HANDLING RELATIONSHIPS IN A WIKI SYSTEM

CS 297 Report

Presented to

Dr. Chris Pollett

Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Class

CS 297

By

Yashi Kamboj

May 2016

ABSTRACT

In this project, ways to add relationship handling to the Yioop wiki system are considered. Articles were reviewed on Wikipedia classification. After an initial understanding of how Wikipedia articles are organized and classified, a presentation on the basics of Wikipedia classification was prepared. Then, to understand Yioop, an open source search and content management platform developed and managed by Dr. Chris Pollett and to get a basic hands-on experience working with Yioop code, a bug was resolved in Yioop. It was faced while adding multiple users to a group. Later in the semester to progress slowly towards the final goal, links between Yioop Wikimedia pages were fetched and their entry was made into database. To better understand the type of relationship, both 'IS A' and more complicated relationships will be taken into consideration. Thus, this report aims at detailing the deliverables done in part A of the master project, thus, depicting proficiency gained to start working on Part B of the master project.

TABLE OF CONTENTS

INTRODUCTION

This project is an approach to handling relationships in a Wiki System. Yioop is an open source search engine developed and managed by Dr. Christopher Pollett. This project is a small attempt to enhance Yioop's data retrieval capability by providing users related pages to their searches.

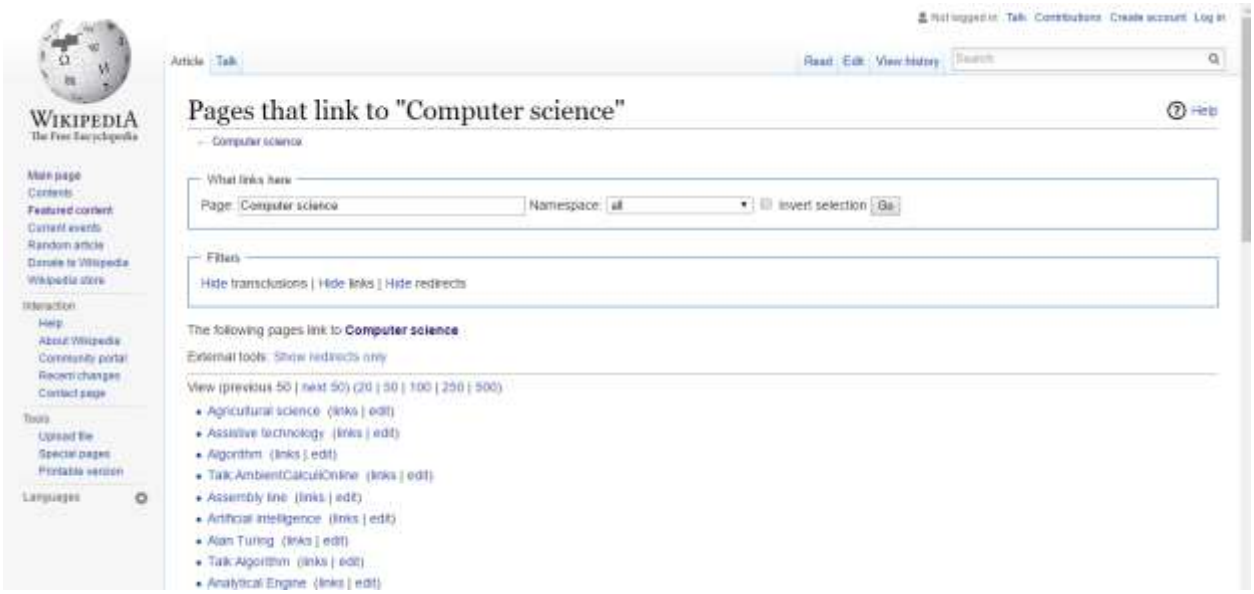
The motivation for this project is to set up a systematic approach to fetching pages by the search engine. There is a vast expanse of pages that the search engine has to index. It is efficient and effective for search engine to provide related results to the user on the basis of query fired.

Yioop crawls pages and stores internal references into its database along with the page IDs. For implementing relationships in Yioop, we need to fetch what all pages does the present page link to and then understand the type of relationship that they have. Once all this data is stored, the search can be made more effective for the user by presenting the linked pages.

