I.J. Modern Education and Computer Science, **2016, 9, 12-19**
Published Online September 2016 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijmecs.2016.09.02

Implementation of a Simple Document Repository System

Kazuaki Kajitori
Department of Fisheries Distribution and Management, National Fisheries University, Shimonoseki, Yamaguchi, Japan
Email: kajitori@fish-u.ac.jp

Kunimasa Aoki
Department of Fisheries Distribution and Management, National Fisheries University, Shimonoseki, Yamaguchi, Japan
Email: aoki@fish-u.ac.jp

***Abstract*—**Document repository systems have been developed actively for decades and have become quite popular on the internet. Now we can recognize several typical types of document repositories and find the full-text-search functionality as a useful search method for document repositories, so we think it has some significance to show as a mile stone an implementation scheme which covers basic types and functionality of the current document repositories which have full-text-search functionality.

The basic functions covered by our scheme are: to create, read, update, delete (so called CRUD) of documents and a session control with an optional authentication. We present a description for each part of CRUD of our scheme.

We implemented a library according to our scheme and developed small document repositories using the library. Our repository library and its applications form web applications whose script language is Perl. The implementation loosely follows the MVC (Model, View, Control) model for modularity and each MVC part of the library and its application named TEST are described. We also present a description of another application of our library which the author is conducting in the department he belongs to. We show some views of these applications.

We conclude that our library works fine for the needs of building small repositories in a short time.

***Index Terms*—**Document Repository, Full-text-searching, Web application.

I. Introduction

Document repository systems have been developed actively for decades since the web technology became available and have become quite popular on the web. Many universities and institutes have constructed their site repositories which serve research papers and research data produced at their sites and others (for instances [1][2]). The concept 'document repository' also includes code sharing sites like GitHub [3] and open online paper archives like arXive [4]. For sites like GitHub and arXive, it is necessary for users to be able to upload their files onto the sites.

Experiencing full-text searching on sites like Google and on document repository sites over the world, we have noticed that the full-text searching functionality is quite helpful in searching documents in a large collection. At the present we have many good choices of OSS full-text-search engines such as Solr [5] and Elasticsearch [6].

In this paper we focus on a document repository system with the following features. (1) it uses extensively OSS so that it is useful to more people who want to develop their own repositories. (2) it has a full-text-search function. (3) it pays attention to user uploading, thus has a user account management. (4) the system is simple and flexible to fit various needs including the need to build a small document repositories in a short time.

We hope that proposing a repository system of the sort is useful to help a plenty of digital resources in universities and institutes and etc to be used effectively as those digital resources will get more and more abundant and may become main resources for study and research in the near future.

The paper is organized as follows. In the section II we review some of related works. In the section III we propose our scheme of a repository system. In the section IV we present a detailed description of a sample implementation of the scheme in the section III and in the section V we present another repository example of implementing our scheme which is in a practical use. We state concluding remarks in the section VI.

II. Related Works

Here we review some of related works. Some of them pioneered the developments of document repositories and some of them let us know recent concerns about improving document repository systems.

As the document repositories have been flourished on the web, many papers on how to construct document repositories have appeared [7][8][9][10] [11][12] etc. For instance, in [7] the authors describe how their famous DSpace@MIT [1] was introduced and what issues should be addressed for institutional repositories. DSpace was first launched in 2002 and since then has been a resource

Copyright © 2016 MECS

of digital research materials for scholarly work taking place at MIT which now can be recognized as a pioneer of a typical type of repositories of universities and institutes. The source code of DSpace was released under Open Source BSD license and has been adopted by other universities and institutes for their institutional repositories. [10] is an empirical case study of adopting DSpace in National Taiwan University and the authors describe several modifications and extensions to DSpace in order to fit their needs for their repository. In [11] the authors based on their experience of Stanford Digital Repository (SDR) consider an architecture of 2nd generation of SDR (SDR2.0). Then the paper focuses on the technical systems plan and the data specifications for SDR2.0. These papers suggest important aspects of repository implementation which implementers of repositories should consider but these papers do not present concrete technical descriptions of the way how to develop their code maybe because their systems are quite large. The authors think that it is meaningful to present a simple way of developing a small repository system with technical details.

