



Google Technologies for Cloud Development Workshop, Tel-Aviv University

Hadar Weiss,
Ofir Israel,
Ari Sharon,
Dan Noter

1.9.2010

Table of Contents

Introduction	1
Purpose.....	1
Scope	1
Architecture	2
Overview.....	2
Backend	2
Frontend	3
Features	3
Operating Environment	3
System Features	4
Searching and analyzing of articles	4
Articles sorting and ranking.....	5
The Toolkit	7
Users support	8
Monitoring.....	9
Keywords search.....	10
Admin capabilities	11
Articles addition	12
Language support	13
Facebook Integration	13
Implemented Technologies.....	14
Application Code	15
Backend	15
Frontend	15
Responsibilities	17
Lessons Learned	18
Release Notes.....	18
Minor UI issues with Internet Explorer	18
Slow login	18
Comment detection	19
RSS Parsing	19
Rouge IFrame	19

Introduction

Purpose

iLike-IL is an application designed to elevate the level of awareness and understanding of the State of Israel. All around the world public opinion is determined by the media, which often presents a one-sided picture. In many places Israel, despite its size and population, is considered to be the biggest threat to world peace.

iLike-IL is an advocacy application designed to allow users an easy way to search and comment on articles related to Israel.

Combined with an aiding toolkit the user can easily find relevant tips and videos, and thus post a more knowledgeable talkback for other readers to see.

Moreover- with simple modifications to the search criterions, the application can be used as a PR application, designed to find articles on any particular issues (i.e McDonalds, Toyota, etc.) and allow company representatives to comment on that issue.

Currently, there is a lot of interest in the application from different organizations that deals with Israeli advocacy. Our main cooperation is with StandWithUS, one of the biggest organizations in the field. We are in contact directly with the manger in Israel who promised us full support publishing the site.

Scope

The goal of this application is to allow an easy and convenient way to search articles which contain talk-backs, and aid the user to post knowledgeable comments.

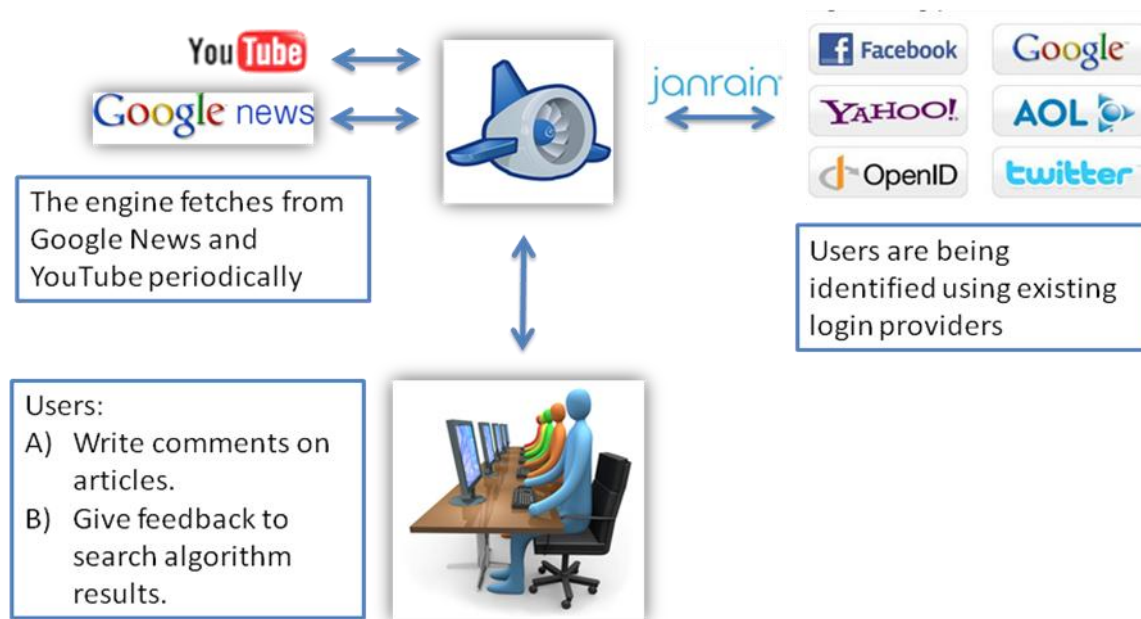
Registered users can save and track history of their activities, including articles they commented on, marked as favorites and/or marked as relevant/irrelevant.