I will be working on Handling Relationships in Yioop pages under the guidance of Dr. Pollett. With its implementation, Yioop will be more efficient to answer user queries as well as present the pages that link to those pages for effective search and information retrieval. Handling relationships has an obvious application in Genealogy. In future, another direction of work would include using this knowledge of handling relationships to implement Genealogy using Wikipedia.

To understand the present direction and scope of work, let us take an example. Consider Wikipedia page of Computer Science. It has the link: 'What links here' towards the left side of the page:

On clicking this link, we have a list of pages that link to the Wikipedia page of Computer Science as shown below:



At the end of Wikipedia pages, there is a section named ‘See Also’ which contains links to other Wikipedia pages. These pages are, somehow, related to the search page.

Thus, implementing similar functionality in Yioop where Yioop can handle relationships between the different pages and present them for efficient information retrieval by the user.

The following are the deliverables that I have done to understand the code structure of Yioop and making some enhancements to its functionality. I will now discuss the organization of the rest of the report. Next section describes the contents of my first deliverable. I had created a presentation covering basics of categorization in Wikipedia and Wikimedia documents. This deliverable helped me understand how documents are organized by Wikipedia. In Section 2 of the deliverables, I fixed a small issue in Yioop faced while adding multiple users to a social group. This deliverable is an effort to fix this bug and let user activate/ban/delete/reinstate multiple users without the hassle of traversing the list multiple times to reach the exact page of users. More details on this would be discussed under the section titled 'Deliverable 2.' In the next deliverable i.e. Deliverable 3, I started off with the path for actual master project implementation. Here, I worked on mapping parent page to the child pages (child referring to the pages mentioned in the parent page in the form of URLs) and saving them to a database. This was implemented by traversing Wikimedia pages and extracting the links present on the page using regular expression. The IDs of these pages were searched in the database and then stored with the relationship value along with parent and child ID. Section 3 focusses on handling parent and child links but not so much on the type of relationship that they share. The next section, Section 4 focusses on implementing mechanism for pages

to have relationship. This section will be completed during summer and then updated in the report and on professor's site as well.

DELIVERABLE 1

In order to move forward with the goal of implementing relationships in a wiki system, it is imperative to understand how pages are organized in a Wikipedia system.

ABOUT WIKIPEDIA

With the development of World Wide Web, there were several attempts to develop internet encyclopedia projects. It was launched on January 15, 2001. Unlike printed encyclopedias, Wikipedia is continually created and updated, with articles on historic events appearing within minutes, rather than months or years. Because everybody can help improve it, Wikipedia has become more comprehensive than any other encyclopedia.

The Wikipedia project has grown rapidly in the course of its life, at several levels. Content has grown organically through the addition of new articles, new wikis have been added in English and non-English languages, and entire new projects replicating these growth methods in other related areas (news, quotations, reference books and so on) have been founded as well. The organization of such vast expanse of data is a challenge in itself. Then I explored how Wikipedia manages its colossal amount of data.

On 30 May 2004, the first instances of "categorization" entries appeared. Category schemes, like Recent Changes and Edit This Page, had existed from the founding of Wikipedia. The categorization effort centered on individual categorization entries in each article of the encyclopedia, as part of a larger automatic categorization of the articles of the encyclopedia.

The central goal of the category system is to provide navigational links to all Wikipedia pages in a hierarchy of categories which are the defining characteristics of a topic. Categories enable users to browse sets of related pages. Users can browse and quickly find sets of pages on topics that are defined by those characteristics.

NEED FOR CATEGORIZATION

The central goal of the category system is to provide navigational links to all Wikipedia pages in a hierarchy of categories (characteristics of a topic). Categories are normally found at the bottom of an article page. Clicking a category name brings up a category page listing the articles (or other pages) that have been added to that particular category. There may also be a section listing the subcategories of that category. The subcategorization feature makes it possible to organize categories into tree-like structures to aid navigation.

HOW IS CATEGORIZATION IMPLEMENTED

A 'category page' is any page in the Category namespace. They each act as a category, and are termed a category. The category page has one section titled Subcategories listing other categories, and one section titled Pages, listing pages as categorized (in other namespaces). New categories are created by creating a page in the Category namespace. A category page displays at the bottom a generated list of all pages in that category, in the form of links. Other category pages which appear in this list are treated separately, as subcategories.