III. A Scheme of a Simple Document Repository System

Before proceeding concrete implementation of our repositories we had better to set an abstract scheme for repositories which makes basic ideas clear through implementation. We propose a simple scheme of a document repository system as the following:

A. A panoramic view of the system

The system has a library which provides with a uniform way of constructing various repositories and managing them under it. As a document repository library system, the main part deals with so called CRUD (Create, Read, Update, Delete) [14] of documents. The system has a uniform way to control the users' access which consists of session handling and authentication and 'routing'.

Therefore repositories of the system are implemented as applications of the library collection stated above. The library and its application repository most likely constitute a web application.

Since the system utilizes external databases, the system must have some abstraction to make it easy to change those databases (as asserted in [11]).

B. CRUD of the system

We assume that we select some engines for full-text-searching among those which are considered appropriate for the following scheme.

For document repositories, 'Creating' in CRUD means uploading a document file to the server and registering the document to the full-text-search engine and indexing it in the engine. The users can select the engines to use so that the CRUD of the system should be appropriately abstracted.

Indexing a document is done by the full-text-search engine. The indexing means to extract words to be searched from the document and record them in the database of the engine so the engine can find very quickly the documents which include the indexed words. This is just a digital version of the indexing for printed books, but we don't have to select words to be indexed for ourselves because today's full-text-search engines like Solr or Elasticsearch do the job for us. A difference between the indexing for a printed book and the digital indexing for a digital text is that the digital indexing is much faster and can gather much more words (almost all the searchable words in effect) in the text to be indexed. We call the searching words in the texts of documents using this digital indexing 'full-text-searching'.

'Reading' in CRUD for our repositories means searching documents including the full-text-searching by the engine and downloading the document file if the user wants it. 'Updating' is just updating the registration of the document in the system (in the full-text-searching engine and in the directories where the document files are stored) and 'Deleting' are just deleting the document from the system.

Also we have CRUD functionality for user administration and session handling. This sort of CRUD doesn't necessarily use the same database engine as the document searching uses (for example see the next section).

We suppose that a user can configure his/her environments such as the shown fields in each search result. The configured data are perpetual for each user account so that the user can enjoy the configuration in the later sessions.

C. User access control

The access control does the following:

(1) To check the user's eligibility to access the application (authentication). But authentication is not mandatory.

(2) To determine user's role for each application repositories of the system.

(3) To store session data so the user can use them in the later sessions (only when user accounts and authentication are used).

The superuser called 'admin' should be able to do the CRUD operations to all the documents in the system. It is optional that admin can do the system configuration for the repository system and the user administration (CRUD of users) from within the application's interface (the web interface if the application is a web application).

Each document has a flag showing if it is open to public which means that everyone can read it and the flag is set when the document is registered to the repository. The privileged user called 'uadmin' can do the CRUD operations to all the documents which he/she created or are open to public. Other users called just 'user' can do only READ of CRUD (Searching and Downloading) public documents. See Table 1.

Copyright © 2016 MECS

Table 1. Roles of Users

action	user	uadmin	admin
Search docs	△	△	○
Download docs	△	△	○
Add docs	×	○	○
Update docs	×	△	○
Delete docs	×	△	○
System admin	×	×	○

In Table I, a triangle means:

(a) For 'user', the activities are restricted to the public documents only.

(b) For 'uadmin', the activities are not allowed for non-public documents except for those he or she registered and the actions updating and deleting are restricted to the only docs that he or she registered.

(c) The action 'System admin' (abbrev. of the system administration) in the table means the system administration such as system configurations and user administration. Since web interfaces for 'System admin' is optional, 'admin' may have to do the jobs using the system tools for each server.

IV. A Sample Implementation

Now that we have a basic scheme of our repositories we show a sample implementation of a repository using the scheme. The sample is first described by an overview which is followed by the details according to the so called MVC model.

A. An Overview

We implemented a library collection for repositories in accordance with the scheme described in the previous section. We refer this library as repo or repo library which is used to construct repositories in accordance with our scheme. We implemented the repo system so that the repo library and a repository using the repo library constitute a web application and the web scripts are written in Perl language. We use Linux as OS and Apache as the web server and the other software tools we use are also OSS as shown in the following.

We implemented a repository for demonstration by using the repo library and we refer this repository as TEST. We demonstrate the TEST repository on an internet site [17] which will be maintained with some improvements.

As designated above, TEST is a web application whose web scripting language is Perl.

