

A Content-Sensitive Wiki Help System.

A Project Report

Presented to

The faculty of Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Eswara Satya Pavan Rajesh Pinapala

December 2014

© 2014

Eswara Satya Pavan Rajesh Pinapala

ALL RIGHTS RESERVED

SAN JOSE STATE UNIVERSITY

The Undersigned Project Committee Approves the Project Titled
A Content-Sensitive Wiki Help System.

by

Eswara Satya Pavan Rajesh Pinapala

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Chris Pollett, Department of Computer Science

Date

Dr. Chris Tseng, Department of Computer Science

Date

Dr. Suneuy Kim, Department of Computer Science

Date

APPROVED FOR THE UNIVERSITY

Associate Dean Office of Graduate Studies and Research

Date

Abstract

Context-sensitive help is a software application component that enables users to open help pertaining to their state, location, or the action they are performing within the software. Context-sensitive “wiki” help, on the other hand, is help powered by a wiki system with all the features of context-sensitive help. A context-sensitive wiki help system aims to make the context-sensitive help collaborative; in addition to seeking help, users can directly contribute to the help system. I have implemented a context-sensitive wiki help system into Yioop, an open source search engine and software portal created by Dr. Chris Pollett, in order to measure the effectiveness of said help system. An experimental evaluation study has been performed on users of Yioop and the results are discussed in this report.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my advisor Dr. Chris Pollett for the useful comments, remarks and engagement through the learning process of this Writing Project. Furthermore I would like to thank my committee members Dr. Chris Tseng and Dr. Suneuy Kim for their effort, time and feedback. Also, I like to thank the participants in my experimentation, who have willingly shared their precious time during the process of testing. I would like to thank my parents, Usha Rani and Nagesh Kumar, and wife, Shamantha Sowmya, who have supported me throughout entire process, both by keeping me harmonious and helping me putting pieces together.

Table of Contents

1. Introduction.....	5
2. Yioop Open Source Software	7
3. Background and Preliminary Work	9
3.1 Context-Sensitive Help	9
3.2 Context-Sensitive Help in Microsoft Windows	10
3.3 Procedural Context-Sensitive Help	11
3.4 Implementing Context-Sensitive Help Using Popular Help-Authoring Tools	12
3.5 Research on Wiki Systems.....	14
3.5.1 MediaWiki.....	14
3.5.2 TikiWiki.....	16
4. Context-Sensitive Wiki Help Design	20
4.1 Designing the Help User Interface	20
4.2 Designing the Storage and Retrieval of Help Articles	23
4.3 Editing the Help Articles	25
5. Implementation of Context-Sensitive Wiki Help	27
5.1 Implementing the Wiki Editor	27
5.2 Implementing the Web Service to Retrieve Wiki Pages	31
5.3 Implementing the Wiki Parser and Help Display.....	33
5.3.1 Rendering the Help Button.....	33
5.3.2 Wiki Page Retrieval and Help Display	36
5.4 Implementing Upgrade in Yioop.....	41
6. Experimenting on Context-Sensitive Wiki Help in Yioop.	44
6.1 User-Assisted Testing.....	44
6.2 UI Testing Using PhantomJS	48
7. Conclusion	50
7.1 Future Work	51
References	52

List of Figures

Figure 1: Context-sensitive help for a basic login form	9
Figure 2: Screen-based help in Windows 3.1.....	10
Figure 3: “What’s This?”-based help in Windows 2000	11
Figure 4: Procedural help in Madcap Flare 3.....	12
Figure 5: Context-sensitive help integrated into a Yioop page	14
Figure 6: MediaWiki basic wiki editor	15
Figure 7: Sample page_table database entry	16
Figure 8: Sample rows in the revisions table	16
Figure 9: TikiWiki homepage view.....	16
Figure 10 TikiWiki editor view.....	17
Figure 11: Markup comparison between MediaWiki and TikiWiki.....	19
Figure 12: Help-mode initialization and usage mockup	20
Figure 13: Context-sensitive help design in Yioop (left-to-right).....	21
Figure 14: Context-sensitive help design in Yioop (right-to-left).....	22
Figure 15: Context-sensitive help design for Mobile interface	22
Figure 16: Retrieval of Wiki articles in Yioop.....	24
Figure 17: Asynchronous retrieval of help articles using a web service from Yioop.....	25
Figure 18: Initial version of wiki editor	29
Figure 19: Fully implemented wiki editor	29
Figure 20 Sample URL format for retrieving a Wiki page.	31
Figure 21: HTTP GET call to retrieve a wiki article over the web service	32
Figure 22: Form for adding a Locale.....	34
Figure 23 HTML button for Context-Sensitive Wiki Help.....	35
Figure 24: A web page on Yioop with no help article open	39
Figure 25: A web page on Yioop with English Locale selected and help article open	40
Figure 26: A web page on Yioop.com with Persian Locale selected and help article open	40
Figure 27: Help opened on a Mobile web browser (English on the left and Arabic on the right)	41
Figure 28 Yioop Database upgrade for Context-Sensitive Help addition.....	43
Figure 29 Long and descriptive help article.....	45
Figure 30 Context-Sensitive help for Add Locale after feedback from users.....	46
Figure 31 shows a Task-based help article with highlighted terms.	47

Figure 32 Procedural Help for Creating or Joining a Group.....	48
Figure 33 Shows PhantomJS UI Test results.	49

List of Tables

Table 1 Yioop Wiki markup.	28
Table 2 Wiki button identifiers.	30
Table 3 Code snippet for rendering a Context-Sensitive Help button.	34
Table 4 Attributes used in the Help button.	36

1. Introduction

A help system is a separate component of any software system that guides the user in performing the actions or tasks that the software system offers. The help system is also responsible for advertising the abilities and features of the software system. Any online application equipped with an efficient help system enables its users to perform tasks easily and painlessly. A context-sensitive wiki help system provides targeted information to users based on where they are, what their current state is, or what job they are performing with respect to the application and is also collaborative in nature.

Traditional help systems are nothing but a large portal of help articles divided into either chapters or pages that reside on external pages. When using this setup, however, the user has to move out of context to search for help, which could be extremely counterproductive. Context-sensitive help systems may take several forms [1], some of the most famous of which include widgets, page overlays, or hyperlinks to a different page or window. The main advantage of a context-sensitive wiki help system is that the user need not move out of context [1] [2] to get help for his or her current work; each help topic is applied exclusively for a given context.

Moreover, a wiki-based help system allows users to collaborate on the help content. Most wiki-based support systems are set up as portals – centralized sites where users can receive and contribute to help content. Although user collaboration systems can be very helpful, they carry the same problem as a traditional help system: the user must leave the context of their current action in order to seek help. This can negatively affect the productivity of the user and the pace at which he or she can use the application.

The basic idea of my project is to implement a collaborative context-sensitive wiki help system for a web application called Yioop. This system aims to combine the positive aspects of context-sensitive help with the collaborative nature of a wiki. Users receive targeted help based on their context, but also have the ability to edit and contribute to the help content.

This project explores how existing context-sensitive help systems work and sheds light on a few existing help publishing systems. A wiki system is already built into Yioop. The main deliverables in this project are Complete User interface for displaying context-sensitive help, Yioop WYSIWYG wiki editor, Yioop Wiki parser in JavaScript, Information Design of Context-Sensitive help article, Experimentation with the help from users and UI test suite and assertion framework for PhantomJS.

In Chapter 2, I will present the background for the project by introducing the Yioop Open Source Search Engine and the content management and wiki features it includes. In Chapter 3, we will look at the preliminary work done in the areas of context-sensitive help and wiki systems. In Chapter 4, I will throw some light on how the User interfaces and Wiki help implementations are designed. Chapters 5 will explain how I implemented the context-sensitive wiki help in Yioop in accordance with design discussed in chapter 4. Chapter 6 will show how I implemented user interface testing and experimentation with users on the context-sensitive help integrated into Yioop. Chapter 6 will also include how the new UI testing framework, PhantomJS, is used to test the context-sensitive help UI in Yioop. Finally, Chapter 7 will provide a conclusion and future work for this project and a subsequent list of references.

2. Yioop Open Source Software

This Research Project is to add a Context-Sensitive Wiki Help system to Yioop. Yioop would be a good starting point to experiment with a context-sensitive help powered by wiki. Yioop contains a vast number of features for both users and web administrators. Blogs, social groups, wikis, and a search engine are Yioop's most important offerings.

Yioop is open source search engine software that was initially designed to serve as a search engine, but acquired social and content management features down the line. Written using Model-View-Adapter [3], Yioop consists of a user-friendly interface that can be used to manage the software's features and easily navigate from one page to another. However, when users want to perform certain tasks on the Yioop website, it's not always easy to find help. The Seek Quarry website [4] hosts all documentation for the Yioop software system. Although the documentation is complete and clear, because it is primarily reference-oriented, it is sometimes difficult for users to acquire immediate help.

Searching for help in a large pool of items is also a pain point for users as it requires spending time to track down the needed information. The user also has to move out of the context of actions he or she is trying to perform on a particular page. This highly affects user productivity negatively and thus is not advisable.

A context-sensitive online help system is an embedded software component of the online system. It provides help to the user in their current context; that is, pertaining to the page the user is on or explaining the action that is possible to perform on the page the user is viewing. Wiki systems, on the other hand, are powerful collaborative

environments where users can contribute their knowledge. The use of wiki systems for maintaining help pages has already been implemented by many online systems. However, there currently aren't any context-sensitive help systems powered by wiki pages. Combining the power of context-sensitive help with wiki systems in order to benefit users is the main idea behind this project.

The goal of this context-sensitive help is to support users with all levels of knowledge on the attributes of the Yioop software. With proper guidance and help, I believe Yioop can be placed in more hands than most of the modern day Search engine software or Portals.

3. Background and Preliminary Work

3.1 Context-Sensitive Help

As discussed, a context-sensitive help system brings an immense advantage to users by enabling them to find the help they need quickly. Figure 1 shows a basic login form with context-sensitive help explaining the form fields. The help box opens up when a user clicks the “?” button. This is an example from Madcap software [5].

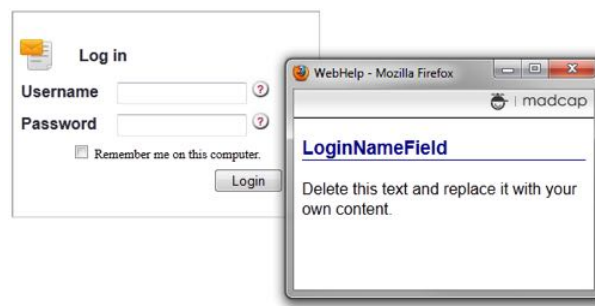


Figure 1: Context-sensitive help for a basic login form

There are various kinds of context-sensitive help, with descriptive context-sensitive help and procedural context-sensitive help being the two main types used in most software. There is Contextual help as well which is mostly used for definitions or short-text help. Descriptive context-sensitive help [6] is help provided based on the current screen or dialog, but tends to describe every field or option on the screen. This is a problem for users who need help while performing an action. If the users are trying to understand the each of the elements on the screen, then a descriptive approach will be helpful. However descriptive context-sensitive help is not user-centered. Context-sensitive help with a user-centered design [6] will be more useful to users who need help

while doing a job. These users are engaged in performing a task when they need help; they do not open help ahead of time.

