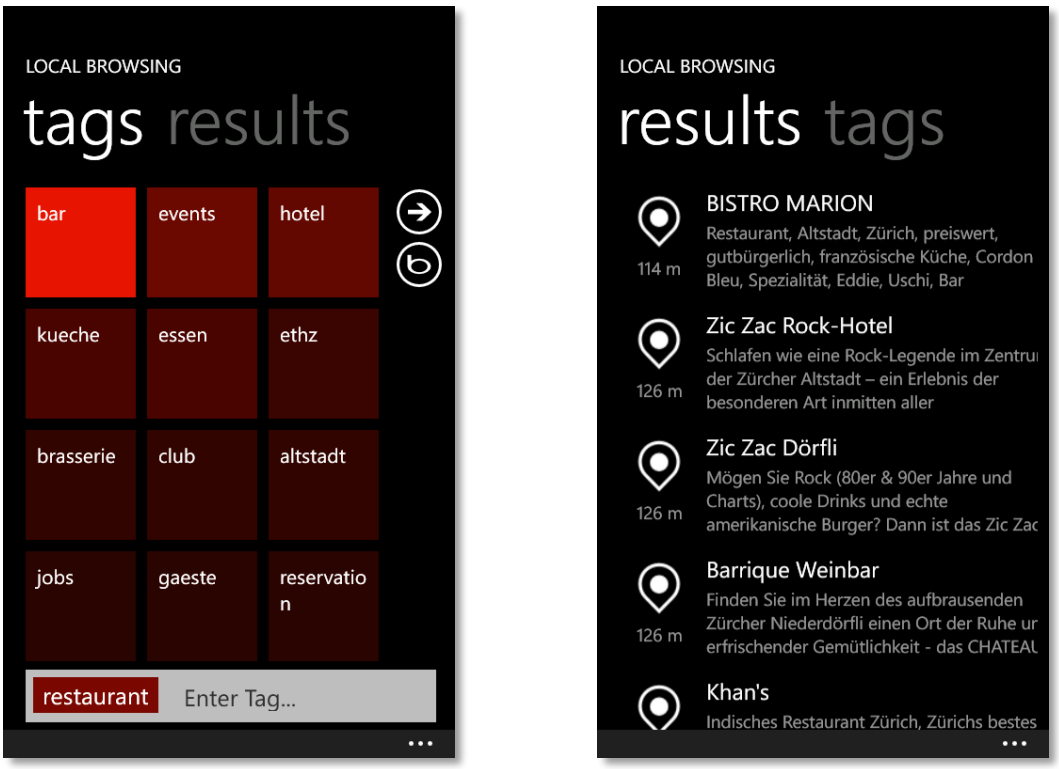

Figure 7: Final Local Browsing UI on Windows Phone 7, with tag selection (left) and result presentation (right)

Finally, when the user opens a web site, it opens in an internal web browser. This internal web browser shown in Figure 8 allows the user to give feedback. In addition it also allows the application to track certain user actions. Both actions are necessary to improve search results further (see also problem definition section). A click on the Internet Explorer button will open the page in the mobile browser of the phone, where further data collection is disabled and special actions are available (like connecting to a https site or displaying flash applications). The thumbs buttons are used to rate the results, the back button will directly jump back to the results screen (the normal back button will navigate through the browse history). In addition it is also possible to enter arbitrary URLs and browse there.

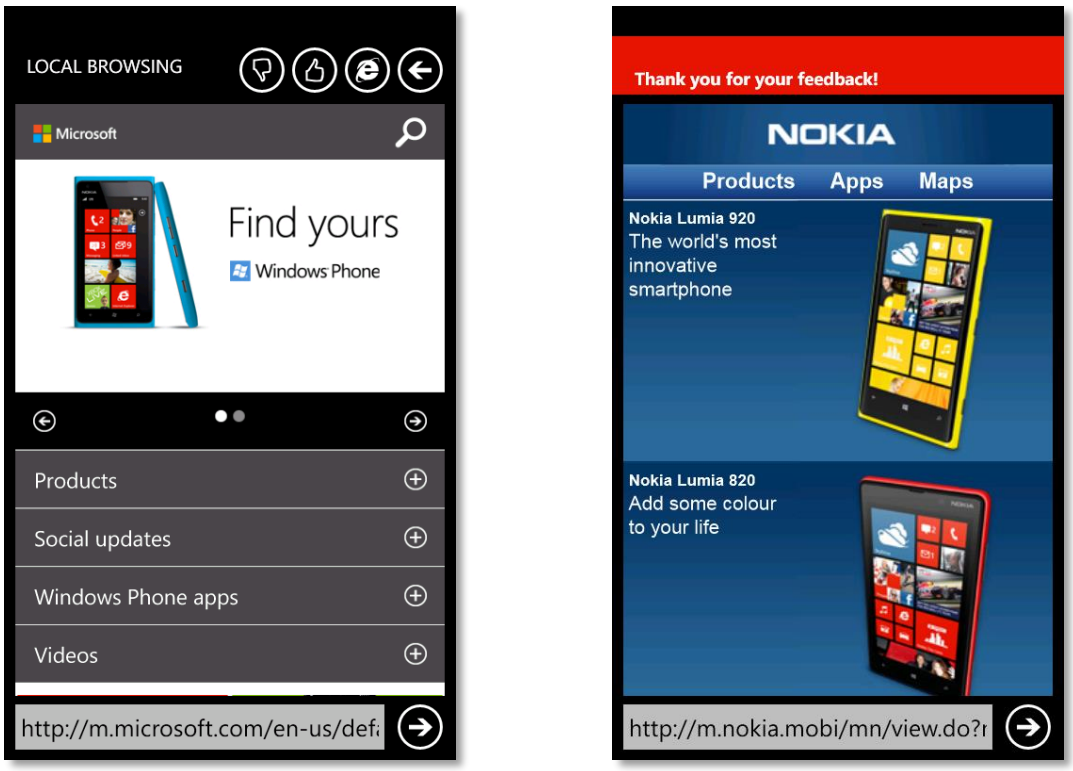

Figure 8: Embedded Internet Explorer in Local Browsing with feedback toolbar on top

Further application parts are not specific to a search application and can be found in many other applications. For completeness a screenshot of the application info page is provided in Figure 9.

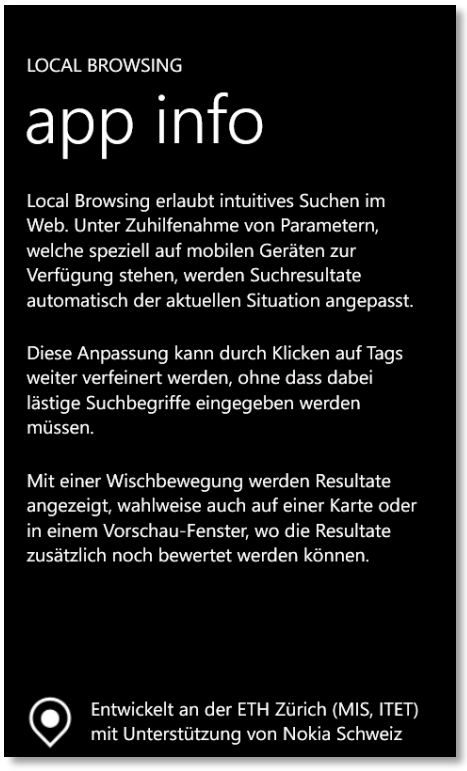

Figure 9: Application info page

Server Backend

The following section describes the Local Browsing Server Backend where all the complex computation and data storage is located.

Smart phones in general have very limited resources compared to a computer. The idea behind the design of the server backend is to take all necessary input from the phone (frontend), do all the calculations which require a considerable processing power and send a well formatted result back to the phone.

Server Backend Overview

The server backend can roughly divided into three different code sections.

- The Model domain which contains all kind of objects and generally accessible controllers, for example the logger. In the server project this is also called *Model* as shown in Figure 10.
- The View which is the interface between the frontend and the backend described in the *API / Communication Interface* section below. As shown in Figure 10 this part is called *Backend Web Interface* which is connected through the internet to the phone.
- The Controller which contains the most important controllers and managers for all kind of computation, for example the recommenders. In the server project this is also called *Controller* as shown in Figure 10 and has additionally a *Database Access* structure which is explained in the next text section.