To give an overview of how an application of the repo library handles its sessions, we show in Fig.1 a UML like activity diagram for TEST around the session loop.

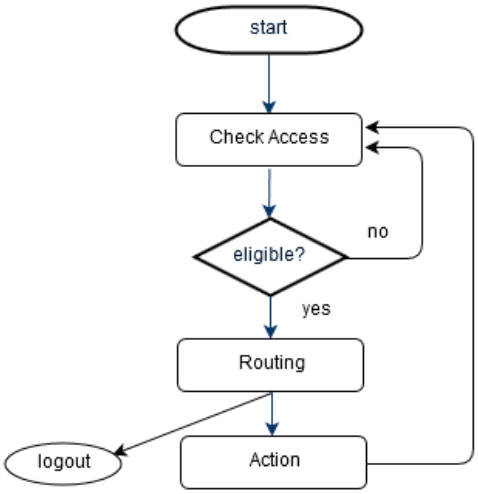

Fig.1. An activity diagram around the session loop

At each time TEST gets a request from the user, TEST checks the user's eligibility for the request at the 'Check Access' node in the diagram. If the check is ok, then TEST routes the request to some actions of our CRUD library (in the repo library) and show the next user interface. See Fig.2 and Fig.3 for interface examples.

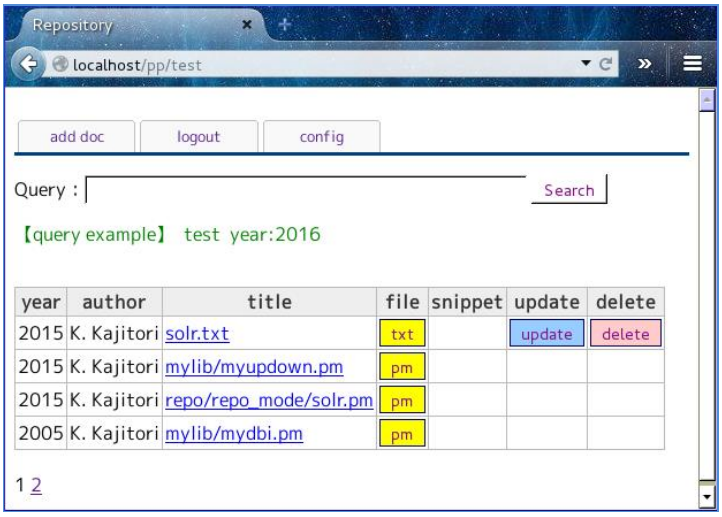

Fig.2. A view after login for a uadmin user of TEST

Copyright © 2016 MECS

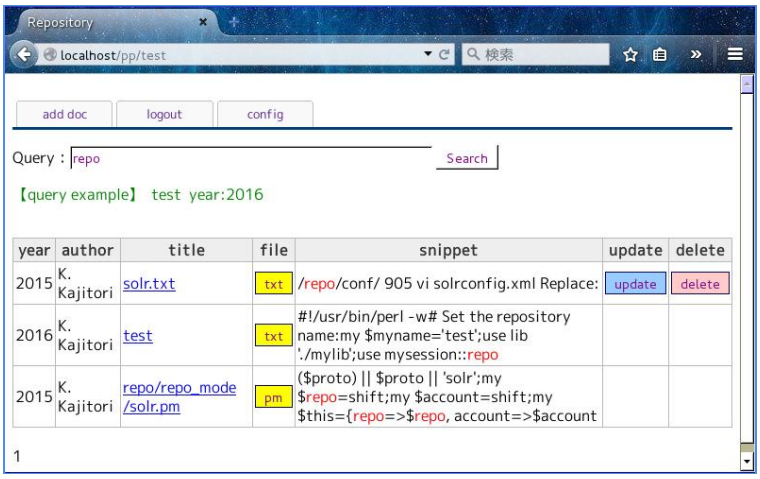

Fig.3. A view of a search result for a uadmin user of TEST

The Fig.2 is the first screen a 'uadmin' user sees after login. The Fig.3 is the screen which shows the result of the search by the query 'repo' which means to search the documents whose texts include the expression 'repo'. In these figures, the column 'snippet' shows the matched word in the query with the text around it in the document. The last two columns of the search result table show that in these pages this uadmin user can update or delete only the document named 'solr.txt' which he/she registered, which means the other documents in the tables are registered by other users and set public.

