

AffilDB: A Distributed J2EE Web Application for
Role-Based Management of Affiliate Data

Shannon William Rice

A Thesis in the Field of Information Technology
for the Degree of Master of Liberal Arts in Extension Studies

Harvard University

November 2005

Abstract

AffilDB is three-tiered software application that allows a user with a thin-client (Web browser) to interact with a Web interface to business logic components and a relational database containing records of affiliates at a university research center with multiple subsidiary departments. Functionality available to the user will be granted or withheld based on the role of the user as defined within the application. This thesis documents the needs assessment, design, development and implementation of AffilDB, and includes an appendix with a user guide and the source code.

Dedication

I dedicate this to my wife Akemi, for her support and hard work while I was writing this thesis and earning my degree, and our two children, Kian and Liana.

Acknowledgments

I'd like to express thanks to my thesis director, Patrick McGowan for his advice and guidance during the project, and Dr. William Robinson, ALM in IT research advisor for helping me select the research topic. I'd also like to thank Ethan Kiczek, director of information technology at the Weatherhead Center for International Affairs, Steven Bloomfield, associate director at the Weatherhead Center for International Affairs, Dr. Shinju Fujihira, associate director of the Program on U.S.-Japan Relations, and Dr. Frank Schwartz for their support of the project.

Table of Contents

Table of Contents	vi
List of Figures	xiii
Chapter 1 Introduction	1
Mission: Create a Database Management Software Solution.....	1
Organization of Thesis	2
Chapter 2 Needs Assessment	3
Description of Data	3
Profile of Users	3
Definition of Scope	4
Amount of data	4
Core users and ‘roles’	4
Priorities of new system.....	5
Priority 1 - Maintain data integrity	5
Priority 2 – Eliminate duplication of work	5
Priority 3 - Maximize intuitiveness/ease of use of system	6
Priority 4 - Design a system that will be easy to maintain and extensible.....	6
Priority 5 - Design a database that will handle multiple affiliations easily	6
Priority 6 - Maintain current functionality of existing systems	7
Chapter 3 Distributed System	8
Chapter 4 Database Schema.....	10

Role-based authorization based on schema	14
Chapter 5 User Interface	16
Chapter 6 Technology Stack.....	26
Operating System – Red Hat Linux 9	26
DBMS – MySQL v. 3.23.54	26
API - Java 2 Platform Enterprise Edition, v 1.4	27
Web Server – Tomcat Version 5.0.....	27
Chapter 7 Security.....	29
Authentication.....	29
Authorization	30
Encryption.....	34
Chapter 8 Release Management Process	36
Packaging of application.....	37
Chapter 9 Third Party Software Libraries.....	39
Library for PDF File Generation.....	39
Library for uploading files	39
Library for encryption/decryption.....	40
Potential Risks with using third party libraries:.....	40
Chapter 10 Conclusion.....	42
Future Enhancements:.....	43
References.....	46
Appendix 1 Users Guide.....	47
Appendix 2 Application Code	49

package edu.wcfia.database.beans	49
AddressBean.java.....	49
AffilBean.java.....	53
AffiliationBean.java.....	58
BioBean.java.....	61
MailingListBean.java.....	62
RIBean.java.....	65
UsersBean.java.....	66
package edu.wcfia.database.servlets.....	68
AccessControlFilter.java.....	68
AffiliateMaster.java	70
AuthenticateServlet.java	73
ExternalMailing.java.....	77
FellowsReport.java	79
InternalMailingLabels.java	81
LogoutServlet.java	89
ProgramMailingLabels.java.....	90
UploadServlet.java.....	97
package edu.wcfia.database.taghandler	98
CalendarTagHandler.java	98
package edu.wcfia.database.utils	100
MyFileRenamePolicy.java.....	100
UnsupportedFileTypeException.java.....	101