The page Special: CategoryTree enables you to see the tree structure of a category (its subcategories, their subcategories and so on.) The basic syntax is: <category tree> Category name </category tree>, to display just the category tree & <category tree mode=pages> Category name </category tree>, to display member pages as well.

MEDIAWIKI AND ITS SYNTAX FOR LINKS

MediaWiki is a free and open source wiki application. The MediaWiki software maintains tables of categories, to which any editable page can be added. To add a page to a category, include

"[[Category:Category name]] " or "[[Category:Category name|Sortkey]] " in that page's wikimarkup. A category is usually associated with a category page in the "Category:" namespace.

MediaWiki uses an extensible lightweight wiki markup language designed to be easier to use, thus, it has its own syntax using which the documents are created. An example is as shown below. This is followed by syntax examples.

MediaWiki syntax	Equivalent HTML	Rendered output

`==== A dialogue ====`
 	
`"Take some more [[tea]]," the March Hare said to Alice, very earnestly.`
 	
`"I've had nothing yet," Alice replied in an offended tone: "so I can't take more."`
 	
`"You mean you can't take ''less'',"  said the Hatter: "It's '''very''' easy to take ''more'' than nothing."`
	`<h4><span class="mw-headline" id="A_dialogue">A dialogue</span></h4>`
	
	`<p>"Take some more <a href="/wiki/Tea" title="Tea">tea</a>," the March Hare said to Alice, very earnestly.</p>`
	
	`<p>"I've had nothing yet," Alice replied in an offended tone: "so I can't take more."</p>`
	
	`<p>"You mean you can't take <i>less</i>," said the Hatter: "It's <b>very</b> easy to take <i>more</i> than nothing."</p>`
		A dialogue
		"Take some more tea," the March Hare said to Alice, very earnestly.
		"I've had nothing yet," Alice replied in an offended tone: "so I can't take more."
		"You mean you can't take *less*," said the Hatter: "It's **very** easy to take *more* than nothing."

Description	You type	You get

Internal link	`[[Main Page]]`
	`[[Help:Contents]]`
	`[[Extension:DynamicPageList (Wikimedia)]]`
		Main Page
		Help Contents
		Extension:DynamicPageList (Wikimedia)

Piped link	`[[Main Page|different text]]` `[[Main Page#Concrete_Paragraph|different text2]]`	different text different text2

The syntax examples shown above will be helpful in deriving regular expressions used to identify presence of links in Wikimedia documents, to be used in Deliverable 3.

DELIVERABLE 2

Groups are collections of users that have access to a group feed and a set of wiki pages. Groups are managed through the Manage Groups activity which looks like:

Unlike Manage Users and Manage Roles, the Manage Group activity belongs to the standard User role, allowing any user to create and manage groups. If either anyone can join the group or the group can be joined by request, then that group will be added to the list of subscribed to groups. If membership is by request, then initially in the list of groups it will show up with access Request Join.

In case of requesting access to a group, there are scenarios when group owner needs to add multiple users to a group at a single time, e.g.: All students requesting to be a part of a class group created by Dr. Pollett.

DESCRIPTION OF THE ISSUE

There was a small issue in Yioop related to adding multiple users to a group at a time. The details of the issue are as follows:

1. There are 10 users requesting to join a group that are displayed at a time.

2. In case of more than 10 requests, the admin has to click on ">>" to view the next 10 requests and so on.

3. While approving/denying the join request of nth request (n>10), the admin is brought back to the first set of users i.e. first 10 users (1<=n<=10).

4. Basically, there is a small bug which has to be resolved for the admin to stay on the same set of users, where he/she is presently working and not being brought to the first page of users.

5. This saves time and efforts while adding multiple users at a time.

6. To sum up, while adding large number of users to a group, if a user is activated/ deleted/ banned/ reinstated in the group, then it brings the admin back to the beginning of the list of users in the group. For instance, there are 21 users in the group and the admin adds user 16 to the group, then upon Activating user 16, the admin is brought back to the beginning of the list of users in that group. The issue here is that the admin has to traverse the list sequentially to reach to user 16. This can be problematic if there are very large number of users, say 50 or 100.