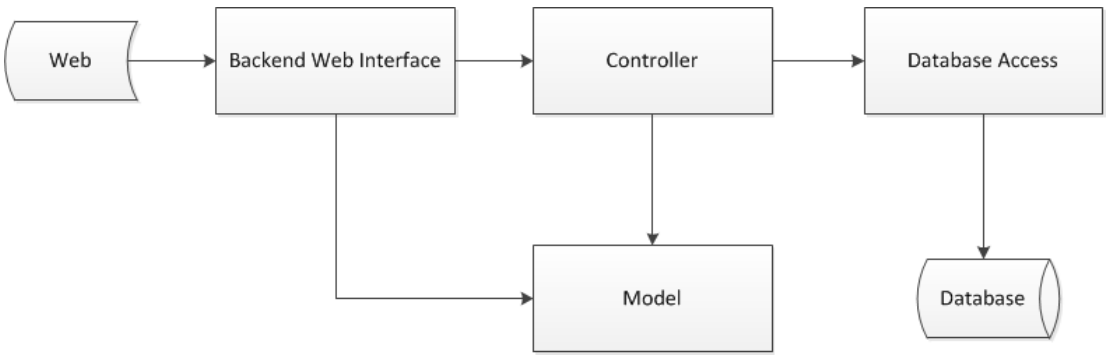

Figure 10: Server backend overview

The *Backend Web Interface* is the interface between the web (the phone application) and the server, and is well defined over a known API. It also ensures that no invalid requests from the outside reach the next layer. The *Database Access* serves as an additional layer between the controller and a chosen database. Every access to the database is abstracted in a way that the choice of a database type is independent from the controllers. For more information see The Database Interface.

To test the functionality of the business logic within the above mentioned code sections and all kinds of computation or data fetching from external databases, there exists an additional code section called *Unit Tests.* These tests are based on the *jUnit*[1] test framework for Java. They can control

[1] http://www.junit.org. August 2012

classes from all other sections and are used to ensure the correct behavior of most of the computations.

MVC - Model View Controller

The used code structure is based on the *Model-View-Controller* design pattern, which decouples the business logic from the user interface and the model domain. For this purpose three different domains were created shown in Figure 11: Design pattern MVC:

- Model domain is used for the program states and also provide a data structure.
- Controller contains the business logic and interactions with internal storage.
- View is mainly used for processing data for visual representation or forwarding them to another output.

This design makes it possible to change most of the logic computation from the controllers without the need of modifying the web interface or on the other hand changing the web interface and reusing the same computations.

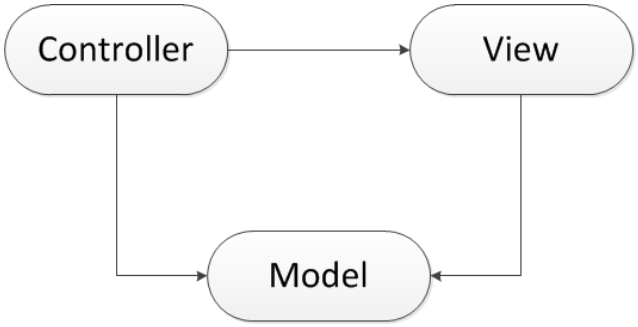

Figure 11: Design pattern MVC[2]

Adding new abstract structures in the model domain is also fairly easy, because of its independence from the other two domains.

Overview of the most important functions

The Local Browsing backend system consists of the following parts:

Crawlers	There are specialized crawlers that crawl open platforms for location tagged web sites and objects. The main web site crawler crawls web sites and extracts link structure, keyword structure, general web site information and meta information and creates a linked structure which can be used to generate results to user queries.
Recommenders	There is a keyword recommender that aggregates keywords based on selected keywords, user ID, location, time, popularity of keywords and other parameters. To generate results there is the results recommender that aggregates objects (e.g. websites) based on the user input and history data.
Application API	The application is accessibly publically via the Local Browsing API. This allows various services to take advantage of the system.
Logging System	The logging system is primarily for debugging and optimization of the system as well as some sort of passive watchdog function (in case of system failures).

[2] msdn.microsoft.com/en-us/library/ff649643.aspx. August 2012

Job System	The job system allows executing various jobs (e.g. crawl jobs) independently from the main functions of the system. Jobs can be queued, executed periodically and stopped in case of failure.
Database Interface	The database interface is independent of the database and can thus be used with various database systems in the background (at the time of writing this thesis we use MongoDB[3]).

Table 2: Backend components

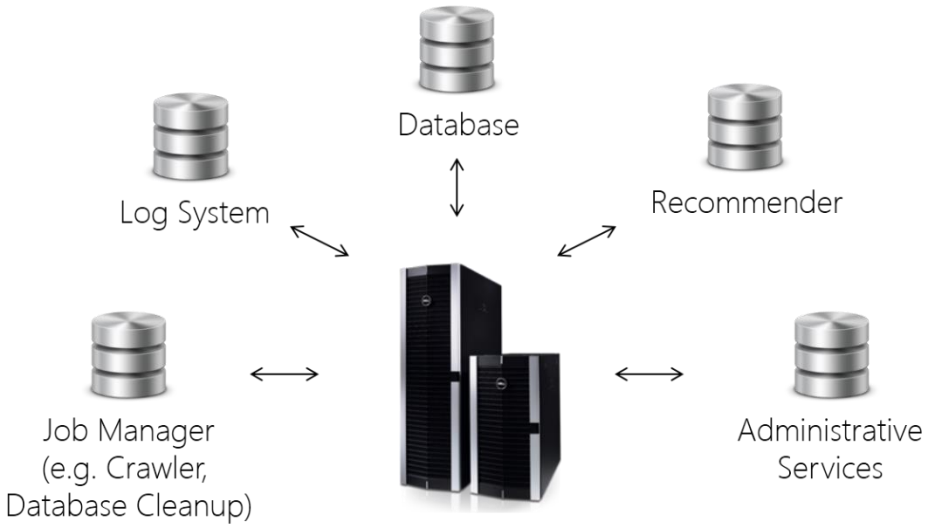

Figure 12: Backend

The whole specification of the system can be found in the documents "LocalBrowsing Application Design" and "LocalBrowsing Reommender Specification". The API documentation can be found in "LocalBrowsing REST-API Reference". The following sections give a more detailed insight of this important functions.

The Crawler

The main web site crawler downloads HTML pages and processes their content. The crawler stores information in four different collections:

- The **Web Site Collection** is the collection search queries are run against. This collection contains all necessary information to find and weight results.
- The **Domain Statistics Collection** collects various information about the number of crawled web sites and their domains.
- The **URL Collection** stores extensive information about web sites. This information is required to update ranking information on web sites but is not needed for direct queries.
- The **Word Collection** stores information about words and their distribution in text on websites. This information can be used by the keyword recommender to recommend words that are similar, potentially connected or otherwise interesting to the user.

[3] www.mongodb.org. August 2012

The *Web Site Collection* is the most important collection, as search queries are run against it. The other three are crawler-specific. The collections are described in more detail in the following sections.

The Domain Statistics Collection

The domain statistics collection captures information about the crawled web sites and the domains they belong to. For each domain (.ch, .com, .net, …) the number of crawled, the number of uncrawled (links that were collected from web sites but are not crawled yet) and the number of not crawlable web sites (probably because they were offline, or forbidden for search engine robots) is stored. The total number of web sites is stored as well.

The URL Collection

The URL collection holds the following fields for each URL (indexes and internal fields are omitted):

URL	The unique URL where the further information was extracted.
URL Type	The type of the URL is either PARENT or CHILD. A parent site is at the top level of a domain (e.g. *www.google.com*).
Top Domain	The top domain (e.g. *com*).
Referenced By	A list that contains all other URLs that reference this URL. The list also contains a number for each entry that tells how many times the web site was referenced.
Total References	The total number of times this web site was referenced.
Crawl Date	The date when this URL was crawled the last time.
Import State	The import state, this can be UNCRAWLED, CRAWLED, NOT_CRAWLABLE, ROBOTS_FORBIDDEN and more.
Crawl Time	The time it took to crawl and process this URL.
Language	The language as found in the meta information. This language will pass on to words extracted from this web site.
Number of Sentences	A number counting the number of sentences found on this web site.
Number of Words	A number telling how many words were found on this web site.
Meta Information	The complete Meta information from this URL.
Keywords	All the keywords from this web site and the number of their respective occurrences.
References	The links this web site references to. This is a list containing all the links and the number of times they were referenced.

Table 3: Database URL collection

The Word Collection

The word collection stores information about words found in documents, their relations with each other, their occurrences and so on. This information can be used by the keyword recommender. The collection holds the following fields:

Word	The name of the word.
Number of References	The number of times this word occurred overall the documents.
Languages	The languages associated with this word. The language is taken from Meta information of web sites.
Word Window 3	Words that were within the previous or next three words. This can be used to find words that usually turn up together.

Table 4: Database word collection

The Web Site Collection

Within the web site collection, entries have the following important fields:

Web site Meta information	Meta information is extracted directly from Meta tags. This includes general Meta information and also specialized Meta information like language which is further used to classify keywords.
Keywords	Keywords are extracted from Meta information and from web site content. The keywords are weighted according to the number of occurrences and stored in a sorted list. This is a simple word frequency approach.
Words	In addition to keywords, word structures are extracted and stored in their separate table. This includes word relationships generated from windowing, as well as overall word frequency and language. Windowing is done by storing all words that appear within a window of three words around the current word. Those words are counted and stored alongside the original word.
Link Structure	References to other websites are stored in an URL collection, as well as total number of references of an URL.
General URL Information	Various side information is stored for every web site. This includes things as number of words on the pages, number of sentences, language, crawl status, crawl time, and more. Using this information, web sites themselves are ranked.