3.2 Context-Sensitive Help in Microsoft Windows

Microsoft Windows is an excellent example of how context-sensitive help has evolved over time. I have installed older versions of Windows Operating systems to understand how Context-sensitive help evolved over time. The first versions of Microsoft Windows had screen-based help. A help button was provided on each window. Clicking the help button opened another pop-up window containing a definition or explanation of the first window, as shown in Figure 2.

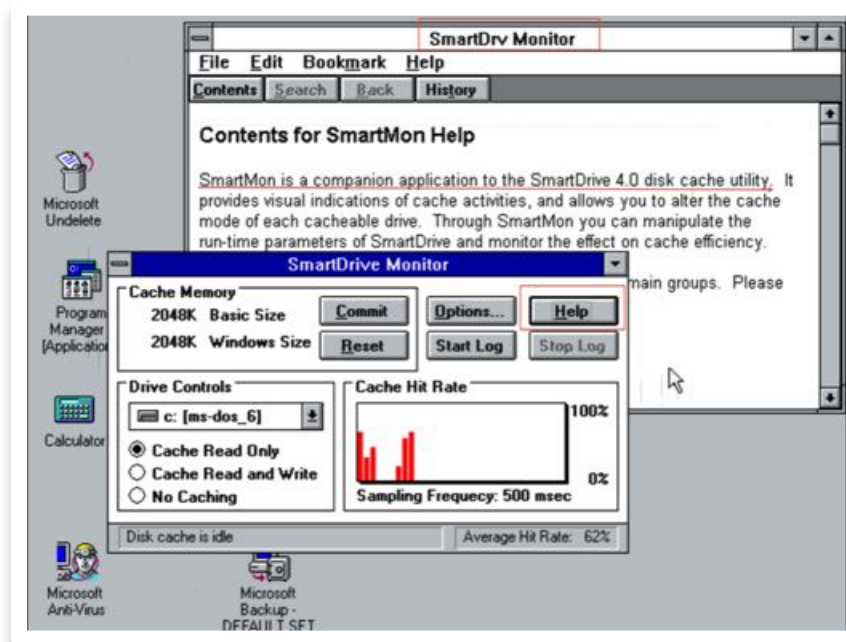


Figure 2: Screen-based help in Windows 3.1

Figure 2 is a screen grab of Windows 3.1 that I installed on a virtual machine.

Windows 3.1 was introduced with screen-based help. From Windows 95 until Windows

XP, Microsoft moved away from screen-based help and instead adopted “What’s this?”-based help.

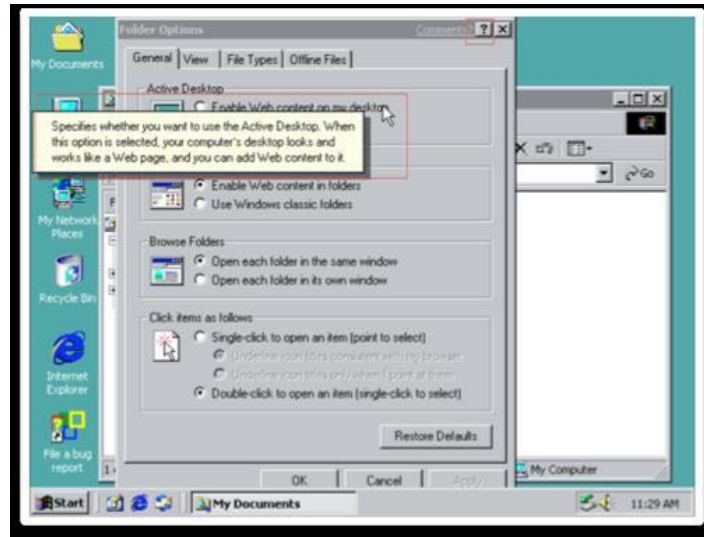


Figure 3: “What’s This?”-based help in Windows 2000

Figure 3 is a screen grab of Windows 2000 that I installed on a virtual machine. The “What’s this?”-based help, as shown in Figure 3, was the first step towards serious context-sensitive help. The user simply clicks on a question mark placed at the top right corner of the window, to enable the help mode, denoted by a question mark attached to the cursor. When the “?” is attached to the cursor, the user can then click on predefined areas on the window to view context-sensitive help text. Later versions of Windows followed with improved screen-based help and finally, more embedded help.

3.3 Procedural Context-Sensitive Help

The initial context-sensitive help system in Windows was descriptive (or reference-based) while later versions are procedural (or task-based). The main advantage of task-based implementation is that the user will be able to get help for exactly what he

or she is trying to accomplish. Instead of providing a reference for all elements on a page, task-based help explains how to perform specific tasks on a user interface.

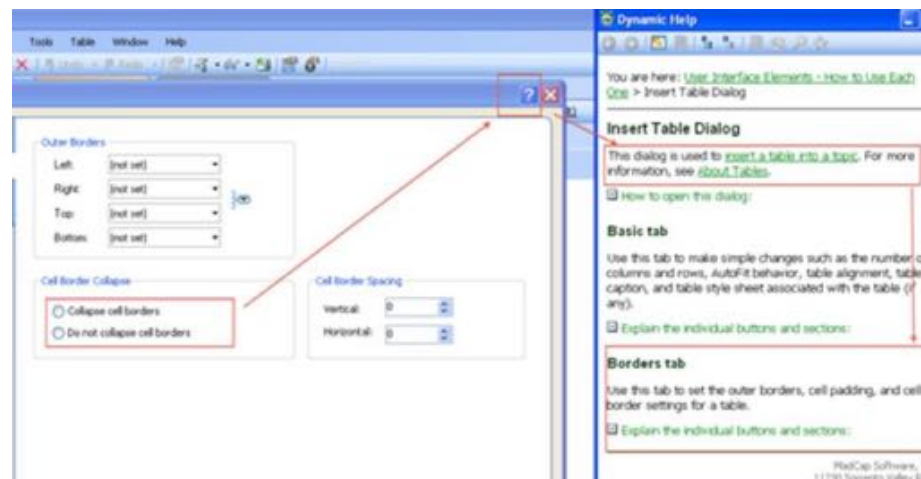


Figure 4: Procedural help in Madcap Flare 3

Figure 4 shows procedural help in the software Madcap Flare. This Figure is a screen grab of Madcap Flare I installed on Windows XP. Most importantly, the user is still on the page without losing context and is able to get help for any possible task that can be performed on that page.

3.4 Implementing Context-Sensitive Help Using Popular Help-Authoring Tools

There are many help authoring tools that provide turnkey implementation for embedding context-sensitive help into any web-based application. As a part of studying best practices for implementing the Context-Sensitive help into Yioop, I have looked at how popular Help authoring tools let help publishers implement context-sensitive help. In particular, I have looked at 2 major tools: Flare by Madcap [7] and Adobe Robohelp by Adobe Software [8]. I have implemented context-sensitive help using both Madcap Flare and Adobe Robohelp, and both are similar in terms of features and configuration. What

follows is a brief unified explanation of the context-sensitive help provided through the use of Madcap Flare and Adobe Robohelp.

Flare and Robohelp are traditionally used to generate help content that is split into sections. Thus, all the places on the web page where context-sensitive help is inserted – in other words, help topics – will be divided into individual chapters. Each chapter has a chapter ID which in turn is mapped to a subject having a topic name and topic ID. Both Flare and Robohelp use a concept called “Header files” and “Alias files”, both of which are plain text files used for mapping documents.

Header file entry: # define {TopicName} {TopicNumericId}

Alias file entry : <Map Name="{TopicName}" Link="/path/to/chapter_for_topic.htm" />

As one can see, Header files and Alias files are joined by the topic name. Together, Alias and Header files provide a way to identify the chapters using their unique names and identifiers as parameters. The HTML or JavaScript code embedded into the web pages can use the topic identifier information from the Alias and Header files.

Once the help content, along with header files and alias files, is created, Madcap Flare and Adobe Robohelp compile and generate all help content into a single directory. The compiled help content also includes a tiny JavaScript framework that can be embedded into any web application. The purpose of this JavaScript framework is to provide an easy programming interface for web developers to integrate the help content onto their web applications. Without the JavaScript framework, the developer would have to use the header and alias files to manually build to integrate context-sensitive help pop-ups. Figure 5 shows the context-sensitive help inserted into Yioop.

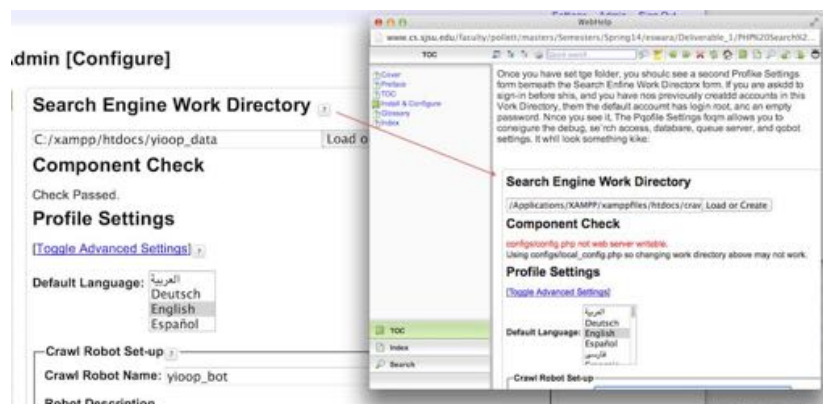


Figure 5: Context-sensitive help integrated into a Yioop page

3.5 Research on Wiki Systems

As part of my preliminary work, I looked at various wiki systems and their features. The wiki engines I ultimately chose are MediaWiki, TikiWiki, and Fossil Wiki. With Fossil being primarily a code revision control system with limited wiki features, I will explain the basics of MediaWiki and TikiWiki only. MediaWiki influences most of the features in Yioop's wiki system.

3.5.1 MediaWiki

MediaWiki is a wiki engine written in PHP. It is one of the most widely used wiki engines on the web, with Wikipedia being the most popular application using MediaWiki [9]. Wiki pages are created, accessed, and edited with the same URL aliases in MediaWiki. Every wiki page in MediaWiki has a “Page Name” attribute associated with it, which is also a unique identifier for the wiki page. Users can search for pages or access the wiki page URLs directly using the page names. If the page exists, MediaWiki renders an editor page for users to edit the wiki page. If the page does not exist, MediaWiki prompts the user to create a new page.