JSP Files.....	103
actionOnAddress.jsp	103
actionOnAffiliation.jsp	109
actionOnBio.jsp	111
actionOnMailingList.jsp	114
actionOnRI.jsp	120
affilTable.jsps.....	122
cancel_delete.jsp	127
cancel_insert.jsp.....	128
checkAuthorization.jsp	129
confirm_enter.jsp	129
delete.jsp	131
deleteAddress.jsp	134
deleteAffiliation.jsp	134
deleteAllAddresses.jsp.....	134
deleteAllAffiliations.jsp.....	135
deleteAllBios.jsp.....	136
deleteAllMailingLists.jsp.....	136
deleteAllRIs.jsp.....	137
deleteBio.jsp.....	137
deleteMailingList.jsp	138
deleteRI.jsp	138
displayAddressRows.jsp	139

displayAffiliationRows.jsp	141
displayBioRows.jsp	143
displayMailingListRows.jsp	145
displayRIRows.jsp	146
enter.jsp	148
error.jsp	155
fedex_report.jsp	155
header_navbar.jsp	157
insert_in_db.jsp	158
list.jsp	161
logout.jsp	163
report_externalMailing.jsp	164
report_fellows.jsp	166
Report_handwrittenmailing.jsp	167
report_internalmailing.jsp	169
report_masterlist.jsp	172
report_programmailing.jsp	174
report_workingpapersmailing.jsp	177
reportLink.jsp	179
reports_individual.jsp	180
reports.jsp	181
results_adv.jsp	182
results.jsp	185

search_adv.jsp.....	186
search.jsp.....	189
update.jsp	192
upload_photo.jsp.....	196
viewAddAffilData.jsps	197
view_individual.jsp.....	199
HTML Files	204
AffiliateMasterList_help.html	204
AffiliatePhoneList_help.html.....	204
ExternalMailing_help.html	204
FellowsList_help.html	205
InternalMailing_help.html	205
MutlitplePrimaryDesignations_help.html.....	206
ProgramMailing_help.html.....	206
ProgramMailingTab_help.html.....	206
WorkingPapersMailing_help.html.....	207
Configuration Files	207
style.css	207
web.xml.....	211
Database Scripts.....	214
init_db.sh.....	214
cr_address_tbl.sql.....	215
cr_affiliation_tbl.sql.....	216

cr_affiltypes_tbl.sql	216
cr_bio_tbl.sql	216
cr_census_tbl.sql.....	217
cr_country_tbl.sql	217
cr_mailinglist_tbl.sql	218
cr_programs_tbl.sql	218
cr_ri_tbl.sql	218
cr_users_tbl.sql	219
load_address_tbl.sql.....	219
load_affiliation_tbl.sql.....	219
load_affiltypes_tbl.sql.....	220
load_bio_tbl.sql.....	220
load_census_tbl.sql.....	220
load_country_tbl.sql.....	220
load_mailinglist_tbl.sql.....	220
load_programs_tbl.sql.....	220
load_ri_tbl.sql	220
load_users_tbl.sql.....	220

List of Figures

3-1 System diagram of AffilDB System	9
4-1 Schema design of core database tables	11
4-2 Schema design of auxiliary database tables.....	12
5-1 Chart of flow of control of AffilDB.....	16
5-2 Screenshot of search page.....	17
5-3 Screenshot of advanced search page.....	18
5-4 Screenshot of search results with both editable and non-editable results	19
5-5 Screenshot of add affiliate page.....	21
5-6 Screenshot of view individual page	24
5-7 Screenshot of reports page	25

Chapter 1 Introduction

AffilDB is a Web-based application designed to solve a data management problem at the Weatherhead Center for International Affairs (hereafter referred to as WCFIA). WCFIA has several subsidiary departments (hereafter referred to as programs) as well as a central administration that need to share information on current and past affiliates. AffilDB was designed to replace an existing database system. Among the drawbacks of the system to be replaced were that it was not Web-based, depended on client software to be installed on each user's machine to access the database that was stored on a shared fileserver, had a complicated user interface, and was not easy to administer and maintain. Sharing data on affiliates was cumbersome and existing data was not frequently updated. This led to workflow inefficiencies including duplication of work and use of inaccurate or outdated data.