Table 5: Database web site collection

How the Crawler Works

The crawler is implemented as recurring job in the backend. Whenever new sites are added to the web site collection, they will get crawled on the next crawler pass. The crawler will then follow the link of the web site added, collect all information on the web site and store it in the database. For this the crawler uses the HTML processing library *jsoup*[4]. Usually, links to other web sites are stored, but not followed. This is one option amongst others that can be specified when using the crawler though.

The crawler can of course be run multiple times in parallel or on demand if the amount of newly added web sites becomes too large for a single crawler to process.

[4] www.jsoup.org. August 2012

The Recommender

The Local Browsing result recommender is built upon the requirements coming from [9]. This recommender aggregates objects from four different sources:

- Location-based: Objects that are physically close to the location of the user.
- Situation-based: Objects that are interesting because of the situation the user is in. (not implemented yet)
- User-history-based: Objects the user accessed before.
- General-history-based: Objects a majority of users accessed before.

The result recommender takes information about web sites and objects from the web site collection, which is described in the following section.

History Aggregation

For the first recommendation the history is one of the most important source for a good result. The History is only calculated once per session and is cached as a hash table for every further request from the phone. Only a specific threshold between the old and the new location of the user will trigger a new calculation.

All data comes from the history database collection which is described in later chapter and is collected with four different queries. Two of them search for history entries near the location of the user, all already browsed websites around this location. First, all history entries without taken account of the current user ID are collected and secondly only the browsed web sites from the current user are taken. For all queries the resulting list of web sites is sorted by the number of views in a defined period of time which gives every web site a weight. Both queries are merged together by adding those weights multiplied with two different factors (represented as F in Figure 13). The merged result of these two queries is then merged with the combined result of the global history (without taken account of the location).

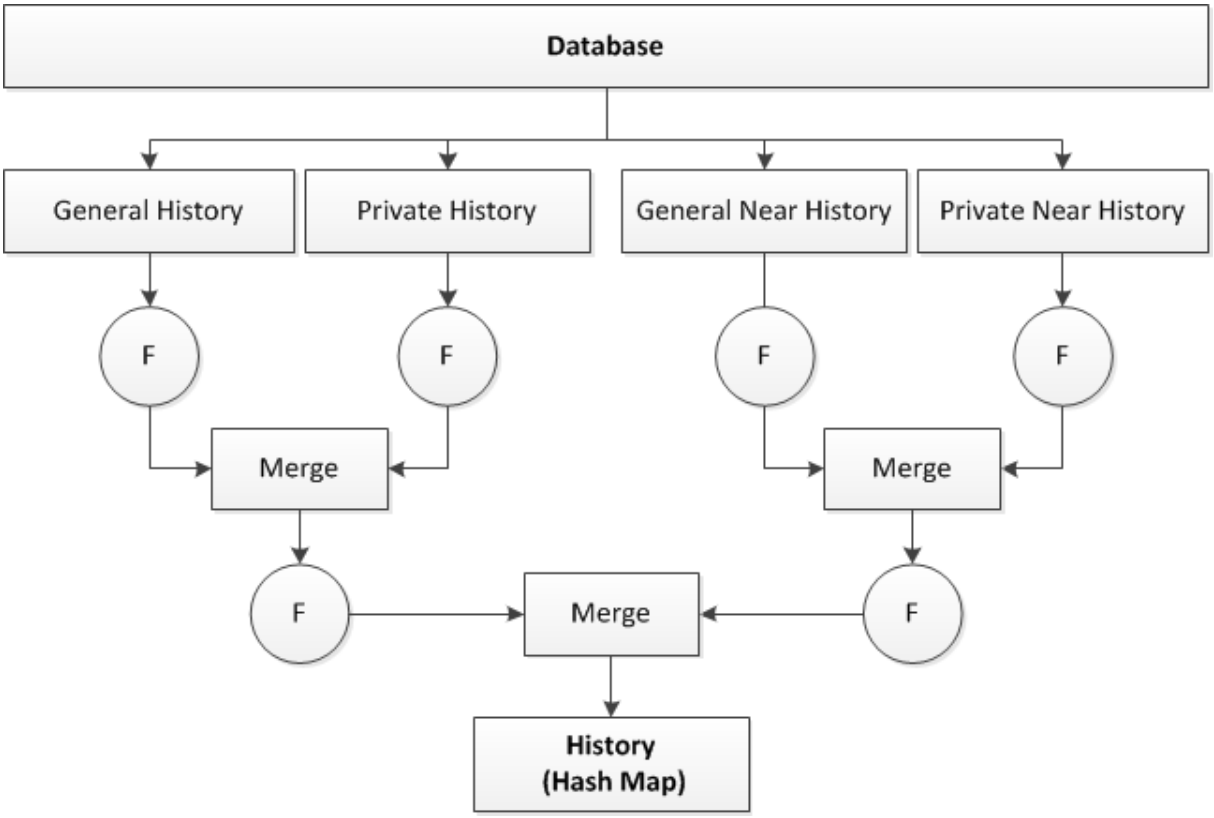

Figure 13: History aggregation

Results Aggregation

After the calculation of the history the recommender is ready to aggregate web sites for the first recommendation. Without user input, meaning no tag was selected or entered manually, the web site database collection can only be queried by location. The cached history is merged with the list of web sites resulted from the location-based query.

Available user input changes the type of querying the website collection. Like shown in Figure 14, we got two database requests after this tag, one of them additionally with the current location of the user. The results of the location query are sorted by a weight which is calculated by the distance between object and users location. However, the weight of the query based only on users input is calculated by the importance of this tag for this object. Detailed information on tag/keyword database search functions is given later.

Similar to the history calculation this weighted results are merged together with corresponding factors given through changeable recommender settings. In the final step the results are additionally weighted by the cached history. This means that if a website is popular at the moment it can be found in a top position in the resulting list of websites.