MediaWiki organizes content by arranging all similar content or pages into a namespace. Thus, namespaces tie together pages or content by their topic. A good example of a namespace is “category”. A page can be grouped into an existing category by providing the namespace as part of the wiki source.

Example: [[Category: Games]]

“Category” is the namespace and “Games” is the name of the category.

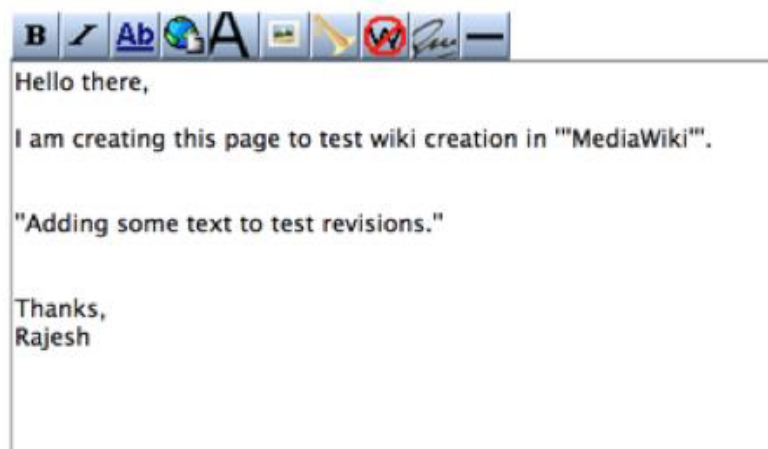


Figure 6: MediaWiki basic wiki editor

The MediaWiki editor shown in Figure 6 supports some basic content styling features such as bold text, italic text, underlined text, image insertions, and hyperlinks within its basic WYSIWYG (What You See Is What You Get) editor. In addition to the edited text itself, users will be able to add a summary to the page or the editing session.

Revision control for wiki articles is mostly maintained in 2 database tables in MediaWiki. Figure 7 shows an example view of page table. The table “page_table” stores the meta details of the wiki pages like “page_title”, “page_counter”, “page_namespace”, “page_id”, etc.

page_id	page_namespace	page_title	page_restrictions	page_counter
1	0	Main_Page	0B	20
2	6	Example.jpg	0B	1

Figure 7: Sample page_table database entry

The other table, named “revision”, stores all revisions of each wiki article. The wiki page contents are stored as blobs in the database by the name “old_text” with an “old_id” timestamp and other entries. Every new revision inserted into the revisions table points to the latest version of the concerned wiki article. The entire wiki text is stored as a data blob in the database instead of the diffs between revisions of the article. Figure 8 shows sample rows in the revisions table.

old_id	old_text	old_flags
1	'''MediaWiki has been successfully installed.'''Consult t...	524B utf-8 5B
2	'''MediaWiki has been successfully installed.'''Consult t...	529B utf-8 5B
3	'''MediaWiki has been successfully installed.'''Consult t...	538B utf-8 5B

Figure 8: Sample rows in the revisions table

3.5.2 TikiWiki

TikiWiki, like Yioop, is not just a wiki system but also a software portal and CMS. TikiWiki is written in PHP and is the basis of Mozilla’s support site.



Figure 9: TikiWiki homepage view

The TikiWiki homepage, as shown in Figure 9, is an editable default wiki page. However, I have the option to create a new wiki page altogether. To do this, I click on “Wiki” on the left sidebar to pull up a menu to create a new wiki page, along with the wiki editor.

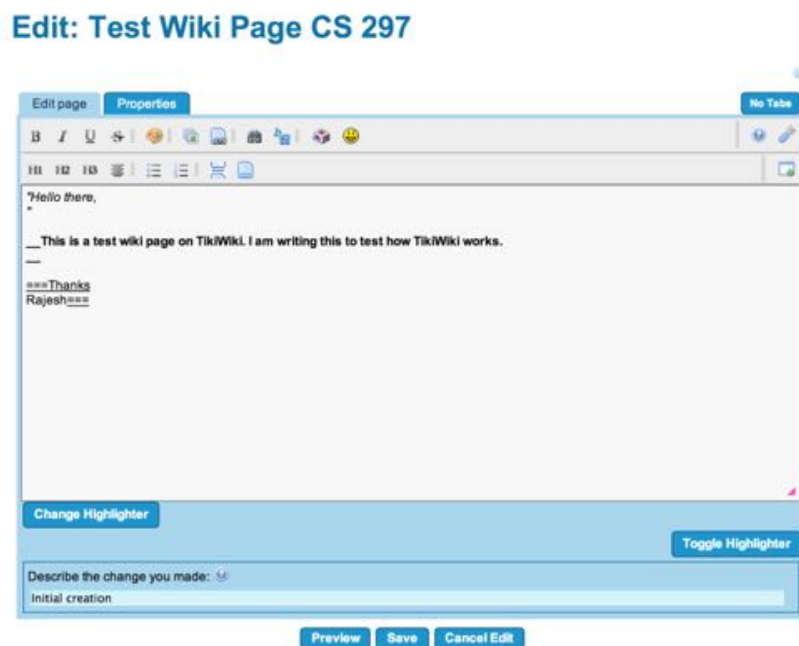


Figure 10 TikiWiki editor view

After writing the content as shown in Figure 10, we can add a comment pertaining to the current edit by entering it into the field “Describe the change you made” . Clicking on “Save” will save the page in the database.

TikiWiki stores revisions in the database to enable diff generation and diff comparison for each wiki page. TikiWiki stores all wiki page content in the tiki_pages table. However, it keeps track of version history in a table called tiki_history.

When an edit happens to a wiki page, the page details are inserted as a new record into the tiki_history table with a version number. The wiki content, along with the edits, are inserted as a data blob in the data field. Again, this is related to other wiki engines where the contents of the edited page are stored in their entirety instead of diff metadata.

There are various parameters in the tiki_pages table for each wiki page record. When a wiki page is created or edited, a new record is inserted into the tiki_pages table. The tiki_history table stores all revision history and edited wiki page blobs.

Due to the complicated nature of TikiWiki and taking into account the appealing features of MediaWiki, Yioop's wiki system is mostly influenced by MediaWiki. Most users of the World Wide Web are already used to the MediaWiki markup thanks to Wikipedia.com. Moreover, with MediaWiki being much more popular compared to TikiWiki [10], MediaWiki was the obvious choice. Figure 11 shows a comparison of wiki markup for basic styling in MediaWiki and TikiWiki.

Wiki functionality	MediaWiki	TikiWiki
Bold Format	" bold "	==bold==
Italics Format	" <i>italic</i> "	" <i>italic</i> "
Underline Format	<u>underlined</u>	===underline===
Internal Link	[[a link]] [[a link with title]]	{{Name of page}} or {{Name of page link to a page}} or {semantic(link to a page)} or, if activated: CamelCase WikiWords
External Link	[http://example.org The title]	[http://example.com] or [http://example.com label]
Headlines	==Section== ===Subsection=== ====Sub-subsection====	-- Titlebar -- ! Level 1 !! Level 2 !!! Level 3 !!!! Level 4 !!!!! Level 5
Monospace Format	<tt>monospace</tt>	--monospace--
Strikethrough Format	<s>striktthrough</s>	--strike--
Superscript Format	^{superscript}	{SUP }text{SUP}
Subscript Format	_{subscript}	{SUB }text{SUB}
Images	[[image:wiki.png]]	{img src=http://foo.bar/foo.jpg}
Aligning Text	<center>Centered</center>	::centered text::
Text Indentation	: indented line	leading spaces, if activated.
Bulleted Lists	* Item 1 ** Item 1.2 * Item 2	* Item 1 ** Item 1.1 * Item 2
Numbered Lists	# Item 1 ## Item 1.2 # Item 2	# Item 1 ## Item 1.1 # Item 2
Definition Lists	; term : definition	;term:definition
Horizontal Rule	----	---
No wiki formatting	<nowiki>Text not wiki</nowiki>	~np~This "text" not ===formatted===~np~

Figure 11: Markup comparison between MediaWiki and TikiWiki

4. Context-Sensitive Wiki Help Design

4.1 Designing the Help User Interface

This chapter explains the design of context-sensitive help from the mockup stage to the implementation phase. Initially, there were many designs considered for implementing context-sensitive help into Yioop. The semi-transparent help mode used in the LemonAid Help System [2] was the initial choice. Clicking on the dedicated help button initializes context-sensitive wiki help. The help interface is an overlay over the existing interface on which users can click on to seek help. Clicking on a particular item will show a list of help articles pertaining to an element or the page in context. An example mockup showing this implementation is shown in Figure 12: Help-mode initialization and usage mockup.

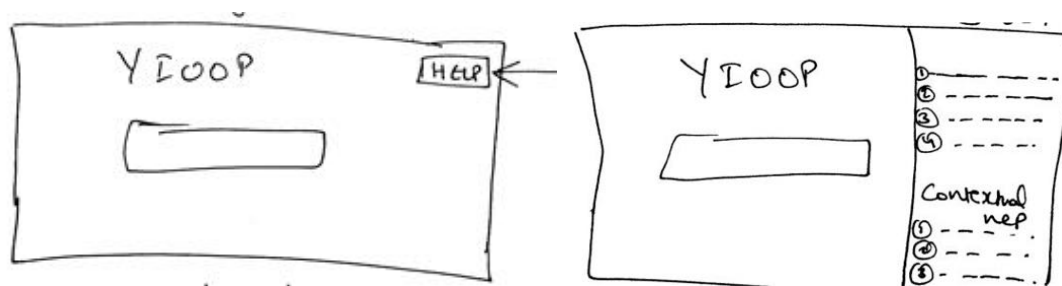


Figure 12: Help-mode initialization and usage mockup

The above mockups are several of the initial mockups drawn as a part of this project's deliverables. Since their creation, I have decided to implement a simpler help interface. The user shouldn't lose focus of what he or she is doing. The very definition of context-sensitive help calls for avoiding showing the user unneeded information, so displaying an entire list of help articles that the user may not need would not be as productive. Furthermore, implementing a more complicated UI to build a semi-

transparent help mode is out of the scope of this project. The main idea is to implement a simple interface where users can receive and contribute to context-sensitive help.

With the help from Dr. Chris Pollett, I have designed a simple user interface for Yioop's context-sensitive help. The user accesses this help by clicking the help button beside an element. Figure 13 shows the confirmed design.