Mission: Create a Database Management Software Solution

AffilDB addresses a classic data management problem. Users need to create, modify and share data, yet have differing business uses for this data. In addition to analyzing the organization and individual users' needs, issues to address include security (authentication, authorization, and encryption), choice of platform to use, schema design, business logic design, as well as user interface design.

Organization of Thesis

This thesis is organized into chapters, with this chapter providing the general introduction to the project. Chapter 2 describes the needs assessment phase of the project and preliminary lessons learned from interviews with users prior to development. Chapters 3 through 9 each focus on a specific technical issue central to development. Issues covered include discussion of the distributed design, database schema, the user interface design, security, the release management process, as well as decisions regarding which platforms and third party libraries to incorporate into the application. The thesis concludes with a chapter discussing the project as a whole, including reflection on major challenges faced, problems encountered and solved, changes made during development, as well as recommendations for future enhancement.

Chapter 2 Needs Assessment

To assess the needs of WCFIA in preparation for designing an appropriate system interviews were held with potential core users to ask how they currently manage their data and what they would like to see in a new database application. This chapter provides background on decisions regarding the type of information to be stored in the database, as well as the priorities determined through these interviews.

Description of Data

WCFIA is a research center founded in 1958 (then known as CFIA). The data in the database contains records of the over 4,000 people who have had an affiliation with the center (or in many cases, multiple affiliations). The data grows each year, due to over a hundred new affiliates joining the center annually to conduct research and natural staff turnover.

Profile of Users

The primary users of the database will be employees of the central administration department who need to manage the records of all affiliates and staff of the various subsidiary programs who are primarily interested in the subset of affiliate records corresponding to (i.e. have had official university appointments with) their particular program. Some programs choose to maintain their own departmental databases. Ideally, the new database application, referred to in this paper as AffilDB, would attract

administrators from these programs through clean, attractive design, ease of use, and robust functionality that meets their business needs.

Definition of Scope

Amount of data

The data in the database will consist of between 4,000 – 5,000 affiliate records, as well as relational data in secondary tables that contain information on things such as affiliation types, mailing addresses, research interests, etc.

Core users and ‘roles’

Although anybody with a valid university ID can be granted access by the AffilDB administrator, core users will be employees of the WCFIA. It is expected that initially a group of about three to four users from the WCFIA central administration will use AffilDB, and an additional ten to twenty employee users from the various programs will use AffilDB to update and view data on their respective affiliates. Each user who is set up to use the system will be assigned a role that will grant them authority to use functionality according to their needs. There will be three categories of roles.

There will be an ‘admin’ role assigned to the central administration staff responsible for maintaining the database. There will be a set of program roles, each of which will correspond to a name of a program at the WCFIA, and will allow users to edit affiliate records with a current or past affiliation with the specified program, and there will be a ‘read-only’ role, for users who only need to view data and generate reports.

The ‘admin’ role, not surprisingly, will have read, write, and delete permissions on all records in the database. Users, such as program coordinators, assigned one of the

program roles will be allowed to add or modify affiliate records with an affiliation corresponding to their role/program. For example, a user with a 'US-Japan Program' role would be able to add new members to the US-Japan Program, as well as edit records in the database that have an affiliation with US-Japan Program. Records of affiliates in the database with multiple affiliations will be editable by more than one category of user. For example, if a person was affiliated with one program from 1998-99, left, and later came back to the WCFIA to be affiliated with another program from 2003-04, administrators from both programs could edit this record. However, no administrators from unrelated programs would be able to make any edits on this record. All users of the database will have read access to view the information. The third 'read-only' role will be granted to users who have a need to do look-ups of affiliates' information, but have no need to edit any records.

Priorities of new system

Below is a list of priorities identified during the preliminary needs assessment meetings.

Priority 1 - Maintain data integrity

As one colleague put it, "a database is only as good as the data that's in it." We need to incorporate systematic methods for updating data and maintaining data integrity. It is very important to the users that administer programs, that they, and only they be granted access to modify data for affiliates they are responsible for.

Priority 2 – Eliminate duplication of work

A few of the WCFIA programs are currently using decentralized databases. The major drawback to this is that much effort is duplicated. As a result, although Center

subsidiary programs may have correct information regarding their affiliates, this information is often not transferred into the central database.