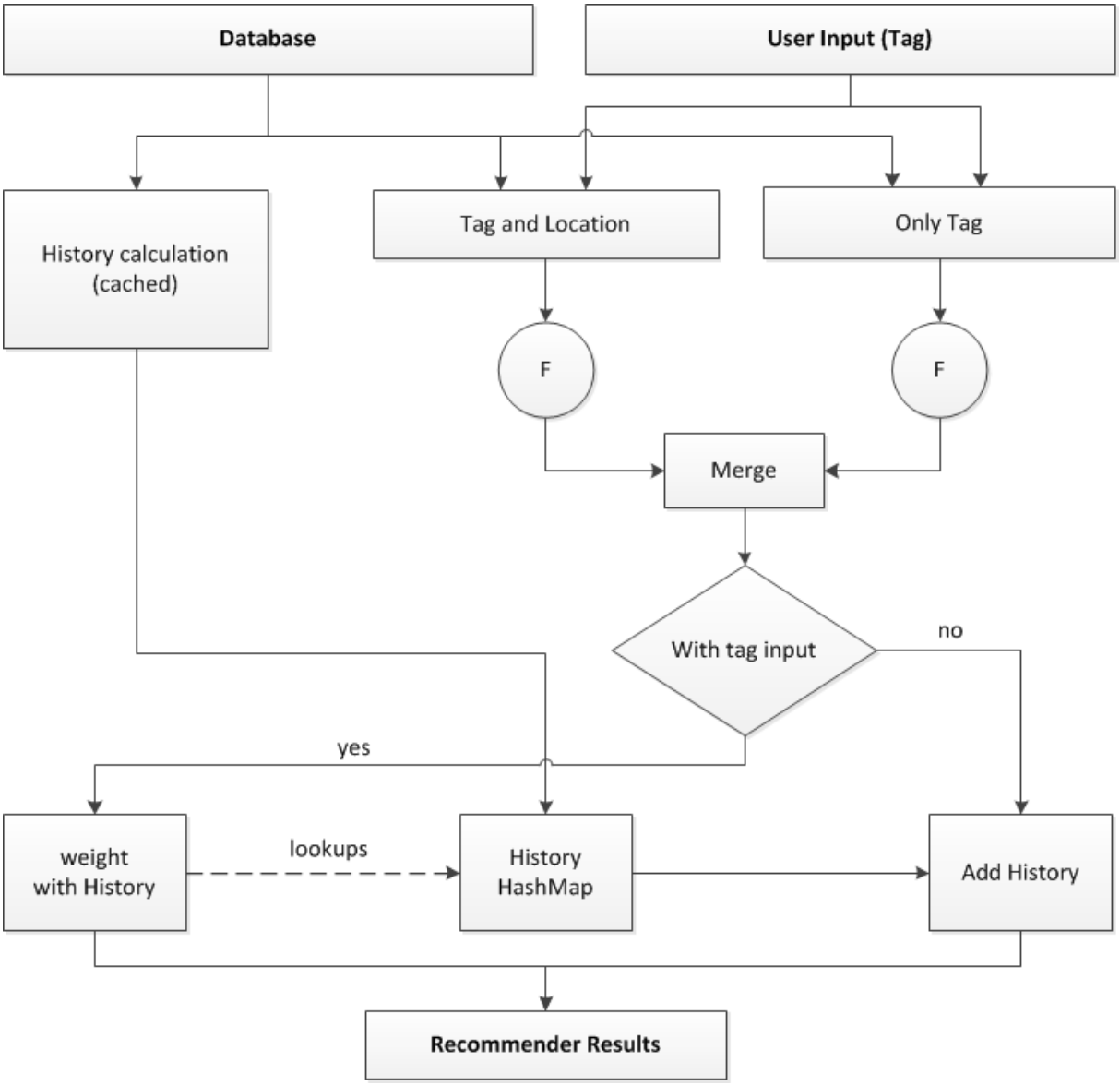

Figure 14: Website Recommender dataflow

Keyword Aggregation

Keywords which are recommended to a user are aggregated by the keyword recommender. The most important keyword selection is the first one, because this decides if a user can directly select one which reduces the number of user interaction. If no presented tag is near the topic the user is searching for, he needs to enter a keyword manually which must be avoided.

A good first tag representation is again a mix out of different sources. First a selection of all keywords the user was searching for in the past is computed, and also the same for all users. Second the database lookup additionally depends on the current location. So we want to know what this user or all users have recently searched at this location. Like shown in Figure 15 all this four database results are merged together the same way explained in history aggregation.

This final keyword history is then combined with keywords belonging to recommended web sites. Without user input, which is the case at startup of the application, most tags comes from tag history or near websites. As soon as we got user input the merged tag history is only used to adjust the weight of keywords coming from website recommender. This leads to a selection process as the user

always gets presented a set of tags that reduce the current set of web sites. Of course, as a keyword is selected, the whole results recommendation process is started again which results in a completely new set of keywords again.

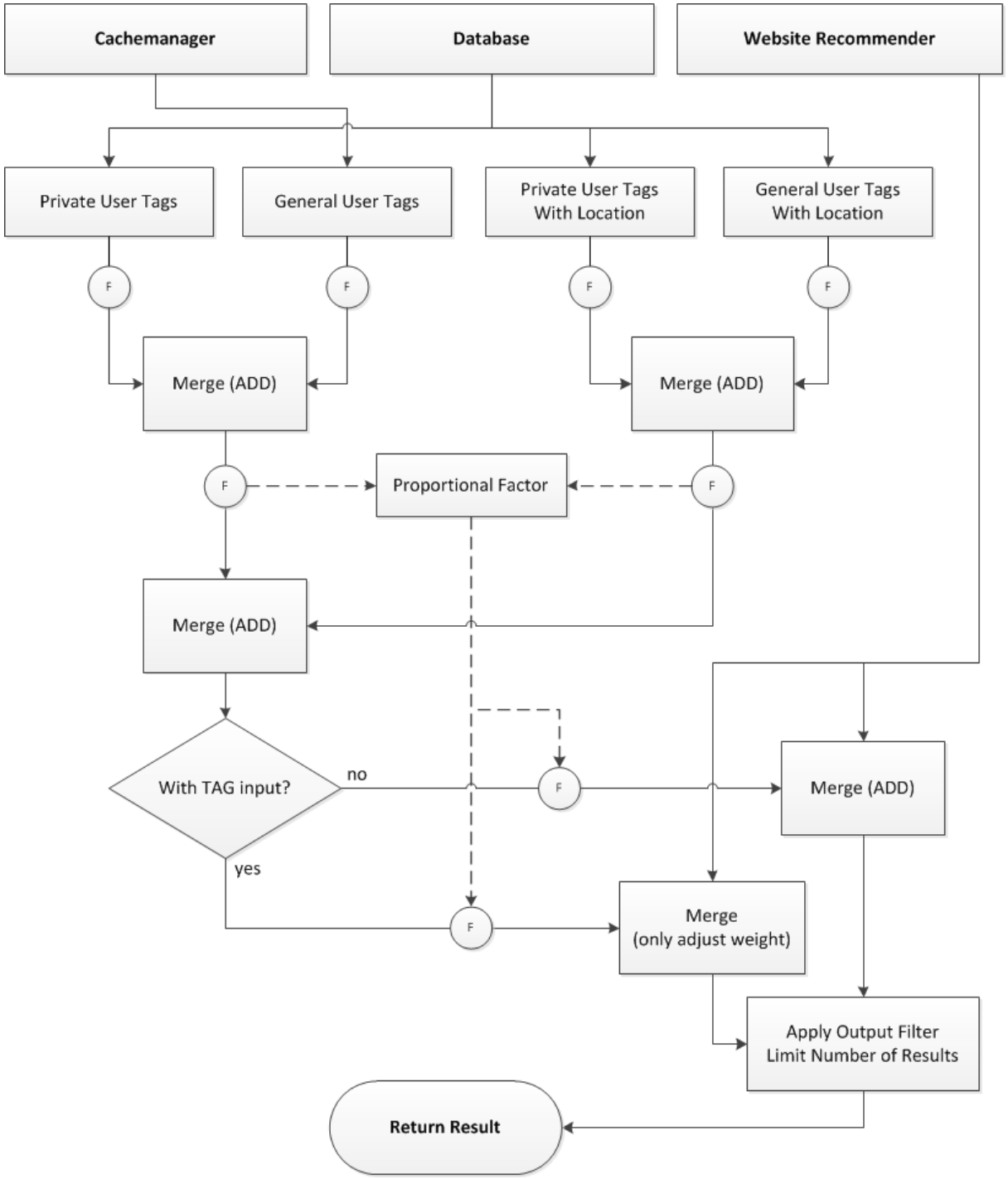

Figure 15: Keyword Recommender dataflow

The API / Communication Interface

The application programming interface provides a unified and secure way to access the Local Browsing system. The API is described in a separate document and can be used by independent developers to develop applications that make use or further extend the Local Browsing System.

The main aspect of the communication interface is the communication between the server and the Local Browsing application but it also provides support for other application, for example *WebNear.me*. It is based on a HTTP session which is created the first time an external communication device connects to the server. The session is managed by the Apache Tomcat service [5] and includes several servlets. The three most used of them are:

- *StartSessionServlet* which is first called to create a user.
- *UpdateServlet* to update any states e.g. the location.
- *ResultServlet* to get the new calculated results from the recommenders.

The communication protocol

A standard procedure for a new created communication session from the LB application to the server looks as follows:

1. The application on the phone connects to the server and starts a new HTTP session.
2. The name of the user and his/her location (if available) is sent.
3. Regular requests or updates.

After saving the user identity in step 2 a new object is created which consists of all useful results for this session, for example precalculated recommender results. For more information about the detailed communication protocol see the Local Browsing REST-API Reference.

The Logging System

The logging system is the main means of debugging on the server. Depending on the level selected, various outputs are omitted that give insight about the system. These outputs include:

- User information: Anonymized information about users accessing the system or trying to do so.
- Database information: Information about database queries, results, response times and more.
- Error and warnings log: System failures and warnings, e.g. when someone tries to abuse the system.

For this the well know logging Java library *log4j*[6] is used and embedded in the server code. The output of all log entries are forwarded to:

- A separated log-database.
- A file, which is used if the database fails somehow. Only a limited amount of critical log entries are saved in this file.
- The standard java output console, used for debugging on a local machine.

[5] tomcat.apache.org. August 2012
[6] logging.apache.org/log4j/2.x. August 2012

The Job System

The job system offers a flexible way of running different programs on the system. Jobs include but are not limited to:

- Crawl jobs: Jobs that crawl the internet and store information retrieved in the database.
- Cache jobs: Jobs that take data from the crawler databases and put them in a new form which is easier and faster to run queries against.
- Clean up jobs: Jobs that clean up the database by removing entries that proved to be of no value.
- One time jobs stored in a configurable queue.

The job manager is the main controller and holds access to all existing jobs. This manager uses the *quartz job scheduler*[7] which is an easy to use open source job scheduling service library for Java.

The Database Interface

The database interface was developed to make the underlying database transparent. At the moment the open source NoSQL database *MongoDB*[8] is used on the server, but by using the interface, which acts as an additional abstract layer, the underlying database could be replaced easily given the interface functions are implemented correctly.

Data types

The most basic object used for most calculations and results is the *LBObject*. It defines the fundamental properties which every result object must have. All more specific objects are children of this object for example a *Website*-object. The following table gives a summary of these important properties.