In Fig.2 the search result table is splitted into two tables by the pager. At the bottom-left of the page there are links to each of the splitted pages. The maximum number of rows in the table is user-configurable (in Fig.2,3 it is 3 but usually more). In these figures, on the top of each page is a tab menu which has the three tabs 'add doc' , 'logout' and 'config'. If the tab 'add doc' is clicked, then the page with a form for adding documents appears. The 'logout' tab is for logging out. If the 'config' tab is clicked, then the configuring page appears to configure the search engine and the fields and the maximum number of rows in the table (and possibly more).

To show the file structure of TEST, let the base directory of the application TEST be written as APP_HOME which is set as an executable directory under the web home.

TEST is just an application of our repo library which consists of the files:

APP_HOME/test
APP_HOME/repo/repo_model/*
APP_HOME/repo/repo_crud.pm
APP_HOME/repo/repo_view.pm
APP_HOME/repo/repo_view_template.pm
APP_HOME/template/*
APP_HOME/mylib/mysession.pm
APP_HOME/mylib/mysession/repo.pm
APP_HOME/mylib/myupdown.pm
APP_HOME/mylib/myroutine.pm

APP_HOME/repo/* files constitute the core of our repo library and APP_HOME/mylib/* files are generic libraries used in our TEST. APP_HOME/template/* files are HTML templates used in repo_view_template.pm.

TEST uses these libraries via the Perl program file APP_HOME/test which controls an access of users and 'routes' users' requests to functions of these library modules. So we can say Fig.1 is the activity diagram of the program file 'test'. The file 'test' determines how to use our repo library and so defines a repository application of the library. So we call a file like 'test' an 'app' file of our library.

The structure of 'test' is shown as follows;

(we only show the comments.)

```
#!/usr/bin/perl
# Initial set up
        # Set a session object
        # Check access
        # Set an object of the full-text-search engine
        # Set other session data
# Routing
        # Logout
        # Search
        # Add docs
        # Update docs
        # Delete docs
        # Configurations by users
        # Exception handling
# Subroutines
```

The 'Set a session object' part includes setting a CGI object for helping handle the web session and the application specific information such as the categories of documents. If you don't need user authentication, then just comment out or delete the 'Check access' part which only calls the check_access() method in mysession.pm module in just one line (so the change is easy). The part 'Set other session data' includes setting the fields and the maximum number of rows to be shown in a search result table which the user configures at the part 'Configurations

Copyright © 2016 MECS

by users' in the 'Routing' part. For more details of the file 'test', see 'Model part' and 'Controller part' in the following.

We can write another repository application by just writing another app file similar to 'test' which is as small as having just more than 250 lines and setting a database in the full-text-search engine (which we will explain later).

For the modularity of our repo library, we loosely obey an MVC (Model, View, Controller) architecture for a web application (for example [18]).

B. Model part

The model part includes repo_model class files:

repo/repo_model/solr.pm
repo/repo_model/elastic.pm
etc

Each file under the directory repo_model executes CRUD of the full-text-search engine like Solr and Elasticsearch. Solr and Elasticsearch both have HTTP API. For web applications HTTP API is quite useful because HTTP API is certainly available for a web application (the web server and the web scripting language you choose both talk HTTP) and so you have a convenient web browser interface to the full-text-search engines and you don't need a driver for the full-text-search engines written specifically for the web scripting language you want to use. Here is a Solr example of HTTP API which deletes the document with id=abc :

SOLR_URL/update?commit=true&stream.body=<delete><id>abc</id></delete>

(Here, SOLR_URL is the Solr server url for HTTP access which usually has the port 8983)

So far we have implemented solr.pm only as the repo_model class files and elastic.pm is just the same as solr.pm except for the module name.

For Solr, we adopt Solr5.4 for enough features and stability. Solr manages each database as a 'core'. We set a core 'test' for TEST in Solr and define the fields for the core test as in Table 2.

The fields could (should?) be chosen so that they fit to the sort of documents the repository assumes. But in many cases we can not decide precisely the types of documents we will accept to the repository. So the above list of fields are quite generic and we think that this generic list fits many cases of repositories.