In addition- users, and especially preferred users and admins- can effect on the articles ranking and relevance with their ranking.

Architecture

Overview

Our application was designed so that it could easily be changed to be a different PR application- by simply changing the keywords used for the article searching and analysis, the application can allow searching of articles on about any subject, thus allowing company representatives to easily find and comment on negative articles on their company.



Backend

The backend is responsible for processing and finally inserting new articles to the datastore. Using cron jobs (and task queues) we are scheduling our article "pipeline" which puts a new article through a series of checks (explained later). Afterwards, if qualified, the backend is responsible to add the article to the system. This includes, adding a newly discovered source to the DB, updating the cache and so on.

We also scheduled article recalculation process which updates the rank of the article which is described later.

Frontend

Consists page handlers, templates, javascript and JQuery scripts. Responsible for authenticating the user, page rendering and processing user input. Also consists the admin interface that allows editing the system entities.

Features

- Search and Analyze articles on Israel including talkback recognition, keywords analysis etc.
- Use a ranking algorithm to classify the articles based on their importance, and present the articles to the users based on their classification.
- The toolkit- help the user by presenting relevant tips and articles management buttons.
- Users support- allow users to register to the website using different accounts, and give preferred users with special capabilities.
- Monitoring- allow users to save information about articles- favorites, commented, relevant/irrelevant articles, as well as effect articles importance according to users actions.
- Keywords search- allow users to search for specific articles based on one or more keywords from the keywords database.
- Admin capabilities- allow admin users to manually add tips and keywords, manage sources and keywords priorities, and verify new (unverified) articles.
- Articles addition- allow users to manually add new articles easily using a special bookmarklet / favorites button.
- Language support- Spanish is also supported in addition to English, and other languages which are supported by google news can be easily added later.
- Facebook integration- Turn iLike-IL into a Facebook application.

Operating Environment

Our website functions correctly in all major web browsers, except minor problems with IE – see release notes. The system is of course OS agnostic (no Silverlight or Flash)

System Features

Searching and analyzing of articles

Description

New articles should continuously be added to the database in order for our application to be useful. The article which are added to the db should be relevant, and they should be processed and analyzed- both in order for us to have an elegant way to present them in the main page (including their title, short description, a picture etc.) and to keep information about those articles which will use us in the future (such as main keywords, language etc.).

The first and robust searching of articles is performed via Google News. We search for recent articles which contain the word "Israel".

We then go over the results, parse the RSS and get information on that article- title, description, date of creation, name of source etc.

After the RSS parsing is done, we need to decide whether this article is relevant or not, and if it is- process it and add it to the db.

Deciding if the article is relevant or not is done by two procedures:

- **Talkback Recognition:** *The first criterion for deciding if an article is relevant is deciding if an article has talkbacks. Since our application main goal is to help users comment on Israel-related articles so that other readers can read, we have to make sure that articles we add to the db have a comment section in them.*

Talk back recognition is done by parsing the html of the article and looking for the words "comments", "talkback"/"talk-back", "trackback"/"track-back" either as a link (in a xml tag) or as a xml tag itself.

We generally assume that for a given source, either it has talkbacks in its articles or it does not. That's why every new source we find is analyzed for talkbacks using the algorithm we described. Future articles from the same source won't be rechecked, but instead will be categorized as have/don't have talkbacks based on its source.

This assumption is usually true, but there are some sources which have a comment section in some of the articles, while in others they do not. An admin which identifies such a source can mark it as "Volatile", meaning that every future article fetched from that source will be checked individually for talkbacks using our algorithm.