ID	The unique ID created and used by the database.
Source	Every crawler puts its identifier as source if this object is new created. Also other sources (e.g. user) exist.
Import State	This state defines if the object is fully imported or already deleted from search results (e.g. because of a broken URL).
Name	A meaningful, readable and short title for this object which can be presented to users.
Locations	A list of locations corresponding to this object.
Description	The description of an object, help users to identify the one they are searching for.
Keywords	All keywords are stored in a list with string and weight.
Meta Information	Detailed meta information (for example from http meta tags).

Table 6: Basic data type for all searchable objects

[7] www.quartz-scheduler.org. August 2012
[8] www.mongodb.org. August 2012

At the moment the most used child class of the above shown *LBObject* is the *Website*-object. The following table gives a short overview of the specific fields.

URL	The unique URL where the web site is hosted.
Domain	Domain name with top domain (e.g. *google.ch*).
Subdomain	Everything in front of domain (e.g. *www* or *docs*).
Title	HTTP title of this website (also stored in meta information).
Rating	Internal rating score.
Website Type	If we got multiple hits from the same domain, it is important to order them in a logic way, for example the website structure itself. So we have two types PARENT and CHILD websites.

Table 7: Specified data type for the website object

Functions

The most important class for accessing the database is the interface called *IDbAccess*. It defines all possible access functions to the database and all classes outside of this domain will only use this interface to communicate with the database. Most of the functions defined in this interface are made to gather information from the database and for storing information. Gathering information is often a complex process combining multiple tables, thus they are often cached within the program to minimize the loading times for often used information.

The following table shows an example of some important database access functions.

Function	Definition	Parameters	Returns
getWebsitesNearLocation	Gets a list of *Website* around a given *Location*	*Location*, Limit	List of websites weighted and sorted by the distance
getWebsitesWithKeywords	Gets a list of *Website* including the given *Keywords*	List of *Keywords*, Limit	List of websites weighted and sorted by importance of given keywords
getHistoryLatestPrivate	Gets *History* from user	UserID, Limit	List of websites weighted and sorted by number of visits

Table 8: Example functions of database interface

Database Structure

The database is mainly divided into three different collections:

- **Website collection**: All *Website* objects are stored in this collection.
- **History collection**: All History objects are stored in this collection. Because of the large number of history entries produced by the users, this collection grows really large. Most of the functions which gathering information in a complex way from this collection have to limit their search scope to remain efficient.
- **User collection**: This small collection stores all different user IDs and some user dependent properties e.g. familiar places.

MongoDB

There are different reasons for choosing *MongoDB*[9] in this case. First, it is a document-oriented storage which helps to indicate objects from the internet. Every URL is mapped in one database document with meta-information and keywords completed with statistics, page rank and many more. The second but more important reason is native handling of coordinates. Most documents are attached with one or more locations (stored as tuple of latitude and longitude) and *MongoDB* provides different functions to search for them. There are commands like *$near* which returns all documents near a given location and attach the calculated distance to them.

[9] www.mongodb.org. August 2012

Results

This section describes the results gained from comparing the application to different other search engines that can be used on mobile devices. The comparison is done by searching for different websites, counting the number of clicks required to get to the desired result. The tests are performed on a Windows Phone device. Text input clicks are counted separately, as those buttons usually are very small and text input is cumbersome. Testers are asked to give a rating on the perceived usefulness of the results.

As most of the search engines tested use some sort of user history to determine the importance of data sets for a given user, the test results vary for each user. Also, all results could of course be obtained on several different search paths, so the average shortest path (from all the test users) found is chosen for all the search engines.

The search queries come from different areas, mainly the four different areas described in the problem definition section. Those areas are of particular interest on a phone.

In the following the different test scenarios are presented in more detail.

Any Bar Close by (Starting at Zurich Main Station)

The intention of this test is to see how long it takes a user to find any interesting bar close to his location. This situation could arise in the evening when you want to find a nice place, check its web site first, to see if the bar fits your needs.

	Local Browsing	Google	Bing	Local.ch	Nokia Places
Number of Clicks	2	1	1	2	1
Number of Keyboard Inputs	0	0	3	3	4
Number of Results	20+	10+	20+	10+	15
Perceived Usefulness	10	10	7	8	7
Comments			Sparse data in Zurich	No view on map, only list	

Table 9: Comparison of search engines for searching any bar close by

Spaghetti Factory (at Location 47.37/8.54)

The user is searching for a particular restaurant to make a reservation. The test is conducted two times, one time the user is close to the desired restaurant, the other time he sits at home (far away).

Location Close to Restaurant

	Local Browsing	Google	Bing	Local.ch	Nokia Places
Number of Clicks	4	2	3	2	2
Number of Keyboard Inputs	0	4	4	17	3
Number of Results	15	20+	4+	7	4
Perceived Usefulness	8	7	4	5	5
Comments			Only shows global web site, no location	Only phone number and location	Doesn't show web site

Table 10: Comparison of search engines for searching a specific restaurant near by

At Home

	Local Browsing	Google	Bing	Local.ch	Nokia Places
Number of Clicks	5	2	3	2	2
Number of Keyboard Inputs	0	4	4	17	3
Number of Results	15	20+	4+	7	4
Perceived Usefulness	6	7	4	4	5
Comments			Only shows global web site		

Table 11: Comparison of search engines for searching a specific restaurant

NZZ (Accessed Before)

The user doesn't want to type in the website name of a particular website but instead searches for it (with the intention of this being faster than entering the URL). The website has been accessed before, probably many times.

	Local Browsing	Google	Bing	Local.ch	Nokia Places
Number of Clicks	1	2	2	2	2
Number of Keyboard Inputs	0	1	2	3	3
Number of Results	20+	20+	4+	16	2
Perceived Usefulness	10	10	9	8	1
Comments		Shows links to topics		Shows additional information	Only shows location of bureau

Table 12: Comparison of search engines for searching a specific news portal

Anything Interesting Close by

The user walks around bored and would like to be entertained, probably with something that gives him ideas what to do around his current location. The desired result is a website that has an entertaining value to the user.

	Local Browsing	Google	Bing	Local.ch	Nokia Places
Number of Clicks	1	-	-	1	-
Number of Keyboard Inputs	0	-	-	0	-
Number of Results	20+	-	-	20+	-
Perceived Usefulness	10	0	0	6	0
Comments		No browsing possibility	No browsing possibility	Shows close objects, only ordered by distance	No browsing possibility

Table 13: Comparison of search engines for searching anything interesting close by

Train Schedules at Zurich Main Station

The user just arrived at Zurich main station for the first time and would like to get the train schedules. The desired website would be sbb.ch, zvv.ch or anything similar.

	Local Browsing	Google	Bing	Local.ch	Nokia Places
Number of Clicks	2	2	2	2	2
Number of Keyboard Inputs	0	1	3	3	3
Number of Results	20+	20+	4+	20+	20
Perceived Usefulness	10	10	7	5	1
Comments		Shows direct links to topics		Shows travel agencies close by	Shows train stations around Zurich, no web sites

Table 14: Comparison of search engines for searching train schedules at Zurich Main Station

Telephone Number of Co-Worker

The scenario is that a user wants to find the telephone number of a co-worker in the same building. The co-workers name has 13 characters and phone number is published on corporate website.

	Local Browsing	Google	Bing	Local.ch	Nokia Places
Number of Clicks	1	2	2	-	-
Number of Keyboard Inputs	5	8	13	-	-
Number of Results	10	20+	4+	-	-
Perceived Usefulness	10	9	6	1	1
Comments				No web content, tel. nr. not publically registered	No web content

Table 15: Comparison of search engines for searching a phone number of co-worker in the same building

Conclusion

As is clearly visible from the results section, Local Browsing succeeds in reducing the amount of clicks needed to find results. The reduction ranges from only marginal to quite drastic, depending on the nature of the search. For the classical depth search used to solve a single very constrained problem the widespread typed keyword search approach is still suited, however, for a lot of searches the browsing / filtering approach leads to less user input needed. As these searches are of greater importance on mobile devices, the Local Browsing solution is an improvement on the currently available solutions.

The results where Local Browsing stands out in comparison to other solutions include objects that are close to the user's location, that are of general or personal interest or that are situation dependent. These object were already stated in the problem definition section as primarily interesting on mobile devices. In addition, Local Browsing changes user input from keyboard input to tile input which is a lot easier to do on a small device without a physical keyboard.

However, the results do not show some of the drawbacks of the Local Browsing solution, which have to be tackled in the future:

- Assigning locations to objects: A lot of content on the web doesn't have any location tags. This is not restricted to content that has no connection to any physical location, but also to objects that do not provide the location in any computer-readable form.
- Data set size: Indexing the web is a huge task. In the scope of this thesis we limited the indexing to a certain set of web sites that users accessed during usage of the system and a set of web sites specifically crawled for the thesis. The lack of data gives the system a hard stand against other systems that have indexed a lot more objects.
- Keywords: Keywords are recommended based on their occurrences together with other keywords on the same web site. Semantic connections and connections learned from user search queries are disregarded at the moment. Also, as the whole system is based around the recommendation of interesting keywords, there is almost no limit to evaluating different systems and approaches to recommend keywords. Whilst the Local Browsing approach in this thesis is simple and clean, there might be others that outperform the current system.