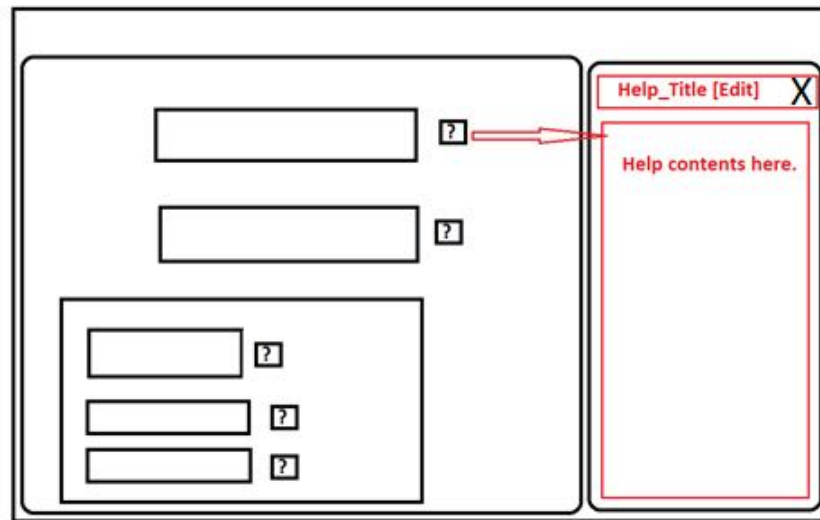


Figure 13: Context-sensitive help design in Yioop (left-to-right)

The help points, where applicable, are preconfigured to include a help button in the form of a question mark. When users click on the “?” button; the help section opens up. Help articles can be opened using help buttons beside multiple elements on the page. When a help article is open, users can use the [X] mark on the top right corner to close the article.

The help interface shown in Figure 13 looks good with languages written left-to-right, such as English. However, this design doesn't hold up as well for right-to-left written languages and mobile interfaces.

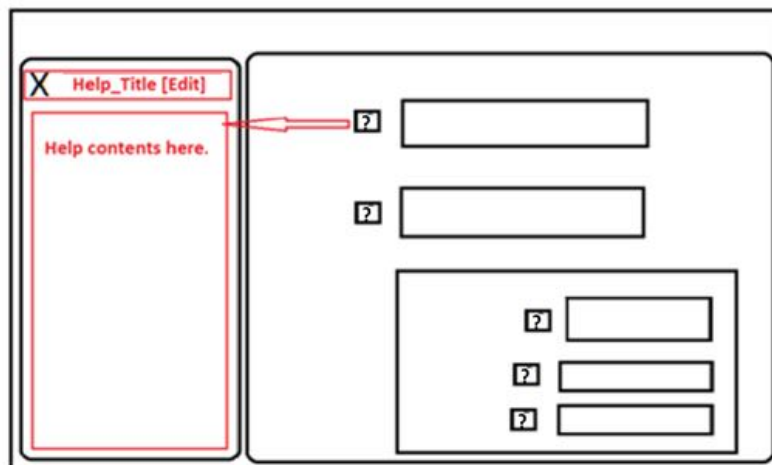


Figure 14: Context-sensitive help design in Yioop (right-to-left)

The interface in Figure 13 can be tweaked for right-to-left written languages as shown in Figure 14. Yioop also has a compact mobile interface that uses less screen space than its desktop counterpart.

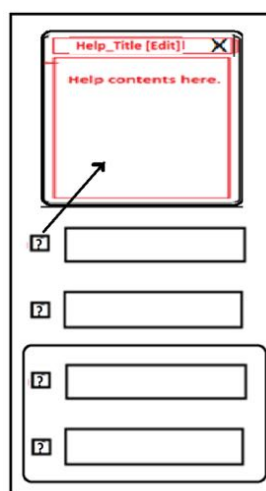


Figure 15: Context-sensitive help design for Mobile interface

Having help on the side panel may not be a good idea with a mobile interface. Thus, the help article placement is on the top of the page instead of on the side panel. You can see the mockup in Figure 15.

4.2 Designing the Storage and Retrieval of Help Articles

The next step is to design a way to store efficiently and retrieve help articles. Using Header files and Alias files, offered in Adobe RoboHelp [8] and Madcap Flare [7], for the retrieval and mapping of help articles was thought to be ideal. The initial idea was to use Header files and Alias files to map the help articles to help topics. After considering how the wiki system was made in Yioop, I used Yioop's wiki system to do exactly what the Alias and Header file combination does. The wiki system in Yioop already maintains wiki pages categorized into groups. Additionally, Yioop has the groups concept built in, which enable users of a group to create relevant wiki pages. For example, User A, who is a member of Group X, can create wiki pages in Group X. Wiki pages created in Group X are then accessible by other members of Group X.

In other words, Yioop has social groups and user access control already built into it. Every wiki page that is created in a group can be uniquely identified using its name – a numeric group ID.

Furthermore, Wiki pages in Yioop are written in Yioop wiki markup. Yioop wiki markup is similar to MediaWiki markup [9], although there are slight differences. Users may enter the wiki article in wiki markup, but he or she will always view the wiki page in HTML. Yioop talks to the database to get wiki markup contents for a page when the user clicks on the help button. Yioop then parses the wiki markup to convert to HTML and sends the page to the client-side for it to be rendered in the browser.

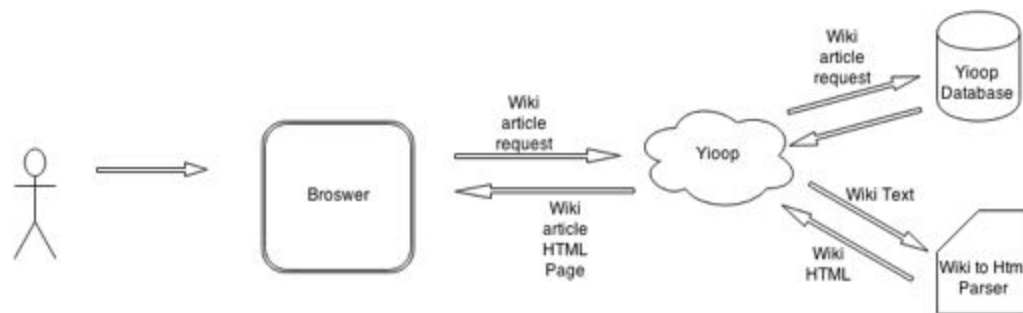


Figure 16: Retrieval of Wiki articles in Yioop

As you can see in **Error! Reference source not found.**, this is a synchronous process. This synchronous rendering of the wiki pages is not suitable for displaying help. A page that is reloaded every time a user clicks on the help button creates a lot of churns and will not help the user remain focused.

In order to implement a responsive and intuitive user help interface, I have decided to show the help articles asynchronously – that is, loading the help from the server on request. This also ensures that, when there are multiple help articles to be opened on any page, each page is rendered only upon user request.

In order to serve the wiki contents asynchronously, the best approach is to implement a web service that retrieves the wiki pages on demand. Also, to keep the pages thin and the web service calls inexpensive, plain wiki markup text is served by the web service instead of the parsed HTML content. Figure 17 shows a flow diagram of this implementation. This implementation makes the asynchronous HTTP calls faster as the time taken to parse the text to HTML is also conserved. However, In order to show HTML-based wiki pages, I have decided to move the parsing to the client side.

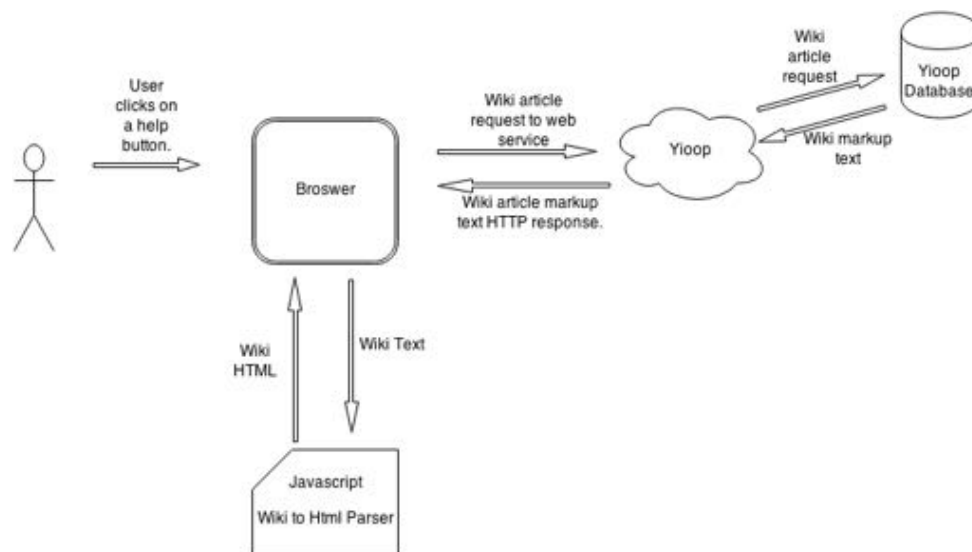


Figure 17: Asynchronous retrieval of help articles using a web service from Yioop

I have implemented a Yioop wiki markup to HTML parser using JavaScript. The idea is to retrieve the wiki markup content asynchronously and parse the content to convert it to HTML before displaying it to the user.

4.3 Editing the Help Articles

Now that I have a design for the retrieval and display of help articles, what follows is how to let users collaborate on help articles. As discussed in the help interface design section, there will be an “Edit” link beside each open article. Clicking on this link will take the user into the editor mode for the wiki page.

A wiki editor, built entirely in JavaScript, is integrated into the wiki edit page. This wiki editor is a WYSIWYG-style editor [11]. It also includes keyboard shortcuts to make use of what users already know.

The other important part of the wiki editor page is a “Back” button. When the user navigates from away from a web page that has an open help article in order to edit the

help article, he or she needs a way to get back to where they previously were. Once the editing is complete, users can navigate back to the web page to view the changes made. More about this is explained in the implementation section for the wiki editor in Chapter 5.

5. Implementation of Context-Sensitive Wiki Help

5.1 Implementing the Wiki Editor

The first part of the implementation I worked on was the wiki editor. I built a wiki editor from scratch using JavaScript. Yioop already has a wiki system built in, which was lacking a wiki editor. Even though Yioop's wiki markup is similar to MediaWiki, there are minor differences for formatting some elements like resources, external links, etc.

Wiki Functionality	Wiki Markup
Bold	" Bold text "
Italic	" <i>Italic text</i> "
Underlined text	<u>Underlined text</u>
Strike text	<s>Strike text</s>
Nonformatted text	<nowiki>Non-formatted text here</nowiki>
Links	[[http://google.com]] – External link [[Browse_Groups]]
List items	* Unordered list item # Ordered list item ; Item : Definition List
Justify Text	{{left This text is left-aligned.}} {{center This text is centered.}} {{right This text is right-aligned.}}

Wiki functionality	Wiki Markup
Horizontal line (<hr>)	----
Headings <h1> <h2> <h3> <h4>	=Level 1 Heading= ==Level 2 Heading== ===Level 3 Heading=== ====Level 4 Heading====
Table with two rows and two columns	{ - ! Table Title !! Table Title - Example Example - Example Example }
Search widget, small in size.	{{search:default size:small placeholder:Search Placeholder Text}}
Resources uploaded to the wiki page (same group)	((resource: resource_file_name.extension Resource Description))

Table 1 Yioop Wiki markup.