We set each of these fields be indexed in Solr. The field type "text_general" means that the indexing uses an analysis typically suitable for English texts. The field type "text_ja" means that the indexing uses a morphological analysis [15] for Japanese texts. For Japanese texts there is also a field type “text_cjk” which uses n-gram [15] analysis. The morphological analysis is one decent way to get words to be indexed from Japanese texts. The type 'text_ja' could be used for texts in other languages like English. The field "year" is meant to be the year the document was made.

Table 2. Solr fields for TEST

field name	type	multivalued
id	string	FALSE
title	text_ja	TRUE
author	text_ja	TRUE
category	text_ja	FALSE
year	int	FALSE
subject	text_ja	FALSE
description	text_ja	FALSE
comments	text_ja	FALSE
keywords	text_ja	FALSE
links	text_general	FALSE
content_type	string	FALSE
text	text_ja	TRUE
resourcename	text_ja	FALSE
last_indexed	date	FALSE
public	boolean	FALSE
resgisterer	text_general	FALSE

The field 'category' determines which category the document belongs to. There should be a selection for category. For TEST we set the selection to be 'Perl program', 'Perl module', 'article', 'other' because sample documents in TEST are files related to the repo library. The selection can be set for each application of the repo library in its 'app' file. The field “public” determines whether the doc is open to public or restricted to the registerer of the document and the superuser admin and so the type of the field is boolean which takes 'TRUE' or 'FALSE' as value.