All summed up, search on mobile devices will change in the future. The Local Browsing system is a new combination of current search technologies and new approaches, making use of information specifically found on mobile devices. As Local Browsing only targets mobile devices, it was possible to design the application from scratch, giving room for innovative concepts. The resulting application is appealing, fast and fun, outperforms other search applications in certain areas and provides a considerable alternative to current applications.

Outlook

In a first step to improve Local Browsing itself, the above discussed issues have to be treated:

- Assigning locations to objects: Locations could be extracted from text, from linked web sites or from meta information. Also, locations could be assigned to objects counting where users access them the most. Currently getting implemented on top of Local Browsing is SALT [8], an engine that uses the text found on web sites to assign locations to them.
- Data set size: One approach to increase the available data is always setting up more computing power and storage space and let the crawler index the web. This is very resource intensive though, which makes other approaches attractive:
    - Using already existing search providers like Google or Bing and make calls on their APIs, fetching keywords from their results and use it to improve the Local Browsing solution. However, API calls are limited by amount and probably also by their further usage (by general terms and conditions).
    - Improving the crawler in a way that it searches only for web sites that have an easily identifiable location (as this is one of the main differences for mobile search). Also, one could crawl only at domain level (internal site navigation can be crawled if users show interest in site or is completely left to the user). This results from the fact that users of mobile search often only want the web site of a place they're visiting, but will search for more information on the web site themselves anyways.
    - Another possibility to limit the enormous quantity of data to be crawled is to concentrate on a niche in search results. Anyway, to compete against top dogs like Google providing a search interface for specified areas like news articles from local news providers or social media postings can be a good choice.
- Keywords: Local Browsing already now stores extensive information about word occurrences and relationships. This information can be used to extract word semantics, which further can be used to make better recommendations (of keywords and web sites). Other approaches would be to include publically available semantics databases or to extract semantics from user search queries (by analyzing which keywords turn up together often in user search queries).

Not discussed in the results section, but considerable extensions of Local Browsing include:

- Expanding the objects space to not only include web sites, but also objects like news, telephone numbers, public transportation schedules, and more. These objects can easily get a location assigned, whereupon they turn up if the user is close by and / or searches for the keywords. Even objects that have no location but are no web site can easily be integrated in the Local Browsing solution, as it allows to search / filter for basically any object.
- Update the user interface so that tiles already show peek previews of what's going to happen. This could be a preview of the new keywords, or the results, some images, or more.
- The user application could allow to directly interact with objects. As an example a tile or a result entry could contain a telephone number which would be dialed automatically upon clicking it.

- It should be tested if the results section is really necessary, or if the results could be included in the tags themselves, possibly by coloring them differently. This would make the constant switching and refining of tags obsolete.
- User guidance can be researched. As the user is mostly guided by the keywords the system recommends, this influence can be used to make the user get to certain results, which can be attractive for advertisement.

Local Browsing was designed in a way to make such extensions easily possible, whilst still providing all basic functionality.

Acknowledgements

We'd like to thank our supervisor Dr. Fabio Magagna for his assistance, for providing the paper the whole thesis is based upon and his useful ideas and insights. Also, to Professor Bernhard Plattner for taking an active part in the thesis by giving feedback and input throughout various meetings and for Professor Juliana Sutanto for making the thesis possible.

Our thanks also go to Nokia Switzerland, namely Simon Schweingruber for providing us with assistance and support concerning Windows Phone development and for providing us with developer licenses and devices.

Last but not least friends and family who tested the application, gave feedback and ideas and listened to hours and hours of babbling about search engines.

References

[1] K. Church, B. Smyth, P. Cotter and K. Bradley, "Mobile information access: A study of emerging search behavior on the mobile Internet," *ACM Transactions on the Web (TWEB),* 2007.

[2] M. Kamvar, M. Kellar, R. Patel and Y. Xu, "Computers and iphones and mobile phones, oh my!: a logs-based comparison of search users on different devices," in *Proceedings of the 18th international conference on World wide web*, Madrid, Spain, 2009.

[3] A. Arasu, J. Cho, H. Garcia-Molina, A. Paepcke and S. Raghavan, "Searching the Web," in *ACM Transactions on Internet Technology*, August 2001.

[4] K. Church, B. Smyth, K. Bradley and P. Cotter, "A large scale study of European mobile search behaviour," in *Proceedings of the 10th international conference on Human computer interaction with mobile devices and services*, 2008.

[5] F. Magagna, A. Gasimov and J. Sutanto, "Mobile Search Engine as a Business Model," in *International Conference on Electronic Commerce*, Honolulu, Hawaii, August 2010.

[6] A. Gasimov, F. Magagna and J. Sutanto, "CAMB: Context-Aware Mobile Browser," in *Proceedings of the 9th ACM SIGMOBILE Conference on Mobile and Ubiquitous Multimedia*, Limassol, Cyprus, December 2010.

[7] F. Magagna and J. Sutanto, "Reeco: A privacy-safe mobile context aware recommender system," *International Journal of Computer Engineering,* 2012.

[8] F. Maganga, B. Hess and J. Sutanto, "Building Location-Aware Web with SALT and Webnear.me," in *Procedia Computer Science*, Elsevier, August 2012.

[9] F. Magagna, *Why web search on mobile phones is web browsing OR A concept for a real mobile search engine,* Zurich, 2011.

[10] A. Zubiaga, "Content-based clustering for tag cloud visualization," in *Social Network Analysis and Mining, 2009. International Conference on Advances in Networks Analysis and Mining*, Madrid, Spain, July 2009.

[11] M. Baldauf and R. Simon, "Getting Context on the Go – Mobile Urban Exploration with Ambient Tag Clouds," in *Proceedings of the 6th Workshop on Geographic Information Retrieval*, Zurich, Switzerland, Februar 2010.

[12] G. Salton, Automatic text processing: the transformation, analysis, and retrieval of information by computer, Addison-Wesley Longman Publishing Co. , 1989.

[13] D. Ahlers and S. Boll, "Urban web crawling," in *Proceedings of the first international workshop on Location and the web*, Beijing, China, April 2008.

[14] B.-R. Blattner, "B-Rank: A top N Recommendation Algorithm," in *Proceedings of International Multi-Conference on Complexity, Informatics and Cybernetics*, August 2009.

[15] S. Brin and L. Page, "The Anatomy of a Large-Scale Hypertextual Web Search Engine," in *Seventh International World-Wide Web Conference*, Brisbane, Australia, April 1998.

[16] F.-H. Wang and S.-Y. Jian, "An Effective Content-based Recommendation Method for Web Browsing Based on Keyword Context Matching," *Journal of Informatics & Electronics,* pp. 49-59, November 2006.

[17] D. C. Robbins, B. Lee and R. Fernandez, "TapGlance: Designing a Unified Smartphone Interface," in *Designing Interactive Systems*, 2008.

[18] K. Church, J. Neumann, M. Cherubini and N. Oliver, "The "Map Trap"?: an evaluation of map versus text-based interfaces for location-based mobile search services," in *Proceedings of the 19th international conference on World wide web*, Raleigh, USA, 2010.

[19] A. Karlson, G. Robertson, D. Robbins, M. Czerwinski and G. Smith, "FaThumb: A facet-based interface for mobile search," in *Proceedings of CHI*, 2006.

[20] S. Lohmann, J. Ziegler and L. Tetzlaff, "Comparison of Tag Cloud Layouts: Task-Related Performance and Visual Exploration," in *Proceedings of the 12th IFIP TC 13 International Conference on Human-Computer Interaction: Part I*, 2009.

[21] K. Church and B. Smyth, "Who, what, where & when: a new approach to mobile search," in *Proceedings of the 13th international conference on Intelligent user interfaces*, Gran Canaria, Spain, 2008.

LB REST-API Reference

Version 0.2.x

The following topics offer a detailed reference of the REST API used for the *Local Browsing Backend* project running on a server. The version is depending on the API version. For more information see the changelog at the beginning.

Changelog

Version 0.2.x

- The Update Resource responds now with a list of new tags and their weights.
- The Results Resource responds now with a list of LbObjectMapping instead of a list of `LbObject`.
- The History Resource does not accept a `LbObjectId` anymore. Instead it requires a valid URL (`String`) in form of a HistoryObjectMapping.
- Added new Type Reference for `LbObjectMapping`.
- Added new resource references for Crawler Resource, Website Lookup Resource, Job Resource and Settings Resource.