The creation of a well-designed and easy to use centralized database and set of application interfaces to access it has the potential to reduce the workload and improve the work processes for all people involved in data management at WCFIA by allowing a single data repository.

Priority 3 - Maximize intuitiveness/ease of use of system

Another important concept is 'ease of use.' Comments from initial meetings included "err on the side of simplicity," and "think about the non-technical user when designing." One of the main problems cited with the existing MS Access database is that it is very difficult and not intuitive to use. This is particularly discouraging to the occasional user, who cannot remember the set of steps required to access a particular functionality.

Priority 4 - Design a system that will be easy to maintain and extensible

It is important to create a system that is as easy to configure, manage, and use as possible. The system should be scalable so that it will be easy to modify to add functionality in the future as the needs of the organization change.

Priority 5 - Design a database that will handle multiple affiliations easily

Some records in the database will have multiple affiliations. For example, a former 'Olin Institute' Fellow may also be a former 'US-Japan Program' Fellow. It is important that the role-based scheme we have discussed prevent users of the database from altering information on affiliates who are not members of their organizational unit. As previously mentioned, one technical challenge is how to handle the case of an affiliate with multiple

affiliations. A technical explanation of the solution to this can be found at the end of Chapter 4.

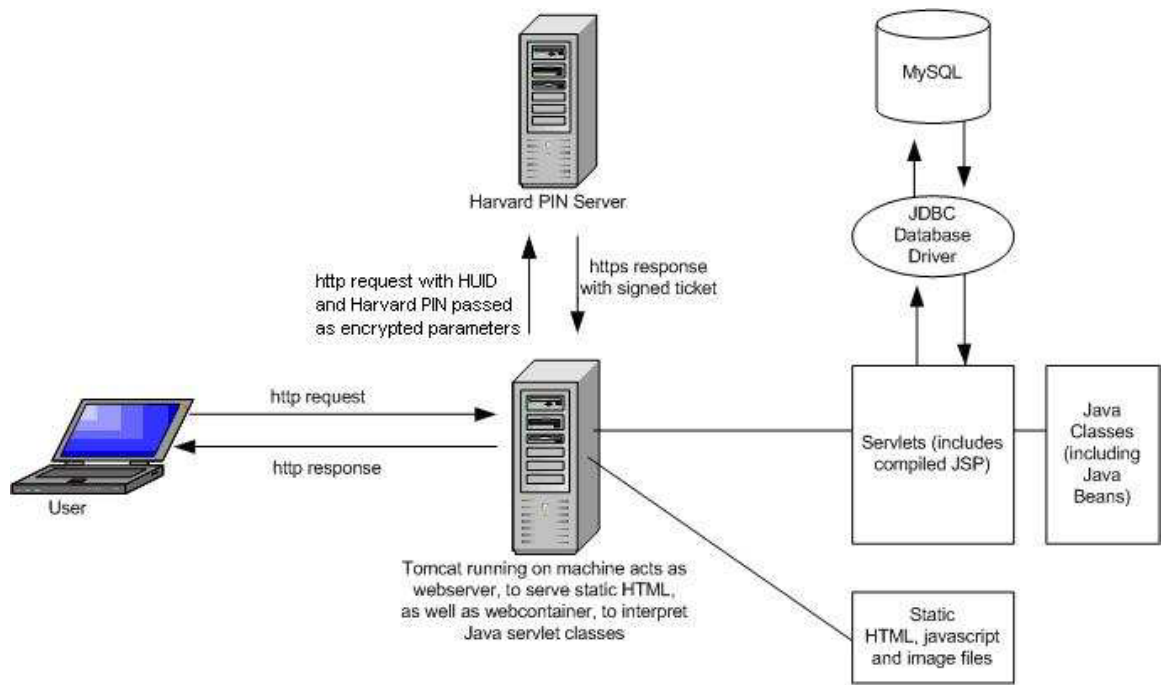
Priority 6 - Maintain current functionality of existing systems

Since some Programs have data management systems of their own, we should attempt to maintain the functionality that they currently find useful and transfer this to the new system to the extent possible without compromising other priorities.