User data (account names, password, configuration data by users etc) are stored in an RDB (not one of the full-text-search engines) for some reasons. One reason is that we are accustomed to RDB than full-text-search engines for tasks like administrating user data. Another reason is that at least for now the security does not seem to be a primary concern for full-text-search engines. For example, Solr5.4 does not have security controls on the web admin interface other than the Basic Authentication. We use MariaDB as an RDB engine and define the database 'user' which has the tables created by SQL's CREATE sentences of MariaDB as follows.

```
CREATE TABLE `account` (
  `account` varchar(20) NOT NULL,
  `password` varchar(20) NOT NULL,
  PRIMARY KEY (`account`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `app` (
  `name` varchar(20) DEFAULT NULL,
  `code` int(11) NOT NULL,
  PRIMARY KEY (`code`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `repo_role` (
```

```
  `account` varchar(10) NOT NULL,
  `app_code` int(11) NOT NULL DEFAULT '0',
  `uadmin` tinyint(1) DEFAULT NULL,
  `admin` tinyint(1) DEFAULT NULL,
  PRIMARY KEY (`account`,`app_code`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `session` (
  `session_id` varchar(40) NOT NULL,
  `account` varchar(10) DEFAULT NULL,
  `login_datetime` datetime DEFAULT NULL,
  PRIMARY KEY (`session_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `session_data` (
  `account` varchar(10) DEFAULT NULL,
  `name` varchar(20) DEFAULT NULL,
  `value` text,
  `dumped` tinyint(1) DEFAULT '0',
  KEY `account` (`account`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

The tables 'account' and 'session' are used for authentication. The tables 'app' and 'repo_role' are used to determine the roles of users for each application of the repo library. The table 'session_data' stores user configuration data. Since data in the table 'session_data' is perpetual in the sense that it is not lost when the session ends. So the name of the table could be 'user_data' or something but we chose the name because it means the data be valid through the sessions.

As in [13], authentication during a session (after login) is done by checking the (account,session_id) pair in the table 'session' being the same as the cookie pair from the client. Also as suggested in [13], for security we delete the record on the 'session' table on logging out and delete the uploaded document file in APP_HOME/repo/doc on deleting documents.

As in [16], session data are kept during a session. In [16], session data are connected to the session id using the supergloval $_SESSION of PHP. In our case, the (account, session_id) pair is saved in the table 'session' but other session data are saved in the table 'session_data' and connected to the user 's account (not to the session_id) so that they can be used in the later sessions. The table 'session_data' can be used typically as the place storing user configuration data such as the choice of full-text-search engine or the choice of fields or the maximum number of rows to be shown in the search result tables. We can save almost any data structure of Perl as session data by serializing data to a string using a Perl module like Storable[20] or Sereal[21]. If the field 'dumped' in the table 'session_data' has the boolean value TRUE, then the data in the field 'value' is a serialized string so has to be restored (de-serialized) by the same tool (e.g. Storable, Sereal) we use to serialize it.

C. Controller part

The controller part is carried out by two files. One is repo/repo_crud.pm which sends user requests to the CRUD functions of the model part and get responses. The add method in repo_crud.pm transforms the HTTP form data into JSON data and pass it to the corresponding method of repo_model::* class (many full-text-search engines accept JSON data when adding a document) and saves the uploaded document file into a directory under APP_HOME/repo/doc. Similar for the update method. The delete method just calls the corresponding method of repo_model::* and deletes the uploaded document files. The search method transforms the query if necessary and send it to the corresponding method of repo_model::* and get the result in JSON format which is transformed into HTML table by the methods in repo_view.pm.

The other file in Controller part is the app file 'test' which like we said before is called at each access of the user and do the following jobs:

1. Session handling, including authentication if needed.
2. Routing which calls actions of repo_model or repo_crud in response to HTTP requests from the user.

The URL to access the repository TEST is APP_WWW_HOME/test/PATH_INFO where APP_WWW_HOME is the web home of TEST (which is the same directory as APP_HOME for TEST) and PATH_INFO should be some concrete string of letters which is used for the routing. For example if PATH_INFO is the empty string, then the app file 'test' calls actions to show the search form and the search result table for the empty query (Fig.2). The routing is controlled also by 'action' parameter designated by <input type="hidden" name="action" value="***"> tag which can be used in the 'app' file instead of PATH_INFO for routing.

D. View part

The view part is handled by the files repo_view.pm and repo_view_template.pm.

The module repo_view.pm generates a search form and add, update, delete forms and a search result table in HMTL format. Then each of those HTML data is handed over to a corresponding method in the module repo_view_template.pm where the HTML data is poured into an appropriate template in the directory APP_HOME/template and an HTML page is generated by the Perl module HTML::Template::Pro. HTML templates are used to make it a lot easier to generate similar HTML pages at times.

For example, right after the user logins we show a page with a search form and the search result table of the empty query "" (Fig.2) where we use an HTML template APP_HOME/template/search.tmpl and 'print_form_result' method in the repo_view_template module for which the search form is made by the method 'search_form' and the search result is HTML formatted by the method 'search_result' in the repo_view module.

V. A Practical Implementation

The department of the college the authors belong to is now operating a repository system which was developed