Version 0.1.x

- The Update Resource does not require a parameter (all optional).
- The Results Resource accepts now a new parameter ("*locationonly*").
- The Data Upload Resource has been removed. Instead one can use the History Resource.
- Added new Type References for `HistoryObjectMapping`, `Location`, `LbObjectType`, `Feedback`, `UserAction` and `LogEntry`.
- Added HTTP Response Status Codes.
- General improvements of all articles.

Contents

The Start Session Resource

The start session resource is used to handle a new created session. This resource is called, if a user starts a new local browsing search and connects the session with the user.

Resource URI

/start?userid=...&[lat=...]&[long=...]&[acc=...]

Resource Parameters

Parameter	Type	Description
userid	String; required	The unique user ID for this session.
lat	Number (double); optional	The latitude of the updated location; Also requires a valid longitude.
long	Number (double); optional	The longitude of the updated location; Also requires a valid latitude.
acc	Number (positive double); optional	The accuracy of the location.

The GET Method

The GET method provides nothing; It only processes the request data and returns status codes.

Response Media Type

–

Expected Response Status Codes

On successful completion, the resource responds with status code 200 OK.
For a complete list of possible response status codes, see HTTP Response Status Codes.

Request Example

```
GET http://localhost:8080/api/start?
userid=C9WAFi6WXeKP9hCfIiEyVvJdYl4=&lat=47.378127&long=8.53981&acc=1.
5
```

Request Response Example

```
-
```

The Update Resource

The update resource is used to handle new inputs from a local browsing search session and returns new tags based on the update, which can be a new/removed tag or a new/updated location.

Requires a valid session created with the start session resource.

Resource URI

/update?[tag=...]&[removetag=...]&[lat=...]&[long=...]&[acc=...]

Resource Parameters

Parameter	Type	Description
tag	String; optional	A new tag for the current local browsing search session. There can be multiple tags in one request.
removetag	String; optional	A tag to be removed in the current local browsing search session. There can be multiple tags in one request.
lat	Number (double); optional	The latitude of the updated location; Also requires a valid longitude.
long	Number (double); optional	The longitude of the updated location; Also requires a valid latitude.
acc	Number (positive double); optional	The accuracy of the location.

The GET Method

The GET method provides a list of weighted new/updated tags based on the current state of selected tags and location.

Response Media Type

```
HashMap <String, Integer>
```

Expected Response Status Codes

On successful completion, the resource responds with status code 200 OK.
For a complete list of possible response status codes, see HTTP Response Status Codes.

Request Example

```
GET http://localhost:8080/api/update?
tag=SBB&tag=HB&removetag=hotel&lat=47.378127&long=8.53981&acc=1.5
```

Request Response Example

```
[
  {"mittagstisch":16},
  {"essen":23},{"news":23},
  {"mittagessen":24},
  {"events":19},
  {"windows":16},
  {"hotel":68},
  {"bar":19},
  {"restaurant":56},
  {"city":15}
]
```

The Results Resource

The results resource is used to return all calculated Local Browsing-objects (`LbObjectMapping`) from the current search session.

Requires a valid session created with the start session resource.

Resource URI

/results?[prettyprint]

Resource Parameters

Parameter	Type	Description
prettyprint	none; optional	If this tag is set, the answer will be pretty printed instead of a regular JSON String. Only used for debugging.

The GET Method

The GET method provides all calculated Local Browsing-objects (`LbObjectMapping`) in a list as JSON-Strings.

Response Media Type

`java.util.List<ch.ethz.mis.lb.server.web.mappings.LbObjectMapping>`
in short: `List<LbObjectMapping>`

Expected Response Status Codes

On successful completion, the resource responds with status code 200 OK.
For a complete list of possible response status codes, see HTTP Response Status Codes.

Request Example

```
GET http://localhost:8080/api/results?prettyprint
```

Request Response Example


```
[
  {
    "name": "ETH Zürich",
    "locations": [
      {
        "latitude": 47.376732,
        "longitude": 8.547943
      }
    ],
    "description": "lorem ipsum",
    "distance": 26.0,
    "weight": 0.9826300767314771,
    "lbObjectType": "WEBSITE",
    "lbObjectProperties": {
      "title": "ETH Zürich",
      "url": "http://www.ethz.ch"
    }
  },
  {
    "name": "SBB",
    "locations": [
      {
        "latitude": 47.377974,
        "longitude": 8.539895
      }
    ],
    "description": "lorem ipsum",
    "distance": 636.0,
    "weight": 0.3096588273457692,
    "lbObjectType": "WEBSITE",
    "lbObjectProperties": {
      "title": "SBB",
      "url": "http://www.sbb.ch"
    }
  }
]
```

The History Resource

The history resource is used to handle history updates from users which includes new Local Browsing-objects from various sources (e.g. the user, WebNearMe, ...)

Requires a valid session created with the start session resource.

Resource URI

/history

Resource Parameters

-

The POST Method

The POST method provides nothing; It only processes the request data and returns status codes.

Request Media Type

`ch.ethz.mis.lb.server.web.mappings.HistoryObjectMapping`
in short: `HistoryObjectMapping`

Response Media Type

-

Expected Response Status Codes

On successful completion, the resource responds with status code 200 OK.
For a complete list of possible response status codes, see HTTP Response Status Codes.

Request Example

```
POST http://localhost:8080/api/history
```

Request Body Examples

```
{
  "url": "http://www.ethz.ch",
  "lbObjectType": "WEBSITE",
  "dateTime": "2012-06-12T17:18:14+0200",
  "location": {
    "latitude": 8.54458,
    "longitude": 47.38191
  },
  "userAction": "FEEDBACK",
  "parameters": {
    "feedback": "POSITIVE",
    "past_url": "www.google.ch",
    "someOtherParameter": "someValue"
  }
}
```

Request Response Example

```
-
```

The Crawler Resource

The crawler resource is used to handle location crawler requests.

Resource URI

/crawl?[lat=...]&[long=...]&[tag=...]

Resource Parameters

Parameter	Type	Description
lat	Number (double); required	The latitude of the updated location; Also requires a valid longitude.
long	Number (double); required	The longitude of the updated location; Also requires a valid latitude.

The GET Method

The GET method provides nothing; It only processes the request data and returns status codes.

Response Media Type

–

Expected Response Status Codes

On successful completion, the resource responds with status code 200 OK.
For a complete list of possible response status codes, see HTTP Response Status Codes.

Request Example

```
GET http://localhost:8080/api/crawl?lat=47.378127&long=8.53981
```

Request Response Example

```
–
```

The Log Resource

The log resource is used to handle log requests and returns a specific part of the current log as JSON String.

This is an internal resource and only visible/accessible in a password secured private area.

Resource URI

/backend/log?[level=...]&[lines=...]&[skip=...]&[date=...]

Resource Parameters

Parameter	Type	Description
level	String; optional [default: “DEBUG”]	The minimum log level to be shown. It has to be one of the following: “TRACE”, “DEBUG”, “INFO”, “WARN”, “ERROR” and “FATAL”
lines	Number (positive integer); optional [default: 100]	The maximum number of log lines to be returned.
skip	Number (positive integer); optional [default: 0]	The number of log lines which are skipped.
date	String; optional [default: *current date*]	The date from which the previous number of log lines are returned.

The GET Method

The GET method provides a specific part of the current logs as JSON-string.

Response Media Type

see LogEntry

Expected Response Status Codes

On successful completion, the resource responds with status code 200 OK.
For a complete list of possible response status codes, see HTTP Response Status Codes.

Request Example

```
GET http://localhost:8080/api/backend/log?
level=info&lines=2&date=2012-06-12T17:18:14%2B0200
```

Request Response Example

```
[
  {
    "timestamp" : { "$date" : "2012-06-12T13:30:14.866Z"},
    "level" : "INFO",
    "message" : "Logger is initialized",
    "method" : "init",
    "lineNumber" : "62",
 "class" :
{ "fullyQualifiedClassName" : "ch.ethz.mis.lb.server.controller.log.I
nitializeLog4j"}
  },
  {
    "timestamp" : { "$date" : "2012-06-12T13:19:29.717Z"},
    "level" : "WARN",
    "message" : "No user id for this session found!",
    "method" : "doGet",
    "lineNumber" : "61",
 "class" :
{ "fullyQualifiedClassName" : "ch.ethz.mis.lb.server.web.ResultsServl
et"}
  }
]
```

The Job Resource

The job resource is used to handle job requests, such as starting new one-time jobs.

This is an internal resource and only visible/accessible in a password secured private area.

Resource URI

/backend/job?[job=...]&[name=...]&[group=...]&[command=...]

Resource Parameters

Parameter	Type	Description
job	String, *optional**	One of the following jobs: `cleanupdb, crawl, ratewebsite, locationcrawl, cleanupdbextensive, onetimecrawl, periodic`
name	String, *optional**	The name of the job used for the command.
group	String, *optional**	The group name of the job used for the command.
command	String, *optional**	One of the following commands: `run, interrupt, pause, resume, unschedule`

** Requires at least a valid job or a valid triple [name, group and command]*

The GET Method

The GET method provides nothing; It only processes the request data and returns status codes.