Chapter 3 Distributed System

The system AffilDB will replace is a database developed around 1999. One drawback of the system to be replaced is that it requires a specific version of Microsoft Access to be installed on each user's machine, and it requires the user to be connected to the WCFIA network in order to access the file server where the database is located. By designing a distributed Web-based system, the user will have the freedom to run any OS, and will simply need a connection to the Internet and a browser. Of course, the user must also have a valid Harvard ID and PIN and they must be entered into the system in advance by the AffilDB administrator to access the system.

Figure 3-1 illustrates the distributed nature of the system. After a user logs in, and the authentication information is passed to the application from the Harvard PIN server, requests during the session are passed through the Web server to the Servlet container. JDBC provides the Servlets/JSPs an interface through which to connect to the MySQL database, where all of the data resides.



3-1 System diagram of AffilDB System

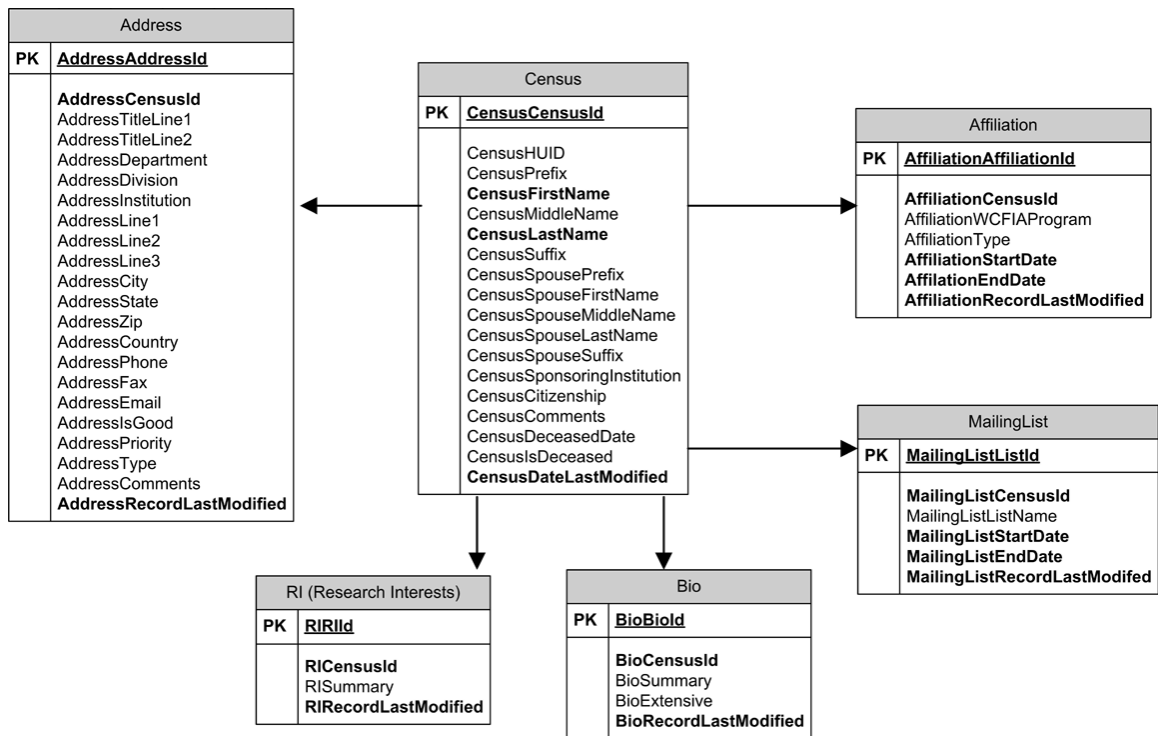
The distributed design of the system provides a separation of functionality that will simplify the process of upgrade or substitution of platforms should it be necessary in the future. Although the system currently uses MySQL as its DBMS, it would be feasible to replace MySQL with another DBMS, such as Oracle. This would simply require a different JDBC Driver to interface with Oracle. In the same manner, as long as a Web server is configured with a J2EE compatible Servlet container such as Tomcat or JBoss, the operating system could be changed with minor adjustments.