- **Keywords analysis:** *Our application holds a database of keywords for relevant topics which we want to fetch articles on- i.e. Iran, Hamas, Flotilla etc. This db of keywords help us to focus on topics related to the State of Israel and current events, and prevent us from fetching articles on unrelated issues- i.e. Israel Uman, 'Beth-Israel' hospital in NY etc.*

If an article has talkbacks, it is then parsed to analyze the number of appearances of each keyword in the article.

In addition- a list of all keywords in the article is created and saved in the article information for future use, as well as the weight of the keywords in the article (number of keywords in the article, multiplied by their importance factor).

Only articles which have talkbacks and contain at least one of the keywords from the db are inserted into our articles database.

Flows

Articles fetching and analysis is done periodically once every hour, using a cron job.

Since articles analysis is pretty heavy and time consuming (and after receiving "deadline exceeded" error from time to time), we have decided to fetch and analyze articles using task queues.

Our fetching algorithm can also be executed manually by accessing the <site url>/fetch. In this case- articles analysis is done immediately, and not through a task queue, to allow immediate insertion of new articles to the database.

Functional Requirements

We aim to continuously fetch new articles, to allow users to comment on the most recent articles posted in news sites all around the world.

Since our application is designed to allow users to post comments on articles, the first and most important analysis is determining whether an article has a comment section.

In addition- we want to fetch articles which regard the state of Israel and current events. That's why we analyze the keywords in the article and look for keywords which regard politics and current events in Israel's day agenda. The analysis of the keyword will also help us later- when we want to search for articles with specific keywords, match between an article and relevant tips, etc.

Articles sorting and ranking

Description

Articles should be sorted based on certain criterions- the articles source, its creation date, the number of appearances of the keywords and the weight of each keyword, and the votes of users for/against this article.

*Articles sorting is based on two different parameters- the article **importance**, and the article **relevance weight**.*

The Importance: *The importance of the article is the first and main parameter used to sort articles when presented to the user.*

Each article receives a rank which reflects how important the article is, based on the article source, and how new it is. The importance of each article is recalculated every hour using a ranking algorithm.

The algorithm: Every article is given an importance level between 0-5 based on the importance of its source and its date of creation.

When an article is first fetched, its importance is determined based on its source importance. Every 24 hours (that pass from the date of creation) the importance of the article is reduced. That way- newer articles are given higher importance than older ones, and there is a balance between the importance of the source and the date of the article.

In addition- registered users may vote for or against an article (see: The Toolkit feature). If multiple users voted for or against an article this will result in addition or reduction to articles importance, respectively.

(Remark: the original calculation of the importance was based also on the weight of the keywords in the article. However- this had negative effect on our sorting algorithm, due to the fact that longer articles have higher number of keywords. This is why we decided to use the keywords weight as only a second parameter for sorting).

When an article becomes old (between 4-7 days, based on its original importance level) it is reduced to importance of 0, meaning it is no longer displayed in the search results.

The relevance weight: The initial weight of an article is based on the number of appearances of each keyword in the article, and the weight of each keyword. This may reflect how relevant the article is, but may also be a subject to some noise due to the fact that longer articles will usually have more keywords in them. That's why the weight of the article is affected by votes from users. As more registered users vote for or against an article, its weight increases or decreases to reflect its relevance.

Articles presented to the user are sorted first by their importance, and articles with the same importance are sorted by their weight.

Remark: To optimize the site CPU usage and loading time, we decided to use caching to store search results. When sorting articles for a user who has not defined specific preferences (specific keywords, only his favorites, etc.), we store the results in the app engine cache. In that way- proximate entries to the web site by other users will use the same results, instead of recalculating for each entrance individually.

This has proven itself useful to reduce CPU usage and speed access time to our site.

Refreshing of the cache is done when new articles are fetched, and periodically every hour.

Flows

The algorithm which recalculates the importance of each article runs periodically every hour using a cron job.

When a user enters the site main-page, the articles in the db are sorted (or taken from the cache if possible) and presented to him in descending order.