I have taken Yioop's wiki markup and created a wiki editor that generates wiki text when users click on corresponding buttons. Figure 18 shows the initial version of the wiki editor.



Figure 18: Initial version of wiki editor

This first version did not support full markup. However, I developed the editor to add more features and styling.



Figure 19: Fully implemented wiki editor

The completed version of the wiki editor looks like in Figure 19. The wiki editor is highly configurable, and there are multiple ways to integrate it into an existing HTML text area. Below is an example of the wiki editor integrated into the text area.

```
<script type="text/javascript" src="./scripts/wiki.js" ></script>
```

```
<textarea id="wiki-page" class="tall-text-area" name="page" data-buttons="all,!wikibtn-bold"></textarea>
```

This script adds buttons to HTML text areas on a page, and the editor automatically renders the editor buttons using the following list of button names for configuration. These can be set on the data-buttons attribute of the HTML text area. See Table 2 for a reference of all wiki buttons and their identifiers.

Wiki Syntax	Wiki button id
Bold	"wikibtn-bold"
Italics	"wikibtn-italic"
Underline	"wikibtn-underline"
Strike through	"wikibtn-strike"
No wiki formatting	"wikibtn-nowiki"
Internal or External Links	"wikibtn-hyperlink"
<u>Unordered lists</u>	"wikibtn-bullets"
Ordered lists	"wikibtn-numbers"
Horizontal separator	"wikibtn-hr"
Headings	"wikibtn-heading"
Yioop Search widget	"wikibtn-search"
Table	"wikibtn-table"
Presentation slide	"wikibtn-slide"
Definition lists	"wikibtn-definitionlist"
Left align text	"wikibtn-leftaligned"
Right align text	"wikibtn-rightaligned"
Center align text	"wikibtn-centraligned"

Table 2 Wiki button identifiers.

Additionally, I included the `wiki.js` source file, which has all source code to paint the wiki editor. Calling a JavaScript function `editorize("wiki-page");` will add a wiki editor to the textarea with a CSS identifier `"wiki-page"`.

Calling the javascript function `"editorizeAll();"` will add a wiki editor to all text areas on the page. The `data buttons` attribute takes in a comma separated list of button identifiers. The item `"all"` includes all buttons, whereas a button ID prefixed with `"!"` excludes that button.

For example: `"all,!wikibtn-bold"` will include all buttons except the `"Bold"` button.

There are also keyboard shortcuts implemented for basic use in the wiki editor. `"Ctrl+b"` formats the selected text to bold; `"Ctrl+u"` formats the selected text to underline; and `"Ctrl+I"` formats the selected text to Italics.

5.2 Implementing the Web Service to Retrieve Wiki Pages

This part of the implementation section explains how the web service is implemented and how the response is used to populate the wiki help article. Yioop already has an action named `"wiki"` implemented on multiple controllers. The action `"wiki"`, along with an argument, `read`, or `edit`, retrieves the wiki page and presents it to the user in the mode requested. For example,

`?c=admin&YIOOP_TOKEN={token}&group_id=2&arg=edit&a=wiki&page_name=Main`

Figure 20 Sample URL format for retrieving a Wiki page.

The above URL retrieves a wiki article matching the page name `"Main"` created in the group with `"group_id"` equal to `"2"`. The `"a"` stands for action, `"arg"` stands for the

argument - read or edit, and “c” stands for controller. The page retrieved using the URL shown in Figure 20 is an HTML page with wiki contents converted into HTML by the server side wiki parser.

I have created a web service completely isolated from the above controller. I named the new controller “api” . As its name suggests, any API or web service calls to Yioop can now use this controller. In order to group all help articles, I created a group in Yioop named “Help” . All wiki help articles are expected to be a part of this group. The group_id for the “Help” group will remain constant. Calling the “wiki” action with “read” argument will now bypass the wiki parsing and directly spit out wiki text in the HTTP response. In order to programmatically extract and parse wiki content using JavaScript, I have decided to produce the HTTP response in JSON [12] format.

Request

```
GET
/yioop/?c=api&group_id=3&arg=read&a=wiki&YIOOP_TOKEN=72zJt6l5BUU|1415863871&page_name=
Locale%20Writing%20Mode HTTP/1.1
Host: Yioop.com
Cache-Control: no-cache
```

Response

```
{
  "wiki_content": "=Heading for Help article=\nSample '''Wiki''' content in Yioop wiki markup.
This is a sample wiki help article. This shows an example help article with title \"Locale Writing Mode\".
<nowiki>This text, as you can see is not formatted by wiki.</nowiki>",
  "group_id": "3",
  "group_name": "Help",
  "page_id": "39",
  "page_name": "Locale_Writing_Mode",
  "page_title": "Locale Writing Mode"
}
```

Figure 21: HTTP GET call to retrieve a wiki article over the web service

This enables easy retrieval and parsing of the HTTP response using JavaScript. A sample response retrieved by the web service is shown in Figure 21. As you can see, the

wiki_content attribute, which is the wiki markup text, is only one of many attributes in the response. All other attributes carry meta information for the wiki article to be rendered properly. The attribute group_id, which is the numeric ID of the group that the wiki article belongs, is used to render resources like images and videos in the wiki article. Other attributes like page_name, page_id, etc. are used to render the help page properly.

5.3 Implementing the Wiki Parser and Help Display

5.3.1 Rendering the Help Button

The next part of the implementation is explaining how I applied the wiki parser and help display. Yioop already consists of a wiki parser on the server side. As described in the design section, Chapter 3, I wanted to optimize the asynchronous HTTP response received via the web service. The wiki markup text is then parsed using JavaScript on the client side into its HTML counterpart. Though it is a good idea to retrieve wiki content as markup text, specifically for use in this case, the actual wiki read pages – which are synchronous in nature – cannot follow this model. Retrieving the wiki pages as markup text would affect the SEO of the site as search engines might not be able to parse the markup text out of the box. Search engines are designed to parse and index HTML-based pages better than custom markup text.

The help buttons on a Yioop page are the starting point for displaying any context-sensitive help. A help button is placed beside any HTML element that may require help. A help button also needs to open the right context-sensitive help article when a user clicks on it, so each help button is tied to a wiki help page dedicated to a specific help topic.

For example, let’s take a look at the Manage Locales page on Yioop. Figure 22 shows the form for adding a Locale in Yioop.




Figure 22: Form for adding a Locale

The Manage Locales page is used to add new Locales to Yioop or manage existing Locales. The first part of the Manage Locales page is the Add Locale form. As you can see in Figure 22, I have placed a help button beside the Select Mode dropdown.

```
<?php
e(
$this->view->helper("helpbutton")-
>render("Locale_Writing_Mode",$data[CSRF_TOKEN])
); ?>
```

Table 3 PHP Code snippet for rendering a Context-Sensitive Wiki Help button.

A view helper class [13] named “helpbutton” is created to generate help buttons. There are two parameters passed to the render function of the “helpbutton” view helper. “Locale Writing Mode” is a help topic, or in other words, the name of the help wiki page. The second parameter is a CSRF token that Yioop uses for all HTTP requests to check for CSRF attacks. In the above snippet, e(); is a global function that prints the parameter passed.

A Yioop administrator is required to insert help buttons beside the elements that need help. Automating this procedure or enabling users to place context-sensitive help buttons on the web pages is out of the scope of this project but may be considered for future improvements.

When the parameters are passed to the view helper, the final result is an HTML button that opens the help. The final HTML button is shown in Figure 23.

```
<button type="button" class="help-button default" data-tl="{wiki_view_edit :
"Edit",wiki_view_read : "Read"}" data-back-params="" onclick="javascript:
displayHelpForId(this,false,'admin','manageLocales','YI00P_TOKEN','NjvcHg1-
bi8|1416956504','3','api','wiki','read')" data-pagename="Locale Writing Mode">
?</button>
```

Figure 23 HTML button for Context-Sensitive Wiki Help.

As you can see, there are multiple HTML attributes attached to the help button. See Table 4 for the description of each of the attributes.

Attribute name	Description
type	Denotes the type of HTML element, which is always a button.
class	Denotes the class, which helps render the styling for the HTML button.
data-tl	A custom HTML5 data attribute used to pass the translated text used in the HELP user interface; for example, “Edit” is used to edit text in English.
data-back-params	Holds the request parameters to render the back button on the editor page. In other

Attribute name	Description
	words, these parameters are used to remember which page the user is currently on in order to get back to it via the back button.
onclick	Responsible for calling the main JavaScript function that opens up help. There are multiple parameters passed to this function to render help based on the current interface the user is viewing.
data-pagename	Page name of the help article to retrieve asynchronously.

Table 4 Attributes used in the Help button.

5.3.2 Wiki Page Retrieval and Help Display

As you can see from section 4.3.1, the “onclick” attribute triggers the function call to the JavaScript function “displayHelpForId”. The “displayHelpForId” talks to various other JavaScript functions to perform multiple actions which result in the correct rendering of the help display.

- Initiating the AJAX HTTP call to fetch the response from the wiki page web service.
- Parse the JSON response to separate out the wiki markup text.
- Pass the wiki markup text to the wiki parser to get converted HTML content.
- Render the side or top pane to display help content with dynamic styling changes.

As I discussed how the JSON response is retrieved from the web service, I will explain how the JavaScript wiki parser works. The wiki markup to HTML conversion is

implemented in a JavaScript function called “parseWikiContent.” The wiki parser function uses regular expressions to parse the plaintext out of wiki markup characters to produce resultant HTML. The main tool used for parsing the markup characters is regular expressions in JavaScript [14]. JavaScript supports powerful regular expressions to extract text contents. Applying regular expressions on an entire wiki markup text returns all possible matches of text wrapped inside the markup characters requested. This enables the extraction of the contents of markup text and the ability to apply HTML styling. Let’s consider an example of extracting HTML for Bold text.

Example for bold text in Yioop markup: *"**Bold text**"*

We need to extract the bold text and apply HTML tags ``

Regular expression applied: `/'"(.*)"'/g`

The regular expression above matches 3 single quotes literally, followed by the capture group and another 3 single quotes matched literally. Code snippets below apply the regular expression and return the matches to a callback function, which we can then use to apply HTML styling.