STEPS TO REPRODUCE THE ISSUE

The following steps would assist in reproducing the issue:

1. Access Yioop homepage as an admin.

2. Go to Social->Manage Groups->Click the group in which you got multiple user requests (User requests can also be created by making multiple user accounts and requesting access to the particular group, say "TestGroup.")

3. Click Edit under Actions and then click on the link which displays the number of user requests under Members.

4. Click on the link. It displays the number of users, requesting access to the group.

5. Scroll next pages of the list using ">>". Choose one of the users and activate the user there by clicking Activate.

6. The admin would be brought back to the beginning of the list.

RESULTS

The issue is resolved by passing the parameter which states where exactly in the list of users are we presently. While performing any action (activated / deleted / banned / reinstated) on a particular user requesting access, if its position in the list is saved to a parameter which is passed to a method where the actual action is performed then the issue gets resolved.

The argument passed is "group_limit" and another parameter taken into consideration is "NUM_RESULTS_PER_PAGE", e.g.: 10 users per page.

Step 1:

In the first step, we navigate to ‘Social Component’ controller of Yioop code and edit the switch case where the request to activate/ ban/ delete/ reinstate is being made. Pass on the parameter ‘group_limit’, which captures the position in the list of the users requesting access as shown in the figure:

```
switch ($_REQUEST['arg']) {
    case "activateuser":
        $_REQUEST['arg'] = "editgroup";
        $user_id = (isset($_REQUEST['user_id'])) ?
            $parent->clean($_REQUEST['user_id'], 'int') : 0;
        if ($user_id && $group_id && $is_owner &&
            $group_model->checkUserGroup($user_id,
                $group_id)) {
            $group_model->updateStatusUserGroup($user_id,
                $group_id, C\ACTIVE_STATUS);
            $this->getGroupUsersData($data, $group_id);
            return $parent->redirectWithMessage(
                tl('accountaccess_component_user_activated'),
                ["arg", "visible_users", 'start_row',
                'end_row', 'num_show', 'user_filter','group_limit']);
        } else {
            return $parent->redirectWithMessage(
                tl('accountaccess_component_no_user_activated'),
                ["arg", "visible_users", 'start_row',
                'end_row', 'num_show', 'user_filter','group_limit']);
        }
        break;
```

Step 2:

To render the webpage for user groups, file named ‘ManagegroupsElement’ was edited. Here the value for ‘group_limit’ was set before rendering it as shown in the figure:

```
if ($data['USER_FILTER'] != "" ||
(isset($data['NUM_USERS_GROUP']) &&
$data['NUM_USERS_GROUP'] > C\NUM_RESULTS_PER_PAGE)) {
$limit = isset($data['GROUP_LIMIT']) ?
    $data['GROUP_LIMIT']: 0;
}
switch ($user_array['STATUS']) {
    case C\INACTIVE_STATUS:
        e("<td><a href ='$action_url".
            "&amp;arg=activateuser&amp;group_limit=".($limit)."'>".
            tl('managegroups_element_activate').
            '</a></td>');
        break;
    case C\ACTIVE_STATUS:
        e("<td><a href='$action_url".
            "&amp;arg=banuser&amp;group_limit=".($limit)."'>".
            tl('managegroups_element_ban').
            '</a></td>');
        break;
    case C\SUSPENDED_STATUS:
        e("<td><a href='$action_url".
            "&amp;arg=reinstateuser&amp;group_limit=".($limit)."'>".
            tl('managegroups_element_unban').
            '</a></td>');
        break;
```

The modified Code has been tested on 45 users after making a Test Group and performing all actions: Activate, Delete, Ban and Reinstate. Thus, similar functionality was implemented for all kinds of actions possible for any user.