Functional Requirements

The application should help user comment on articles which are as popular as possible.

Articles from big news sites such as CNN or FOX are usually read and commented by many readers, and newer articles are probably read more frequently than older ones.

We also assume that articles with many appearances of chosen keywords are probably more relevant to current issues.

In addition- since we know our sorting algorithm isn't perfect, we want to take into consideration the opinion of our users. That's why voting for or against an article by many users can push an article upper or downer in the sorting algorithm.

The Toolkit

Description

When entering an article, the user is given with a toolkit to help him manage and comment on the article.

The Toolkit is consisted from a few main parts:

- **Useful tips:** *Our database contains many useful tips such as videos and articles, which are designed to give useful information to the user when he wants to comment on an article.*

Tips can be added to the database by an administrator, which states for each tip its source and the main keywords it is relevant for.

For each article, the tips are sorted based on the matching between their keywords to the keywords in the article. The more matching keywords we find, the more relevant the tip is to that article.

- **User actions:** *Logged-in users can add an article to their favorites, mark an article as commented once they posted a comment, vote for or against an article to affect its relevance, and recommend it using Facebook.*

In addition- if this article has not been verified yet or commented on, the user may mark it as "has no comments". This is done to help us remove sources which the talkback-recognition algorithm has accidentally identified as "has talkbacks". When enough users vote against a source we flip the algorithm decision and mark it as "has no talkbacks".

If this is a power-user or admin, marking an article as "has no comments" will also remove the article from being presented in any future search results.

- **Power-users and administrator actions:** *A logged-in user which has a power-user or admin permission is given the option to verify new articles or remove unverified articles from the db which are irrelevant or have no talkbacks.*

This is done using the special buttons on the right side of the toolkit, which appear only when the user is a power-user/admin, and the article was not verified yet.

- **Additional control buttons:** *A few additional buttons appear in the toolkit to help it be user-friendly.*

The relevance bar shows the relevance (weight) of the article, as calculated from the article keywords weight and the number of users which voted for/against the article.

The user can also minimize the toolkit if he wants, or go to the original article by pressing the corresponding buttons.

Flows

When a user clicks on article he is sent to the article-page, which is consisted of two main parts- the toolkit at the top of the screen, and the article itself.

The user can read the article and comment on it directly from the article page, or if he prefers- he can press the button and go to the original article to comment there.

If the user does not like the toolkit for some reason, he can simply minimize it.

Using the designated tips box, the user can scroll over the tips using the tips scroll-bar and click on any tip he thinks is relevant. This will help him get additional information on related topics, and post a link to relevant articles/videos together with his comment if he wants to.

A logged-in user can add articles to his favorites using the "star" button. He can also mark articles he commented on by clicking on the "I commented" icon.

The user can also mark an article as relevant/irrelevant using the "thumb up/down" buttons. This will both help the user monitor articles he already read, and help our sorting algorithm increase or decrease the importance of articles.

If our algorithm got it wrong and this article has no comments section, the user can press on the "no comments section". This will help us catch sources with no comments sections which our talkback recognition algorithm has missed.

Power-users and admins can help verify new (unverified) articles by the additional buttons on the right side. Such a user can mark a new article as verified or irrelevant. Once an article has been verified by one power-user or admin, it is considered as verified, thus these extra buttons will not be presented again for this article.

Functional Requirements

The toolkit is designed to help the user- both by getting additional information using the tips, and by giving him an easy way to mark his preferences for different articles.

The toolkit thus has to be intuitive and user-friendly.

Users support

Description

The application support signing-in of users using different accounts- Google account, Facebook, twitter, yahoo, AOL or OpenID.

The signing-in is done using the Janrain tool.

Once a user signs in, we can save information about him- language preference, how many comments did the user post, etc.

In addition- very active and helpful users can be pushed by the admin to be a power-user. Power-users receive special privileges- their voting weight is higher when voting for or against articles and they can verify new articles or mark them as irrelevant.