The below code snippet is a small subset of how parsing is done. All syntax types are implemented the same way using regular expressions for each. Ordered and unordered lists are done the same way but recursively.

```
//Regex replace for bold characters
html = html.replace(/'(.*)'/g, function (match, contents)
{
    var parsed_content = "<b>";
    parsed_content += contents + "</b>";
});

//Regex replace for external links
html = html.replace(/\[\[\](http.*)[!]\]/g, function (m, all)
{
    var matches_array= all.replace(/\[\[\]/g, "").split(/\|/);
    var hyperlink = matches_array.shift();
    var parsed_content = '<a href="' + hyperlink + '"'
        + (p.length ? matches_array.join("|") : hyperlink) + '</a>';
    return parsed_content;
});

//Regex replace for headings
html = html.replace(/(?:^|\n)([=]+)(.*)1/g,
function (match, contents, t)
{
    var parsed_content = "<h" + contents.length + ">" + t + "</h" + contents.length +
        ">";
    return parsed_content;
});
```

Help is displayed once the HTML contents are returned by the parser. In order to make room for the help to be displayed, there are some CSS styling adjustments done dynamically to the user interface. If the user is viewing a help article on a desktop browser, a side pane opens up from the right side for right-to-left languages, while the help pane opens from the left side for left-to-right languages.

Figure 24 shows a normal web page on Yioop.com with no help articles open. As you can see, there is a help button beside the “Browse” hyperlink. There are 2 sections visible on the page: a side menu pane on the left of the page with links to various other

pages on Yioop, and an HTML container with the main activity, “Create/Join Group”, located at the center.

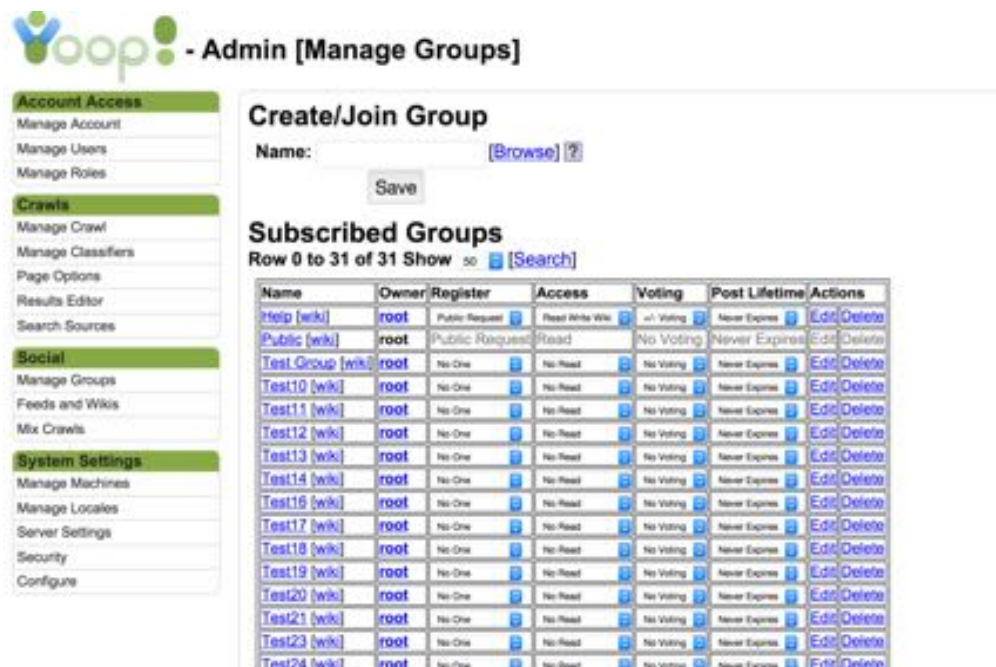


Figure 24: A web page on Yioop with no help article open

Clicking on the help button beside the “Browse” hyperlink opens up the help pane from the right side. The help pane is intended to be opened on the side of the main activity container. In order to maintain a nicer user interface without the need of scrolling when the help opens up, I have implemented the help pane to fit the existing screen real estate. The “Main Activity” container in the center shrinks somewhat to make room for the help pane to expand as shown in Figure 25.

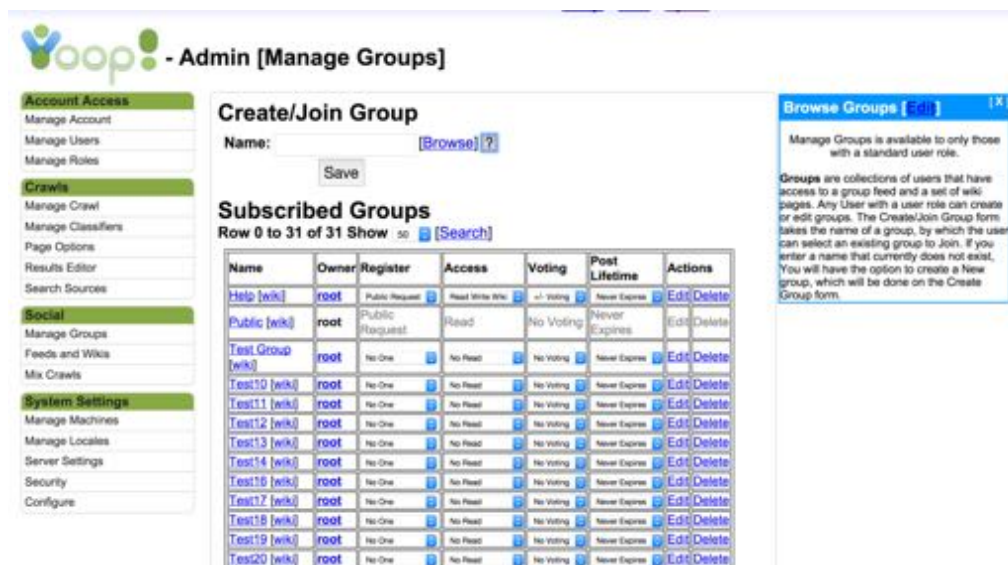


Figure 25: A web page on Yioop with English Locale selected and help article open

For languages written from right-to-left, the side panel opens to the left as shown in Figure 26.

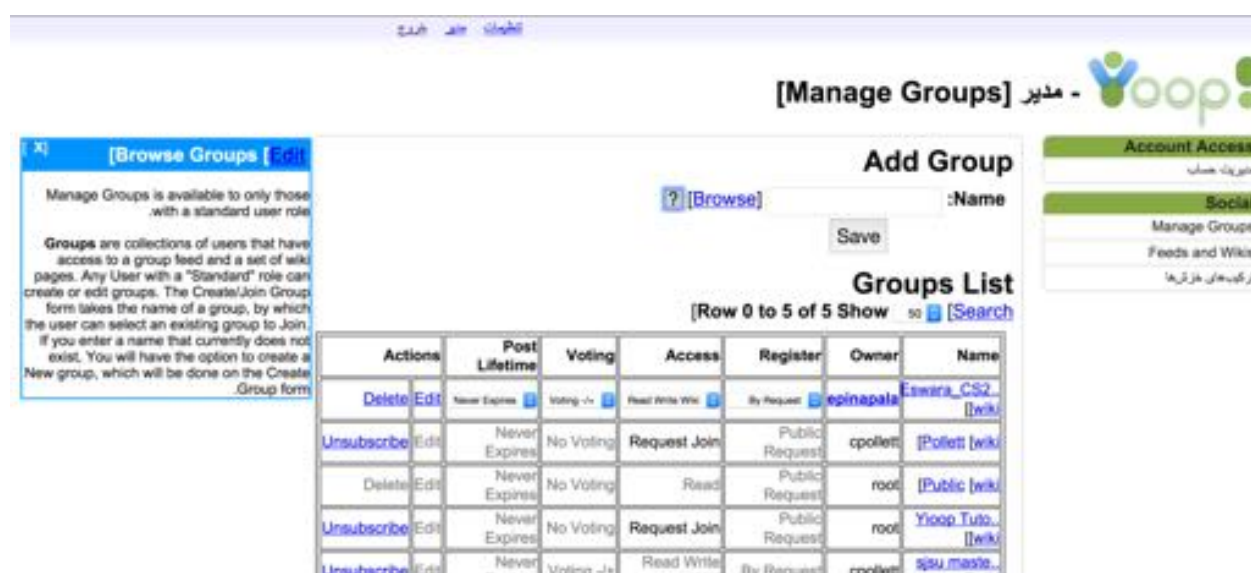


Figure 26: A web page on Yioop.com with Persian Locale selected and help article open

Help is implemented for mobile web browsers in a slightly different way. Due to constrained screen space, mobile interfaces have the help screen sliding from the top

rather than the side. This holds good for any locale selected as the position of the help panel remains on the top irrespective to the way the language is written. Figure 27 shows how help is implemented on a mobile interface.

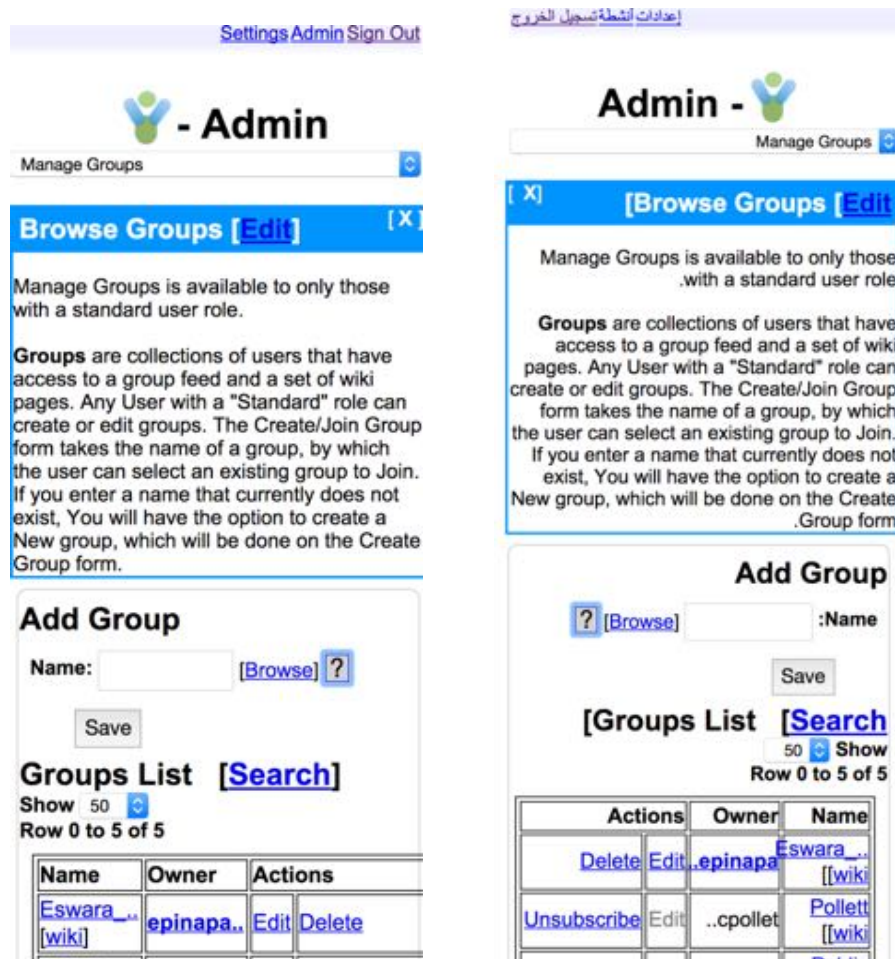


Figure 27: Help opened on a Mobile web browser (English on the left and Arabic on the right)