DELIVERABLES

src/configs/Config.php

src/controllers/components/SocialComponent.php

src/views/elements/ManagegroupsElement.php

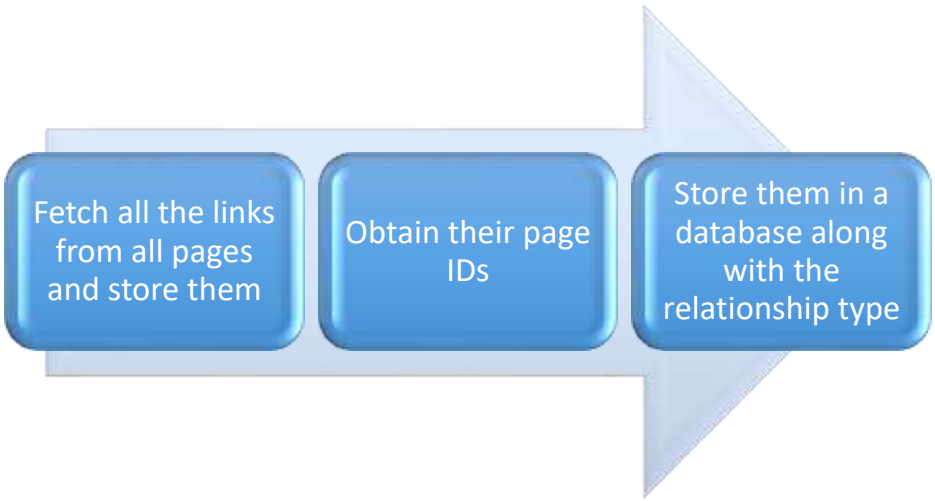
YIOOP PATCH UPDATE

A Patch was created for this issue in Mantis Bug Tracker, the ID for which is 0000177. The patch was accepted by Dr. Pollett after signing the Yioop Developer Agreement.

DELIVERABLE 3

While accessing any Wikipedia page, there is a link towards the left side of the page under *Tools*: "What Links Here". This displays all the other Wikipedia pages that are linked to the particular page. We shall implement the same feature by adding relationships links to Yioop pages. Thus, linkage between pages can be understood well and accessed easily.

To perform this action, three steps needs to be followed:



STEPS TO BE FOLLOWED

Step 1:

To fetch all links from any particular page, we need to observe the file where the Wikimedia content is being translated to HTML. Here, some regular expressions indicating the presence of a link are used to fetch the linkages. These links are stored and changed in a format suitable to get their page ids from already existing database.

The following regular expression is used in the function named 'PREG_MATCH_ALL' to fetch the URLs from the Wikimedia format:

preg_match_all("/\[\[([^\[\]]+?)\|([^\[\]]+?)\]\]/s", $html, $matches, PREG_PATTERN_ORDER); Here, PREG_PATTERN_ORDER is an argument specifying the order of results such that $matches[0] is an array of full pattern matches, $matches[1] is an array of strings matched by the first parenthesized sub pattern, and so on.

Step 2:

Once we have all the links, their Page ID's can be obtained from Yioop database. This is done by using Yioop's method 'getPageId' defined in Models->GroupModel

```
/**
 * Looks up the page_id of a wiki page based on the group it belongs to,
 * its title, and the language it is in (these three things together
 * should uniquely fix a page).
 *
 * @param int $group_id group identifier of group wiki page belongs to
 * @param string $page_name title of wiki page to look up
 * @param string $locale_tag IANA language tag of page to lookup
 * @return mixed $page_id of page if exists, false otherwise
 */
public function getPageId($group_id, $page_name, $locale_tag)
{
    $db = $this->db;
    $sql = "SELECT ID FROM GROUP_PAGE WHERE GROUP_ID = ?
        AND TITLE=? AND LOCALE_TAG= ?";
    $result = $db->execute($sql, [$group_id, $page_name, $locale_tag]);
    if (!$result) {
        return false;
    }
    $row = $db->fetchArray($result);
    if ($row) {
        return $row["ID"];
    }
    return false;
}
```

After obtaining the parent ID and IDs of all pages that link from the parent page, it is time to move to Step 3.

Step 3:

There is a table named GROUP_PAGE_LINK with fields as Relationship Type, Parent Page ID and the Child Page ID. A database connection is established and the table is populated with these values. As mentioned above, "-1" is used for relationship type at this point of time. However, in Deliverable 04, identifying relationship type will be the major focus.

```
CREATE TABLE GROUP_PAGE_LINK(
                LINK_TYPE_ID INTEGER, FROM_ID INTEGER,
                TO_ID INTEGER);
```