Flows

When a user enters the website, he can log in by clicking on the “login” button. After choosing the account type and entering his personal details, the user is redirected to the site- this time as a registered user.

Now the user can change his preferred language, enter an article and comment on it, and perform various actions using the toolkit (such as vote for/against an article, add to favorites etc.), and all information will be saved for the specific user.

Functional Requirements

Personalization of the application is an important feature. We want to save information for each user so that the next time the user enters the website, his preferences and actions will be available.

Monitoring

Description

When a user performs an action on the article, we want to be able to monitor it. If a user viewed an article, commented on it, voted for it or added it to his favorites, we want to know and save the information.

This will allow us to know some information about the article itself- how many users viewed it, did someone already post a comment on that article etc.- as well as keep information for the user- articles he added to his favorites, voted for, commented on etc.

This allows users to know which articles have not been viewed or commented on, as well as allow the user to monitor which articles did he favor, thought were relevant, commented on etc.

Cleaning the DB: *After a few days an article importance is reduced to 0, and it is no longer displayed in the search results in the main page. However- the article itself is saved in the db for 90 days, to allow registered users to see old articles which they added to their favorites or commented on. When a registered user clicks on the “my comments” or “favorites” buttons, he’ll see all relevant articles from the last 90 days (and not just from the last few days, as in the general search).*

After 90 days an article- along with all of the activities made on it, is removed from the db to prevent overflowing.

Flows

After a user logged-in, he can see for every article in the main page the actions he performed on it (including actions he performed in previous entries to our website). If he added an article to his favorites- a 'star' will appear near it, if he commented- a David-shield, and if he voted for/against it- a 'thumb up/down icon'.

A user can press on the "my comments" or "favorites" buttons to see articles he previously commented on or added to his favorites, respectively.

When a user enters the article page- he'll be able to perform these actions by clicking on the designated buttons in the toolkit. These buttons are interactive, meaning that if a user pressed the "I commented" button- the button will not appear again. If he added the article to his favorites or voted for or against it, the button will be changed so that the user knows he already clicked it. A second click will cancel the previous action.

Functional Requirements

We want to monitor user's action, to allow us both to monitor the status of articles and to hold personal information for each registered user. The information should be displayed in an elegant and intuitive way- small icons near articles, a simple click to see your favorites etc.

Keywords search

Description

A user can search articles based on specific keywords.

During fetching of an article, it is analyzed for all of the keywords it contains from our keywords database.

When a user chooses a keyword to search articles by, only recent articles which contain that keyword will be displayed in the search results.

If the user chose more than one keyword, then only articles which contain all of those keywords will be presented in the results.

Flows

When a user enters the main page, he can use the search box to enter one or more keywords. The user simply starts typing a keyword in the search box, and by auto-completion he can see all keywords with the same prefix which exist in the database.

If he wants to search for multiple keywords, he should simply separate between them with spaces. The auto-complete feature works for every word separately.

In addition- below the search box, the user can browse on the hot topics (keywords which receive high importance), and click on any of the hot topics to receive only articles on this specific topic (only articles which contain that keyword).

Functional Requirements

Some users may want to look or comment on specific topics- i.e. specific topics which are very relevant (hot topics) or a topic which the user has a vast knowledge on.

Admin capabilities

Description

The admin (or admins) of the application are the ones which control and manage the tips, keywords, sources and users.

An admin can add new tips, add or remove keywords from the db, changes the priority of keywords and sources, and move a user to be a power-user or even another admin.

Flows

When a user with admin permission login to the site, he can see on the upper right side (near the "My comments" and "favorites" buttons) an 'admin' button. If the user does not have an admin permissions, this button will not appear.

When clicking on that button- the user is redirected to the admin page.

In the admin page the user can switch between several tabs:

Tips: *In the 'Tips' bar the user can add tips to the database. The user can simply paste the tip url to the url box, and click on the "Get metadata from url" to get some of the other fields (such as 'title' or 'description') automatically filled. Since the filling algorithm is not prone-proof (it is based on parsing the html of the tip), we give the user the option to change the auto-completed fields before posting the tip.*

The user should manually insert the keywords which are relevant to this tip (at least one keyword should be given), and as in the search box- auto complete is used to fill the keywords from the db once the user begins to type.