in the way similar to TEST in the previous section. Actually, this repository uses a slightly old version of our repository library and more stable than TEST. We refer this system as SOTU which was named after japanese word 'sotugyo' meaning graduation and the purpose of our repository was to provide the faculties of our department with an easy access to graduation theses of the students of our department. Before operating SOTU, we could read the theses in a book style in the library of our college and anybody can still read them there. So we decided not to set SOTU open to public. SOTU could be easily configured to be open to public as described in the previous sections but for the time being we use the login scheme because we take some security considerations about digital data into account and theses are still open in the college library. (Of course the login scheme is not only for security but also useful to maintain user data as described in the previous sections.)

The theses in our repository SOTU are written in Japanese and stored as PDF files. A search view looks like Fig.4.

Fig.4. A search view of SOTU

Fig.4 is a view for 'user' and so 'update' and 'delete' columns are invisible. Actually for SOTU there are just one 'user' and one 'admin' which are enough for our purpose. Namely the first author adds and updates and deletes theses when necessary and the other faculties in the department are just comfortable to be just 'user'. In Fig.4, 'config' tab is invisible too but this can be made visible by using the new version of the repo library we use for TEST. In Fig 4, the field 'year' represents the year the thesis was written. The fields for the full-text-search engine of SOTU is set to be the same as TEST. SOTU is a thesis repository and mono-categorized and so we could set a single selection for the category field but we set it as Paper', 'Thesis', 'Report', 'Other' for possible future purposes. As in TEST, the only engine we can use now in SOTU is Solr and Solr does a good job for the both repositories.

VI. Conclusions

The library repo works fine for TEST and SOTU and we think our small repo library is quite convenient to build a repository in a short time.

The digital resources are increasing their significance and amount compared to non-digital resources. The library repo can help small organizations like colleges or each department utilize their digital resources. Repositories built with the repo library can allow all users to upload documents and the repo library can be developed further so that it will be able to support some easy-to-use e-portfolio [19] for students and faculties in the future.

The library repo can manage many repository applications under it. So it is more convenient to have a simple interface to browse and select repositories under it which we don't have now. We don't have any admin interfaces to administrate from within the repo applications other than CRUD of all the documents. Web interfaces for 'admin' to manage applications' configuration and others is an optional feature of our scheme described in the section II. But building web interfaces for administration needs an extreme care to security to prohibit non-eligible persons from accessing the admin interfaces. So we will take time to implement those 'admin' interfaces in the future.

Copyright © 2016 MECS

REFERENCES:

[1] DSpace@MIT, https://dspace.mit.edu/.
[2] Stanford Digital Repository, https://library.stanford.edu/research/stanford-digital-repository.
[3] GitHub, https://github.com/.
[4] ArXiv, https://arxiv.org/.
[5] Apache Solr, http://lucene.apache.org/solr/.
[6] Elasticsearch, https://www.elastic.co/.
[7] P. Baudoin, M. Branschofsky, Implementing an Institutional Repository: The DSpace Experience at MIT, Science & Technology Libraries, Volume 24, Issue 1/2, June 2004, p.31-45.
[8] L. Sokvitne, J. Lavelle, Implementing an Open Jurisdictional Digital Repository - the STORS Project, D-Lib Magazine, Vol.10, 6, 2004.
[9] M.Borchent, J. Richardson, G. Mitchell, Implementing HarvestRoad Hive Digital Repository, Queensland Univ. of Tech. ePrints Archive, 2005.
[10] C. Tsai, J. Hsiang, H. Chen, Implementing an institutional repository for digital archive communities: experiences from National Taiwan University, Proceedings of the 2006 international conference on Dublin Core and Metadata Applications: metadata for knowledge and learning. Dublin Core Metadata Initiative, 2006.
[11] T. Cramer, Designing and Implementing Second Generation Digital Preservation Services: A Scalable Model for the Stanford Digital Repository, D-Lib Magazine, Vol.16, 9/10, 2010.
[12] bepress, Building a Framework for IR Success: A Roadmap, Research on Institutional Repositories: Articles and Presentations, 2014.
[13] I. Purushottam, V. Thakare, Designing Efficient Security Technique for Data Storage in Cloud Computing, IJCA Proceedings on National Conference on Recent Trends in Computer Science and Engineering, MEDHA 2015(4).
[14] CRUD,https://en.wikipedia.org/wiki/Create,_read,_update_and_delete
[15] Steve Cohen, Morphological Analysis searched for you, http://www.basistech.com/whitepapers/n-gram-vs-morphological-analysis-EN.pdf, 2011.
[16] Y. Jiang, Zhan Huang, Zhanhong Huang, Design and Implementation of a General Web-based Course Teaching Management System, I.J. Education and Management Engineering, 2012, 11, 1-7 (2012).
[17] http://mantiq.fish-u.ac.jp/pp/test
[18] MVC, https://zeekat.nl/articles/mvc-for-the-web.html
[19] e-portfolio, https://en.wikipedia.org/wiki/Electronic_portfolio
[20] Storable, http://search.cpan.org/~ams/Storable-2.51/Storable.pm
[21] Sereal, http://search.cpan.org/~yves/Sereal-0.330/lib/Sereal.pm

Authors' Profiles

Kazuaki Kajitori, Ph.D, is a professor of the Department of Fisheries Distribution and Management at National Fisheries University in Japan. In teaching, he has been in charge of classes of mathematics and statistics and computer sciences. In his classes, he has been utilizing IT methods extensively. He wrote online texts and courses' home pages and conducted many online exams and let students do online exercises as the preparation of online exams. In research, he has studied mathematical logic which led him computer related fields like data mining and databases and e-learning. He has developed several web applications including one treated in this paper.

Kunimasa Aoki is an associate professor of the Department of Fisheries Distribution and Management at National Fisheries University in Japan. In teaching, he has been in charge of classes of mathematics and statistics and computer sciences. In his classes, he has been utilizing IT methods extensively. He wrote online texts and courses' home pages and conducted many online exams and let students do online exercises as the preparation of online exams. In research, he has studied mathematical logic, especially theory of computation which led him computer related fields like data analysis. He co-authored papers of other fields in which he is in charge of data analyses.

How to cite this paper: Kazuaki Kajitori, Kunimasa Aoki,"Implementation of a Simple Document Repository System", International Journal of Modern Education and Computer Science(IJMECS), Vol.8, No.9, pp.12-19, 2016.DOI: 10.5815/ijmecs.2016.09.02

Copyright © 2016 MECS