DELIVERABLES

For Fetching URLS, their ID and inserting to Database, the following code has been written. Changes have been made to setPageName method in GroupModel.php and another fetch links function added to WikiParser.php:

```
public function setPageName($user_id, $group_id, $page_name, $page,
    $locale_tag, $edit_comment, $thread_title, $thread_description,
    $base_address = "", $additional_substitutions = [])
{
    $db = $this->db;
    $pubdate = time();
    $fetched_ids = array();
    $parent_id_link = 0;
    $link_type_id = -1;
    $parser = new WikiParser($base_address, $additional_substitutions);
    $fetched_links = $parser->fetchLinks($page);
    $parsed_page = $parser->parse($page);
    if ($page_id = $this->getPageID($group_id, $page_name, $locale_tag)) {
        //can only add and use resources for a page that exists
        $parsed_page = $this->insertResourcesParsePage($group_id, $page_id,
            $locale_tag, $parsed_page);
        $sql = "UPDATE GROUP_PAGE SET PAGE=? WHERE ID = ?";
        $result = $db->execute($sql, [$parsed_page, $page_id]);
        $parent_id_link = $page_id;
    } else {
        $discuss_thread = $this->addGroupItem(0, $group_id, $user_id,
            $thread_title, $thread_description." ".date("r", $pubdate),
            C\WIKI_GROUP_ITEM);
        $sql = "INSERT INTO GROUP_PAGE (DISCUSS_THREAD, GROUP_ID,
            TITLE, PAGE, LOCALE_TAG) VALUES (?, ?, ?, ?, ?)";
        $result = $db->execute($sql, [$discuss_thread, $group_id,
```

```
            C\WIKI_GROUP_ITEM);
        $sql = "INSERT INTO GROUP_PAGE (DISCUSS_THREAD, GROUP_ID,
            TITLE, PAGE, LOCALE_TAG) VALUES (?, ?, ?, ?, ?)";
        $result = $db->execute($sql, [$discuss_thread, $group_id,
            $page_name, $parsed_page, $locale_tag]);
        $page_id = $db->insertID("GROUP_PAGE");
        $parent_id_link = $page_id;
        ImpressionModel::initWithDb($user_id, $page_id, C\WIKI_IMPRESSION,
            $db);
        ImpressionModel::initWithDb(C\PUBLIC_USER_ID, $page_id,
            C\WIKI_IMPRESSION, $db);
    }
    $sql = "INSERT INTO GROUP_PAGE_LINK (LINK_TYPE_ID, FROM_ID, TO_ID) VALUES (?,?,?)";
    //get ID for each of the child links stored in $fetched_links
    foreach($fetched_links as $key=>$link) {
        $fetched_ids[$key] = $this->getPageID($group_id, $fetched_links[$key], $locale_tag);
        //insert into GROUP_PAGE_LINK values of parent and child links
        if($fetched_ids[$key]!== false){
            $db->execute($sql, [$link_type_id, $parent_id_link, $fetched_ids[$key]]);
        }
    }
    $sql = "INSERT INTO GROUP_PAGE_HISTORY (PAGE_ID, EDITOR_ID,
        GROUP_ID, TITLE, PAGE, LOCALE_TAG, PUBDATE, EDIT_COMMENT)
        VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
    $result = $db->execute($sql, [$page_id, $user_id, $group_id,
        $page_name, $page, $locale_tag, $pubdate, $edit_comment]);
    return $page_id;
```

```
function fetchLinks($document)
{
    $matches =null;
    $links = array();
    $link_substitutions = $this->link_matches;
    foreach($link_substitutions as $keyLink=>$valueLink){
        $link_substitution = $link_substitutions[$keyLink];
        preg_match_all($link_substitution,$document,$matches, PREG_PATTERN_ORDER);
        $links = array_unique(array_merge($links, $matches[0]));
    }
    foreach($links as $key=>$link){
        $extract= substr($link,2);
        //removing page names with http, .com and .org
        if(preg_match('/http/',$extract)||preg_match('/.com/',$extract)||preg_match('/.org/',$extract)){
            unset($links[$key]);
            continue;
        }
        if(($pos=strpos($extract, '#')) !== false){
            $links[$key]=substr($extract,0,$pos);
            continue;
        }
        if(($pos=strpos($extract, '|')) !== false){
            $links[$key]=substr($extract,0,$pos);
        }
    }
    $links = array_unique($links);
    return $links;
```