5.4 Implementing Upgrade in Yioop

In order to include the Context-Sensitive Wiki Help in Yioop, I had to create a Group named "Help" in Yioop. For better code integrity, I had to make the numeric ID of the group constant, which is stored as a Global constant in Yioop's global configuration file- "config.php". The variable used for the Help group is "HELP_GROUP_ID" and the value assigned to this

variable is “3” . So, for any new Yioop installations, a new help group will be created with ID equal to 3. All the Wiki Help articles are inserted into this group, and help buttons are created at various places in the Yioop web pages so that the help articles can be referenced.

```
/** ID of the group to which all Yioop Help Wiki articles belong */  
define('HELP_GROUP_ID', 3);
```

While this flow works well with new installations, there is an apparent problem with setting the “HELP_GROUP_ID” to 3 in existing Yioop installations. An Existing Yioop installation might most likely already have a group with an ID equal to 3. Before we created context sensitive Wiki Help in Yioop, There was only one group –“Public” with a group ID equal to 2. Any new group that is created will be an increment of “2”. So in order to make room for our Help group in Yioop, I have decided to check for a group with an ID equal to 3; if one exists, I would move the group to the end of the groups list, that is, assign an ID that is the highest number assigned to the groups, and add the Help group at group ID -3. This ensures that we don’t have any group bearing a group ID equal to 3, now I can add a new group named Help with a group ID equal to 3. Figure 28 shows the entire upgrade flow for New and existing installations.

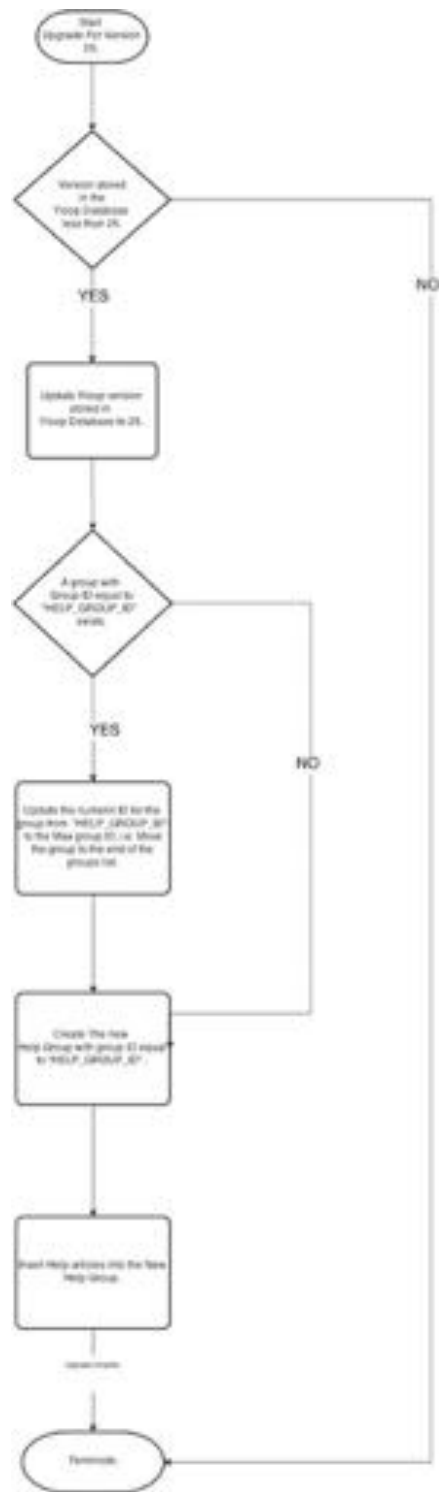


Figure 28 Yioop Database upgrade for Context-Sensitive Help addition.

6. Experimenting on Context-Sensitive Wiki Help in Yioop.

6.1 User-Assisted Testing

In this chapter, I will discuss the results of user-assisted testing of the context-sensitive wiki help. I have selected five users in total who have no prior knowledge of the Yioop system. There were two rounds of testing, and all results were recorded onto individual videos for later reference. Video recording helped us assess how hard or easy it was for users to navigate Yioop. The users on Yioop to test and provide feedback over 2 sessions spent a total time of 350 minutes. That's an average of 40 minutes per session.

The first round of testing was done with minimal to no help articles inserted into Yioop. Users had no idea about Yioop and were asked to navigate freely and perform whatever tasks possible by the stretch of their knowledge on the Yioop website. Depending on prior knowledge of other similar systems, a lesser percentage of the users were able to perform some known actions like managing groups, managing users, and managing locales. The rest of the users spent a lot of time navigating and trying to understand the system by trial-and-error. They were also reaching out to me and asking for help. The first round of testing was intended to get feedback about the Yioop user interface and navigation in general.

The first round of testing also resulted in some feedback about how the help articles should be written. The Context-Sensitive wiki help articles are initially created by the Website administrator. Though the articles can be edited by users later, It is important to keep the articles short and brief to make the users not ignore the help. During this round of testing, there were minimal help articles on a couple of pages, both of which

were pretty descriptive in nature. Users are not interested in reading long descriptive help articles as shown in Figure 29. They actually ignored the help articles when they were long.

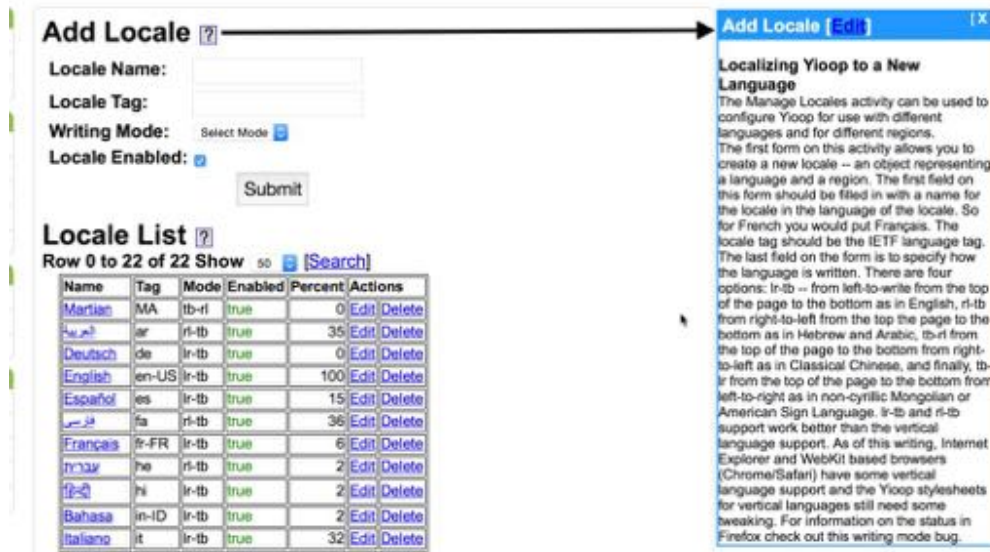


Figure 29 Long and descriptive help article.

Users also complained about not knowing definitions of certain terms on the Yioop site. Some of the items in the wiki help article could either define the technical terms or point to a web location which defines them. This was noted as a part of this round of testing as well.

With the feedback gained during round one, I made changes to the user interface and assisted users in round two by adding help at a number of places on Yioop. A few help articles are also refined to make sure the users do not ignore the help. For example, the help article shown in Figure 29 was very long and descriptive. The help article is now split into multiple help articles pertaining to multiple places on the page. One help article is now split into 3 different small help articles that are more context-sensitive as shown in

Figure 30. This also gives the users a chance to customize the help articles whose scope was small.

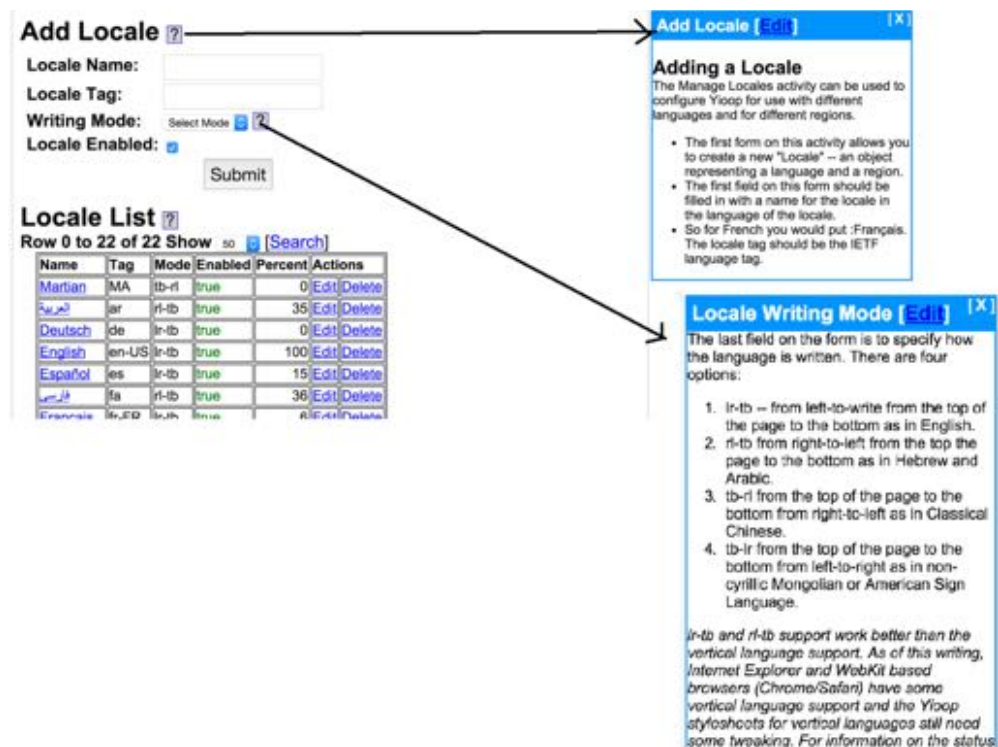


Figure 30 Context-Sensitive help for Add Locale after feedback from users.