Keywords: *In the 'Keywords' tab, the user can add new keywords to the db, change the importance of keywords and remove keywords which are no longer relevant.*

Sources: *In the 'Sources' tab, the user can go over all sources already in db.*

Each source is automatically added to the db the first time an article is found from that source. All sources are given low importance at first. The admin can change the importance of the source to higher importance, decide if this source has talkbacks (and change manually the decision of the talk-back recognition algorithm), or set a source to be volatile (as explained in the talkback-recognition algorithm).

Users: *In the 'Users' tab, the admin can see all users in the db and give specific users with power-user capability. He can also add additional admins to the application.*

Functional Requirements

The databases should be controlled by the admins of the application. They should have an easy way to add tips, decide what the hot topics are, decide which sources are more important to others and give specific users with special capabilities.

Articles addition

Description

Users have an easy way to add new articles to the database.

A user may add an article by entering its url in the article-addition page.

More easily- user may add a bookmarklet to his bookmarks in Google Chrome- or if he is using IE, a button to his favorites- which allows him to manually add an article to the database with a simple click.

After the user has posted this article, the article is processed in the same manner newly fetched articles are processed to decide whether it has talkbacks and contains at least one of our keywords.

Flows

When a user enters the main page, he can click on the "Help us by suggesting an article to iLike-IL" link and go to the article-addition page. He can then paste the articles url in the url box, and click on the "get metadata from url" to fill the article information automatically (by parsing the html of the article). If not all fields are automatically filled, or if they were filled incorrectly (due to wrong parsing of the html), the user can manually change it before posting the article.

Another simple way to suggest articles to iLike-IL is to use our 'suggest to iLike-IL' bookmarklet. From the article-addition page, the user can click and drag on the relevant link in order to add our bookmarklet to his bookmarks.

When a user finds a new article which he thinks is relevant, he can simply press on our bookmarklet in order to add it to our site. The article-addition page will then open up with the article metadata automatically filled in the fields. As before- the user can manually change the fields before posting the article.

After the user has completed posting the article, the article will be analyzed for talkbacks, keywords and importance level, and if the article meets all criterions- he will be added to the db and presented in future search results.

Functional Requirements

We want to allow users to manually add articles in an easy manner. Though the database is continuously filled with articles automatically, we want to allow users to add additional articles manually to help us increase the number of relevant articles in the database.

The addition of new articles should be as easy as possible and with minimum effort to encourage many users to help.

Language support

Description

Our application can be expanded to support additional languages besides English. By simply changing the automatic fetching algorithm to search articles in other language, we can add to our db articles from different languages.

All fetched articles are classified to languages, and when the user chooses its preferred language we can easily present him with only articles from his chosen language.

We have added Spanish support in our application to demonstrate the use of different languages.

Flows

After a user has logged in to our website, he can switch from English to Spanish in the main page by clicking on the "espanol" button on the right-hand side (near the "favorites" button). As a result- articles in Spanish will be presented in the search results instead of articles in English.

The user can click on the "English" button to go back to English results.

Note that the preferred language is saved in the user preferences, so that the next time the user logs in to our site he will automatically receive search results in his preferred language.

Functional Requirements

We want to enable support to multiple languages to allow users from different parts of the world participate in our effort.

We should try and make language-based features as general as possible, to allow addition of future languages with minimum effort.

Facebook Integration

Description

We integrated Facebook in two ways.

Facebook Application: *We have a stripped down version of our app, adjusted for Facebook, so our site is shown in the (small) Facebook IFrame.*

Publish Articles: *Users can publish they've commented in their Facebook profile. They can also recommend an article through Facebook.*

Flows

The user can enter the application thru his Facebook account instead of our site.

When a user clicks on an article, he can recommend the article in Facebook using the “recommend” button in the toolkit.