Preparation work done to make this deliverable work on the local machine is as follows:

```
public function getURLs($data)
{
    //get the array containing wikimedia data as an input to this function
    $html = $data;
    $matches =null;
    echo sizeof($matches);
    preg_match_all("/\[\[([^\[\]]+?)\|([^\[\]]+?)\]\]/s",$html,$matches, PREG_PATTERN_ORDER);
    return $matches[2];
}
public function getIDs($URLArray)
{
    $groupmodel = new GroupModel();
    $group_id = 100;
    foreach ($URLArray as $key => &$value){
        $URLArray[$key] = str_replace(',','',str_replace(' ', '_', $value));
        $pageId[$key] = $groupmodel->getPageId($group_id, $URLArray[$key], "en-US");
    }
    return $pageId;
}
public function updateIDAndGroupLink($linkTypeId, $parentID, $pageID){
    $groupmodel = new GroupModel();
    foreach ($pageID as $key=> $valuePageID){
        if($pageID[$key] == false){
            continue;
        }
        $page=$pageID[$key];
        echo "Start of database insert";
        $groupmodel->updateGroupPageLink($linkTypeId, $parentID, $pageID[$key]);
        echo "Done";
    }
}
```

```
public function updateGroupPageLink($linkTypeId, $parentID, $pageID){
    echo "Group link update function";
    $db = $this->db;
    $sql = "INSERT INTO GROUP_PAGE_LINK (LINK_TYPE_ID, FROM_ID, TO_ID) VALUES (?,?,?)";
    $db->execute($sql, [$linkTypeId, $parentID, $pageID]);
}
```

FILES MODIFIED

src/configs/Config.php

src/library/WikiParser.php

src/models/GroupModel.php

YIOOP PATCH UPDATE

A Patch was created for this issue in Mantis Bug Tracker, the ID for which is 0000179. The patch was accepted by Dr. Pollett after signing the Yioop Developer Agreement.

DELIVERABLE 4

This deliverable involved adding a new feature to Yioop: "What Links Here." This feature lists all wiki pages that are linked to any particular page. The results displayed will be hyperlinks that will take the user to that wiki page. The wiki parser of Yioop fetches each individual MediaWiki internal links present in MediaWiki documents to be able to discover connections between different pages. This is done using a regular expression. For example, if page A is related to page B with a link that is displayed as Page B Link, then MediaWiki of document A would have [[PageB|Page B Link]]. Thus, the presence of these types of internal links can be extracted using a regular expression. Once we have all the links between the pages, it can be saved to the back end, and later used for information retrieval of linkages between different pages.

To perform this action, the following steps needs to be followed:

1. Add "What Links Here" to the drop down of the actions that can be performed on wiki pages.

2. Fetch the pages linked to any particular wiki page from the data base

3. Render the link page by displaying the results as hyperlinks pointing to the wiki content of their pages.

STEPS TO BE FOLLOWED

Step 1:

To fetch all links from any particular page, the first step is to add this feature, selecting which triggers the action of getting all linked pages to any particular page. This is done by assigning a keyword "links", which can then be used in the switch case used to trigger an action to the database.

```
case 'links':
    $data["MODE"] = "links";
    $data["PAGE_NAME"] = "linked";
    if (!isset($page_id) || !$page_id) {
        continue;
    }
    $page_info = $group_model->getPageInfoByPageId(
        $page_id);
    if (!isset($page_name)) {
        $page_name = empty($page_info['PAGE_NAME'])
            ? "links" : $page_info['PAGE_NAME'];
    }
    $limit = isset($limit) ? $limit : 0;
    $num = (isset($_SESSION["MAX_PAGES_TO_SHOW"])) ?
        $_SESSION["MAX_PAGES_TO_SHOW"] :
        C\DEFAULT_ADMIN_PAGING_NUM;
    $data["PAGE_ID"] = $page_id;
    $data["PAGE_NAME"] = $page_name;
    $data["DISCUSS_THREAD"] = empty($page_info["DISCUSS_THREAD"]
        ) ? -1 : $page_info['DISCUSS_THREAD'];
    $data["GROUP_ID"]=$page_info["GROUP_ID"];
    $data["LIMIT"] = $limit;
    $data["RESULTS_PER_PAGE"] = $num;
    list($data["TOTAL_ROWS"], $data["PAGES"]) =
        $group_model->getPagesLinkedToList($page_id, $limit,
        $num);
    break;
```