Response Media Type

–

Expected Response Status Codes

On successful completion, the resource responds with status code 200 OK.
For a complete list of possible response status codes, see HTTP Response Status Codes.

Request Example

```
GET http://localhost:8080/api/log?job=periodic
```

Request Response Example

```
-
```

The Settings Resource

The settings resource is used to handle new setting values and saves them in the settings file.

This is an internal resource and only visible/accessible in a password secured private area.

Resource URI

/backend/settings?[*see Resource Parameters*]

Resource Parameters

Parameter	Type	Description
maximumKeywordResults	Integer, optional	Maximum Keyword Results sent back
maximumParallelLocationCrawlers	Integer, optional	Maximum Parallel Location Crawlers
recommenderScaleInput	Double, optional	Recommender Scale Input
recommenderScaleHistory	Double, optional	Recommender Scale History
recommenderScaleSituation	Double, optional	Recommender Scale Situation
recommenderScaleLocation	Double, optional	Recommender Scale Location
recommenderScaleTags	Double, optional	Recommender Scale Tags
recommenderScaleUserHistory	Double, optional	Recommender Scale User History
recommenderScaleUserLocationHistory	Double, optional	Recommender Scale User Location History
recommenderScaleGeneralHistory	Double, optional	Recommender Scale General History
recommenderScaleGeneralLocationHistory	Double, optional	Recommender Scale General Location History

recommenderScaleUser	Double, optional	Recommender Scale User
recommenderScaleGeneral	Double, optional	Recommender Scale General
collSizeFHistory	Integer, optional	Number of results for F_his() function
collSizeFLocation	Integer, optional	Number of results for F_loc() function
collSizeFTag	Integer, optional	Number of results for F_tag() function
familiarPlacesRadius	Double, optional	Familiar Places Radius in meter
lbObjectNearRadius	Double, optional	Local browsing Object Near Radius in meter
generalNearRadius	Double, optional	General History Near Radius in meter
privateNearRadius	Double, optional	Private History Near Radius in meter
generalHistoryDaysBack	Integer, optional	General History Days Back
privateHistoryDaysBack	Integer, optional	Private History Days Back

The GET Method

The GET method provides nothing; It only processes the request data and returns status codes.

Response Media Type

–

Expected Response Status Codes

On successful completion, the resource responds with status code 200 OK.
For a complete list of possible response status codes, see HTTP Response Status Codes.

Request Example

```
GET http://localhost:8080/api/settings?
maximumKeywordResults=12&privateNearRadius=1337.00
```

Request Response Example

```
-
```

The Website Lookup Resource

The website lookup resource is used to handle informations about websites.

Resource URI

/lookup?[url=...]&[lat=...]&[long=...]

Resource Parameters

Parameter	Type	Description
url	String; *optional**	The unique URL of the website.
lat	Number (double); *optional**	The latitude which is needed to find all websites in the near; Also requires a valid longitude.
long	Number (double); *optional**	The longitude which is needed to find all websites in the near; Also requires a valid latitude.

** Requires at least a valid location (latitude and longitude) or a valid url*

The GET Method

The GET method provides a list of LbObjectMapping. The content of this list can be:

1. If there is a given url, the list contains exactly one LbObjectMapping (with the given url).
2. If there is a given location, the list contains 30 LbObjectMapping which are near the given location.

Response Media Type

`java.util.List<ch.ethz.mis.lb.server.web.mappings.LbObjectMapping>`
in short: `List<LbObjectMapping>`

Expected Response Status Codes

On successful completion, the resource responds with status code 200 OK.
For a complete list of possible response status codes, see HTTP Response Status Codes.

Request Example

```
GET http://localhost:8080/api/lookup?url=http://www.starbucks.ch
```

Request Response Example


```
[
  {
    "name": "Starbucks",
    "locations": [
      {
        "latitude": 47.3724,
        "longitude": 8.54397
      }
    ],
    "description": "",
    "distance": -1.0,
    "weight": -1.0,
    "lbObjectType": "WEBSITE",
    "lbObjectProperties": {
      "title": "",
      "url": "http://www.starbucks.ch"
    }
  }
]
```

HTTP Response Status Codes

Status Code	Description
200 OK	Successful completion.
400 BAD REQUEST	One or more parameters are not valid (e.g. String instead of double) or there are missing required parameters.
401 UNAUTHORIZED	The session has no valid user.
500 INTERNAL SERVER ERROR	An uncatched exception or some other fatal errors occurred.

Type Reference

This section includes the definitions of most of the used media types.

LbObjectMapping

Media Type URI

`ch.ethz.mis.lb.server.web.mappings.LbObjectMapping`

Base Attributes

Attribute	Type	Description
name	String	The name (title) of a local browsing object (e.g. webpage).
locations	List<Location>	The list of all locations of this object (e.g. multiple locations for a shop).
description	String	The description of this object.
distance	Double	The distance between the current location and the location of this object.
weight	Double	The weight (importance) of this object.
lbObjectType	LbObjectType	The type of this object (e.g. Webpage).
lbObjectProperties	Map<String, String>	Additional properties of the object depending on the LbObjectType.

JSON Representation Example

```
{
    "name": "ETH Zürich",
    "locations": [
      {
        "latitude": 47.376732,
        "longitude": 8.547943
      }
    ],
    "description": "lorem ipsum",
    "distance": 26.0,
    "weight": 0.9826300767314771,
    "lbObjectType": "WEBSITE",
    "lbObjectProperties": {
      "title": "ETH Zürich",
```

```
      "url": "http://www.ethz.ch"
    }
  }
```

HistoryObjectMapping

Media Type URI

`ch.ethz.mis.lb.server.web.mappings.HistoryObjectMapping`

Base Attributes

Attribute	Type	Description
url	String	The unique URL of the corresponding LbObject for this history entry.
lbObjectType	LbObjectType	The type of the corresponding LbObject for this history entry (e.g. Webpage).
dateTime	Date	The date and time when this HistoryObject was created.
location	Location	The location of the user when he/she created this object.
userAction	UserAction	The user action which created this HistoryObject.
parameters	Map<String, String>	Additional properties of the HistoryObject.

JSON Representation Example

```
{
  "url": "http://www.ethz.ch",
  "lbObjectType": "WEBSITE",
  "dateTime": "2012-06-12T17:18:14+0200",
  "location": {
    "latitude": 8.54458,
    "longitude": 47.38191
  },
  "userAction": "FEEDBACK",
  "parameters": {
    "feedback": "POSITIVE",
    "past_url": "www.google.ch",
    "someOtherParameter": "someValue"
  }
}
```

Location

Media Type URI

`ch.ethz.mis.lb.server.objects.Location`

Base Attributes

Attribute	Type	Description
latitude	Double	The latitude of a location
longitude	Double	The longitude of a location

LbObjectType

Media Type URI

`ch.ethz.mis.lb.server.objects.LbObjectType`

Base Attributes

Enum	Type	Description
WEBSITE(1)	enum (Integer)	The enum which represents a website.

Feedback

Media Type URI

`ch.ethz.mis.lb.server.objects.Feedback`

Base Attributes

Enum	Type	Description
POSITIVE(1)	enum (Integer)	The enum which represents a positive feedback.
NONE(0)	enum (Integer)	The enum which represents no feedback.
NEGATIVE(2)	enum (Integer)	The enum which represents a negative feedback.

UserAction

Media Type URI

`ch.ethz.mis.lb.server.objects.UserAction`

Base Attributes

Enum	Type	Description
BUTTON_PRESSED	enum	The enum which represents that the user pressed a button as action.
LINK_OPEN	enum	The enum which represents that the user opened a link as action.
NONE	enum	The enum which represents no action.
FEEDBACK	enum	The enum which represents that the user gave a feedback as action.

LogEntry

Media Type URI

`none`

Base Attributes

Attribute	Type	Description
timestamp	Date	The date when this log entry was created.
level	String	The log level which could be one of the following: “TRACE”, “DEBUG”, “INFO”, “WARN”, “ERROR” and “FATAL”.
message	String	The message of this log entry.
method	String	The method name in which the log entry was created.
lineNumber	Integer	The line number in which the log entry was created.
class	String	The fully qualified class name of the class in which the log entry was created.

JSON Representation Example

```
{
  "timestamp" : { "$date" : "2012-06-12T16:51:47.958Z"},
  "level" : "WARN",
  "message" : "Wrong formated latitude",
  "method" : "doGet",
  "lineNumber" : "93",
 "class" :
{ "fullyQualifiedClassName" : "ch.ethz.mis.lb.server.web.StartSession
Servlet"}}
```

Date

Media Type URI

`java.util.Date`

Format

`yyyy-MM-dd'T'HH:mm:ssZ`

JSON Representation Example

```
"2012-06-27T14:45:37+0200"
```