Moreover- after commenting on an article, the user is questioned whether to publish that he commented on his Facebook wall. When his friends see it on his wall, they might want to comment as well.

Functional Requirements

Let the user interact through Facebook and share with friends. This way our site can gain more users and publicity.

Implemented Technologies

iLike-IL is written in python, and designed in HTML over Google App engine.

Our application utilizes the following Google App Engine APIs:

- ❖ *The Datastore*
- ❖ *Scheduled Tasks (Cron jobs)*
- ❖ *Memory caching - memcache*
- ❖ *Task queues*

Additional technologies and APIs used in our application:

- ❖ *YouTube Data API (for the YouTube feeds)*
- ❖ *Janrain API (for the multiple-platform login)*
- ❖ *Facebook API (for the application)*
- ❖ *JQuery Framework - for most of the Javascript in the system, especially for the client-side AJAX.*
- ❖ *JQuery UI - for most of our client-side UI features*
- ❖ *DateUtil - for easy parsing of dates (from manually-added articles)*
- ❖ *Google News RSS feeds – not an official API, but we use it anyway*
- ❖ *Google Analytics – monitoring users and interactions*

- ❖ *Google AppStats – profiling our app in detail*
- ❖ *BeautifulSoup – parsing html from source sites and Google News*

Application Code

As described above, the application is comprised of a frontend and a backend. Please find the relevant Code below:

Backend

1. Addarticle.py – Functions to get meta-data from a manually added article/tip.
<http://code.google.com/p/ilikeil/source/browse/trunk/src/addarticle.py>
2. Feedparser.py – Fetching and parsing of RSS feed and YouTube data feed.
<http://code.google.com/p/ilikeil/source/browse/trunk/src/feedparser.py>
3. Pipeline.py – Generates the flow of the new article fetching process.
<http://code.google.com/p/ilikeil/source/browse/trunk/src/pipeline.py>
4. ProcessArticle.py – Functions to allow talk-back recognition.
<http://code.google.com/p/ilikeil/source/browse/trunk/src/processarticle.py>
5. Article_updater.py – Functions used to re-calculate article importance.
http://code.google.com/p/ilikeil/source/browse/trunk/src/article_updater.py
6. Secretshandler.py – Module that allows us saving secrets (such as the Facebook Application Secret) in the Datastore (and not hard-coded in the open-source Google Code platform).
<http://code.google.com/p/ilikeil/source/browse/trunk/src/secretshandler.py>
7. Rpxusers.py – Functions to allow logging-in and logging-out, saving session information in the cookie, and user management.
<http://code.google.com/p/ilikeil/source/browse/trunk/src/rpxusers.py>
8. Common.py – Definition of all of our Datastore models.
<http://code.google.com/p/ilikeil/source/browse/trunk/src/common.py>
9. Keywords.py – Functions used for the keyword search in articles.
<http://code.google.com/p/ilikeil/source/browse/trunk/src/keywords.py>

Frontend

10. Main.py – The main handler for requests to the site.
<http://code.google.com/p/ilikeil/source/browse/trunk/src/main.py>
11. Admin.py – The handler for the admin page requests and actions.
<http://code.google.com/p/ilikeil/source/browse/trunk/src/admin.py>

12. Fbutils.py – Module for allowing logging-in from Facebook through our Facebook application.

<http://code.google.com/p/ilikeil/source/browse/trunk/src/fbutils.py>

13. Templates:

a. Main.html – a base template for the site.

<http://code.google.com/p/ilikeil/source/browse/trunk/src/main.html>

b. Articleboxes.html – the main page template.

<http://code.google.com/p/ilikeil/source/browse/trunk/src/articleboxes.html>

c. Fb_articleboxes.html – the main page for our Facebook application.

http://code.google.com/p/ilikeil/source/browse/trunk/src/fb_articleboxes.html

d. Article.html – template for showing an article.

<http://code.google.com/p/ilikeil/source/browse/trunk/src/article.html>

e. Min_Article.html – template for showing an article with the minimized bar.

http://code.google.com/p/ilikeil/source/browse/trunk/src/min_article.html

f. Admin.html – template for the admin page.