Step 2:

The controller calls method which fetches all pages that are linked to any wiki page. This is done by obtaining the page id where linkages are to found. This page id is passed to the database to get all linked pages. The page name of all linked pages is obtained and passed back to the controller.

```
public function getPagesLinkedToList($page_id, $limit, $num)
{
    $db = $this->db;
    $sql = "SELECT COUNT(*) AS NUM FROM GROUP_PAGE_LINK
        WHERE TO_ID = ?";
    $result = $db->execute($sql, [$page_id]);
    if ($result) {
        $row = $db->fetchArray($result);
        $total = $row['NUM'];
    }
    $i = 0;
    $pages = [];
    $sql = "SELECT G.TITLE AS PAGES_THAT_LINK
        FROM GROUP_PAGE_LINK L, GROUP_PAGE G
        where L.TO_ID = ? AND (L.FROM_ID = G.ID)"
        .$db->limitOffset($limit, $num);
    $result = $db->execute($sql, [$page_id]);
    if ($result) {
        while ($pages[$i] = $db->fetchArray($result)) {
            $i++;
        }
        unset($pages[$i]);
        $i--;
    }
    return [$total,$pages];
}
```

Step 3:

The final step renders the fetched results to the webpage. The results are displayed as a hyperlink, directing to the wiki page content of each result.

```
public function renderLinkPage($data)
{
    $logged_in = isset($data["ADMIN"]) && $data["ADMIN"];
    if ($logged_in) {
        $csrf_token = C\CSRF_TOKEN."=".$data[C\CSRF_TOKEN];
    }
    if($data["PAGES"]==null){
        e("No page has been linked to this page");
        return;
    }
    foreach ($data["PAGES"] as $key => $value) {
        $var = $data["PAGES"][$key]["PAGES_THAT_LINK"];
        $url = htmlentities(B\wikiUrl($var, true,
                $data['CONTROLLER'], $data['GROUP_ID']));
        $url .= $csrf_token;
        e("<td><a href='$url'>$var</a></td>");
        e("<br />");
    }
}
```

FILES MODIFIED

src/controllers/components/SocialComponent.php

src/locale/en_US/configure.ini

src/models/GroupModel.php

src/views/WikiView.php

src/views/elements/WikiElement.php

YIOOP PATCH UPDATE

A Patch was created for this issue in Mantis Bug Tracker, the ID for which is 0000181. The patch was accepted by Dr. Pollett after signing the Yioop Developer Agreement.

CONCLUSION

In conclusion, I feel that I have done the research part of my project. I am now aware of what parts do I need to work for CS 298.

During the span of CS 297, I started by gaining an understanding of categorization in Wikipedia and then got a chance to work with the vast code of Yioop by fixing a bug. This helped me gain an insight of Yioop's social component and related workflows. In the later deliverable, I studied the translation of Wikimedia document into html document and how are links identified in the Wikimedia document.

Further, I would be studying different relationships in the parent and child pages and how these can be used for effective information retrieval and providing relevant results to the user on the basis of query provided to the search engine. For handling relationships in a wiki system, a table will be used to identify relationship, be it 'binary' relation or 'Is a' relation. Different searches will make use of this table, e.g.: graphs can be drawn by extracting information from the table, depicting relationships. Another use may be building a family tree and identifying ancestors, or identifying siblings. We may also run a query like identifying nodes in a tree which start with a particular alphabet e.g.: In Genealogy, identifying people in a family whose name starts with alphabet A. In addition to this, Wikipages connecting to a particular topic or all pages linking to the topic can be found and the same process applied to child pages recursively to understand the structure of wiki.

REFERENCES

[1] Categorization. (2015). Retrieved from https://en.wikipedia.org/wiki/Wikipedia:Categorization

[2] D. Allemang and J. Hendler, *Semantic Web for the Working Ontologist*, Burlington, MA: Morgan Kauffman Press, 2008.

[3] Help:Category. (2015). Retrieved from https://en.wikipedia.org/wiki/Help:Category

[4] Yioop Documentation. (2015). Retrieved from https://www.seekquarry.com/p/Documentation

[5] Yioop Wiki Syntax. (2015). Retrieved from https://www.seekquarry.com/p/Syntax#HTML Tags