Some help articles were task-oriented, while others were descriptive in nature. In order to prove the effectiveness of collaborative context-sensitive wiki help, users were also encouraged to edit the help articles if they felt modifications were necessary. There were some intentional and unintentional typos as well as wrong sentence formations in the help articles, which encouraged the users to edit the help articles. Almost all users tried to edit and simplify the help articles. Most importantly, help articles edited by the users were helpful to other users who followed them in testing. Users also felt that collaborative context-sensitive help was indeed a plus. Surprisingly, users found errors in wiki articles which we didn't know about. The feedback

was ultimately very helpful. Not only did the users help to improve the context-sensitive wiki help, but they also helped improve the user interface of Yioop.

Finally, Users also requested for a way to understand and reach parts of the wiki article while glancing at the articles. I have Highlighted important parts of the article and highlighted them to bold text so that the users were able to grasp the help text quickly.

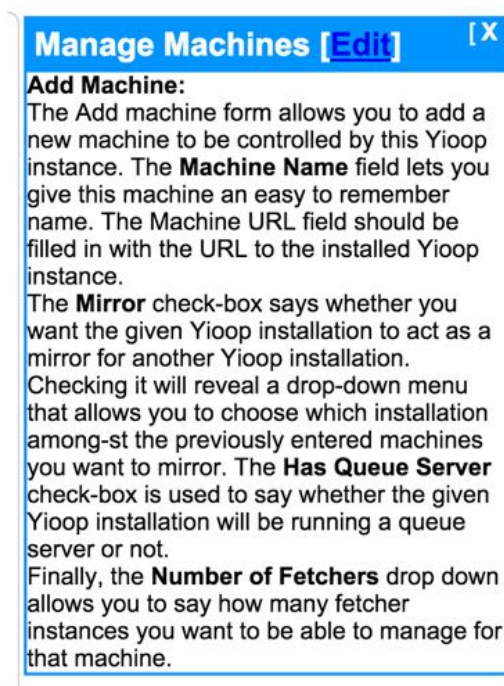


Figure 31 shows a Task-based help article with highlighted terms.

Figure 31 shows an example of Context-Sensitive help article with terms highlighted. The users might already know the task they can perform on the page partially, highlighting the main parts of the page which lets the users perform tasks has enabled the users to glance at the help article and consume only what they need.

Procedural Help was also implemented in addition to screen based or definition based help. Only Procedural help doesn't seem to please all kinds of users, hence the main title always

points to Procedural help , while individual elements on the page tend to define more streamlined definition based help. This can be seen in Figure 32.

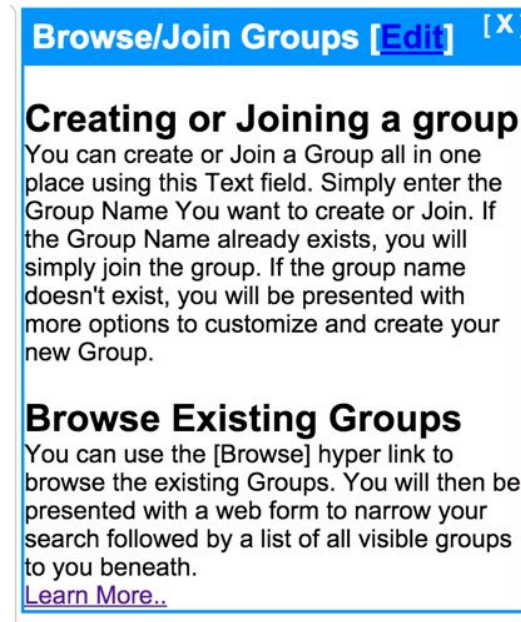


Figure 32 Procedural Help for Creating or Joining a Group.

A “Learn More” hyper link is an additional link to reference documentation, If the user wish to learn more by going away from the task he is performing. A very low percentage of users were interested in reading reference documentation anyways.

6.2 UI Testing Using PhantomJS

Yioop already has a suite of tests written for JavaScript and PHP unit testing. However, Yioop lacked a UI testing framework for running tests in an actual browser. UI testing is important to figure out issues with CSS styling, DOM manipulation using JavaScript, etc. I have done a bit of research on running UI tests. The initial choice to run UI tests was the popular Selenium Framework [15]. While Selenium tests worked fine paired with PHPUnit [16], there were two major problems: the Selenium stack is expensive and needs a Selenium server to run

continuously, and Selenium has a complicated setup. Users who spin off a Yioop instance on their own thin servers do not expect to install the entire stack required to run Selenium tests.

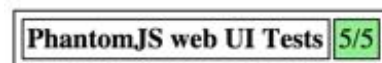
PHPUnit is not a part of Yioop's testing framework, thus working with selenium without PHPUnit meant integration with Yioop's extended PHP unit testing framework.

SeekQuarry Tests

[See test case list.](#)

PhantomjsUiTest

UI test results - web



UI test results - mobile

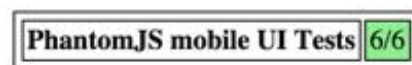


Figure 33 Shows PhantomJS UI Test results.

I dropped the idea of using Selenium and researched a less dubious alternative.

PhantomJS [17] – a scriptable, headless web browser – was an amazing find for what we were trying to achieve. It does not need a Graphical User Interface (GUI) to run and runs all tests silently. Figure 33 shows how the test results look like after running the tests using PhantomJS. PhantomJS is written in JavaScript, and all tests can be written in JavaScript as well. However, the biggest advantage of using JavaScript is that other JavaScript-based tests can also be run using PhantomJS.

7. Conclusion

This project aims to prove that context-sensitive Wiki Help that is context-oriented and collaborative is indeed useful when compared to traditional context-sensitive help systems. In the first round of testing, the users were asked not to edit any help articles, but were apparently troubled with erroneous help articles. This proves that implementing context sensitive help alone will be good only if the articles are written in a perfect manner that is complete and with no errors. Also, when something changes, we, the administrators, have to make sure that the context-sensitive help articles are up-to-date with the changes.

However a context-sensitive Wiki Help, if collaboration is provided with the right access levels, will help users more. Users who might have already completed certain tasks on the page, or who have more knowledge of the system and its changes, can contribute to the Wiki Help articles to make them better for other users. This ensures lesser intervention of the web administrators in editing or publishing help articles.

Another area of exploration was how to design the information setup in the Context-Sensitive Help articles. The feedback from the users helped us refine the help articles and design user-friendly help articles.

The scope of this project is to experiment by building a context-sensitive help that is collaborative in nature and test it on users to see if it is indeed helpful. There were a lot of UI tweaks implemented to make the Yioop User interface better. It was astonishing how minor tweaks in UI make a lot of difference in user navigation. This also helped us understand that we can avoid placing help on certain parts of the web pages, if the User interface is transparent and

helps users to find what they want quickly. The scent of information does matter for the users in finding what they are looking for, and that even includes finding the help articles.

7.1 Future Work

There are a few enhancements that could be included in the context-sensitive Wiki Help system in Yioop. A significant implementation is to let users insert help buttons wherever they want, however this might require a more complicated help framework over Yioop. Another enhancement to the existing context-sensitive Wiki Help is in-place editing without being redirected to the Edit page. Issues we encountered with CSRF token mechanism built into Yioop have forced us to use redirection to the Edit page for modifying help articles, instead of in-place editing.

There could be some automation implemented to synchronize the context-sensitive Wiki Help articles with the Yioop documentation [4] found on seek quarry. Moving the entire Yioop documentation, split into Wiki Help articles in the help group, can highly accelerate this process. This also is an excellent way of achieving clean, up-to-date documentation from user collaborations. However, this requires some amount of experimentation to make sure the help articles are clearly organized to form an index of help documentation.

References

- [1] Hastie, M. Johnston, and P. Ehlen. (2002.) Context-sensitive Help for Multimodal Dialogue. In Proc. of ICMI
- [2] Chilana, P. K. (2012.). LemonAid: Selection-Based Crowdsourced Contextual Help for Web Applications.
- [3] *Wikipedia: Model-view-adapter*. Retrieved on October 14 2014, from <http://en.wikipedia.org/wiki/Model-view-adapter>.
- [4] Chris, Pollett. (2014.). *Seekquarry - Yioop Documentations*. Retrieved on November 15 2014, from <http://seekquarry.com/?c=static&p=Documentation>
- [5] *"Five Steps to MadCap Flare" in Fiddlehead Publications*. (2014). MadCap Software.
- [6] Ellison, M. (2009.). User-centred Design of Context-sensitive Help - UA Europe.
- [7] *Using Flare to Build Field-Level Context-Sensitive Help*. (MadCap Software) Retrieved on October 12 2104, from <http://www.madcapsoftware.com/blog/2012/01/09/using-flare-to-build-field-level-context-sensitive-help/>
- [8] *Adobe Robohelp context-sensitive help*. (2014). (Adobe) Retrieved on September 28 2014 from http://help.adobe.com/en_US/robohelp/robohtml/WS5b3ccc516d4fbf351e63e3d11aff59c571-8000.html
- [9] *MediaWiki Formatting*. Retrieved on August 15 2014, from <http://www.mediawiki.org/wiki/Help:Formatting>
- [10] *Comparison of the usage of MediaWiki vs. Tiki Wiki CMS Groupware for websites*. Retrieved on September 28 2014, from <http://w3techs.com/technologies/comparison/cm-mediawiki,cm-tikiwiki>
- [11] *WYSIWYG "What You See Is What You Get"*. Retrieved November 5 2014, from <http://en.wikipedia.org/wiki/WYSIWYG>
- [12] The JavaScript Object Notation (JSON) Data Interchange Format. Internet Engineering Task Force (IETF) . Retrieved on October 8 2014, from <http://tools.ietf.org/html/rfc7159>
- [13] *View Helper classes in Zend*. Retrieved November 23 2014, from <http://framework.zend.com/manual/1.11/en/zend.view.helpers.html>

- [14] *Regular Expressions in JavaScript*. (Mozilla Developer Network) Retrieved on April 5 2014, from
https://developer.mozilla.org/enUS/docs/Web/JavaScript/Guide/Regular_Expressions
- [15] *Selenium Documentation*. Retrieved on November 20 2014, from
<http://www.seleniumhq.org/docs/>
- [16] *PHPUnit*. Retrieved on November 13 2014, from
<https://phpunit.de/manual/current/en/phpunit-book.html>
- [17] *PhantomJS, A scriptable, head-less Webkit browser*. Retrieved on October 17 2014, from
<http://phantomjs.org/documentation/>
- [18] Daniel J. Barrett. (2008). In *MediaWiki*. O'Reilly Media.
- [19] *Wiki Parser for Yioop Markup*. Retrieved on October 17 2014, from
<http://www.cs.sjsu.edu/faculty/pollett/masters/Semesters/Spring14/eswara/index.shtml?cs298/deliverable3/index1.html>
- [20] Kevin R. Parker. (2007). Wiki as a Teaching Tool. *Interdisciplinary Journal of E-Learning and Learning Objects*, 3(1), 57-72. INFORM.