<http://code.google.com/p/ilikeil/source/browse/trunk/src/admin.html>

g. Support_template.html – template for the support-feedback page.

http://code.google.com/p/ilikeil/source/browse/trunk/src/support_template.html

h. About_template.html – template for the about page.

http://code.google.com/p/ilikeil/source/browse/trunk/src/about_template.html

i. Addarticle_template.html – template for the add article page.

http://code.google.com/p/ilikeil/source/browse/trunk/src/addarticle_template.html

j. Bkmrkl.js – Javascript code to allow the bookmarklet to appear.

<http://code.google.com/p/ilikeil/source/browse/trunk/src/static/bkmrkl.js>

- In addition to the above files we used several graphical libraries (jQuery etc.). Relevant files can be found here:

<http://code.google.com/p/ilikeil/source/browse/trunk/src/static>

Responsibilities

During the Semester, we have split the work and responsibilities based on the features list we have put together, the urgency of the issues (show-stopper bugs, must-have features etc.) and the technical skills of each of the members.

At some stages we each worked alone, but we generally preferred to sit and work together when possible.

We usually assigned each feature to one user, though in some cases there was mutual or overlapping work.

List of responsibilities:

Ari:

- *Admin page logics*
- *Front page JQuery slideshow integration and design*
- *Language support*
- *YouTube integration*
- *DB management- adding tips and keywords, managing sources, etc.*

Ofir:

- *Talkbacks recognition*
- *ArticlePage design and implementation*
- *Manual addition of articles through the Bookmarklet & Article-addition page*
- *Facebook integration*
- *Integration with the 'Janrain' tool for multiple platforms login possibility*

Hadar:

- *Main page design and template hierarchy*
- *Article activities and user interactions*
- *Monitor performance and optimizations, caching and task queues*
- *Hot keywords feature*
- *RSS parsing*

Dan:

- *Keywords processing and analysis.*
- *Implement algorithm for articles importance calculation.*
- *User actions handling.*
- *Articles sorting based on different parameters (importance, relevance, preferred keywords, user preferences, favorites etc.)*

Lessons Learned

The workshop has proven itself to be very demanding and challenging. Developing a big project in an unfamiliar work ground can be very difficult, especially for those of us who had no real former experience in programming and web development.

But at the end- we are pretty proud of what we've accomplished, and we have learned a lot along the way.

Here is a list (very partial) of some of things we've learned:

- Developing a large scale project from scratch.
- Working in a team and dividing the work between members.
- Prioritizing between missions due to time line restrictions.
- Self-learning of new programming languages- Python, JavaScript, HTML, GQL.
- Learning how to work with different APIs and unknown web-technology.
- Familiarizing with web development tools such as SVN, Data Viewer and Google Analytics.
- Facing GUI designing considerations and problems- different browsers and resolutions limitations, how to design a user-friendly HTML page, etc.

Release Notes

We have few known limitations:

Minor UI issues with Internet Explorer

JQuery tab plugin is not compatible with IE8 so the tabs in the admin page are not rendered with JQuery, but as plain html.

Slow login

Sometimes the Janrain user mechanism works slowly. It means, the user can wait ~10 seconds before being logged in. The cause is Janrain and sometimes Facebook which can be slow.

Comment detection

Our talk-back detection algorithm works with heuristics which are not 100% bullet proof, meaning that articles without comments are often shown to the user. We get feedback from users (using the "no comments" button) on non-talkback articles. Our algorithm slowly learns which sources are actually without talkbacks and which sources probably include comments section.

RSS Parsing

The RSS that comes from Google is not 100% readable. Namely, the description element contains many fields which are all encoded inside of a big html table. Sometimes BeautifulSoup is unable to extract the text correctly.

Rouge IFrame

We use an IFrame in the article page to show the original article and our own content simultaneously. Sometimes the content sites do "tricks" to take control over the page. There are ways to "fight back" but we don't employ any of these at the moment.