Chapter 4 Database Schema

Much of the structure of the schema for the tables in AffilDB was transferred directly from the database to be replaced. Some tables were removed or modified in order to best meet the set of needs as defined during the needs assessment phase. For example, the previous database contained data on people who did not technically have affiliations with the center, such as conference participants. The executive decision was made to limit records that will be stored in the database to people with official affiliations with the center.

Figure 4-1 illustrates the schema used in AffilDB. The central table is the Census table which represents basic information on an affiliate. Each of the other tables provides additional data, and one Census record may have zero or more records from any of the other five tables. However, it would be unusual to have a Census record that has no Affiliation record associated with it.

Figure 4-2 shows tables defined that do not contain data relating to Census records, but rather support the functionality of the software. An example would be the Users table that stores information on who is authorized to access the application, and what their defined role is.



4-1 Schema design of core database tables

Below is a brief description of the data in the tables shown in figure 4-1.

Census Table:

The Census table contains the basic information on an affiliate, including name, country of citizenship, etc. Simply put, a Census record corresponds to a person (affiliate) who has a past or present relationship (affiliation) with the WCFIA.

Affiliation Table:

The Affiliation table contains information on the subsidiary Program that the Census record is affiliated with. The fields are: affiliation type (such as post-doc fellow, staff, etc.), affiliation Program, as well as the start and end dates of the affiliation. The Affiliation table will contain zero or more records for each Census record. (Although possible, it would be unusual to have a Census record with zero Affiliation records associated with it.)

Address Table:

The Address table contains zero or more addresses and related contact information for each Census record.

Bio Table:

The Bio table contains zero or more biographical sketch (es) for each Census record.

Research Interest:

The Research Interest table contains zero or more research interests descriptions for each Census record.

'AffilTypes' defines possible values for AffiliationType of 'Affiliation' Entities

AffilTypes	
PK	AffilTypeId
	AffilTypeName

'Programs' defines possible values for AffiliationType of 'Affiliation' Entities

Programs	
PK	ProgramId
	ProgramName

'Users' defines authorization data for end users

Users	
PK	UsersHUID
	UsersRole
	UsersFirstName
	UsersLastName

4-2 Schema design of auxiliary database tables

Below is a brief description of the data in the tables shown in figure 4-2.

Users Table:

The Users table contains information on who is authorized to access the application. With two noted exceptions, the names of permissible roles correspond to values in the Programs table. Typically, a user will have a role that is the name of a WCFIA Program, such as 'US-Japan Program.' The user will then be given rights to modify records of affiliates who have an affiliation with the specified program. In addition to the roles that correspond to the name of a program, there is also an 'admin' role, which has read and write access on all

records and a 'read-only' role. Referential integrity between entries in the 'AffiliateWCFIAProgram' field of the Users table, and the names of programs as defined in the Programs table is made by code in the AuthenticateServlet that checks to make sure unknown role types are not specified in the role field of the Users table. If a 'UsersRole' field in the Users table does not correspond to a valid program name as defined in the Programs table and is not 'admin' or 'read-only' the users will be forwarded to an error page. The code that does this can be found in the isRoleValid () method of the AuthenticateServlet class. This would most likely happen when an administrator mistypes a program name when entering a new user in the Users table.

AffilTypes Table:

The AffilTypes table contains a list of affiliation types. There are many kinds of affiliations. This list will change over time as new types of affiliations are created. Since this list is stored in a table, all dropdown lists in JSP pages are dynamically generated based on the current information in the table. For historical reasons, the list of acceptable affiliation types is rather long, as there have been many different affiliation types over the years.

Programs Table:

This table lists program (a.k.a. subsidiary department) types, to be dynamically generated in the drop down lists. If a new program is created, it will be added to this table and will be dynamically integrated into the application. The program names specified in this table will also correspond with 'UsersRole' values in the Users table. As previously mentioned, referential integrity is ensured by code in the AuthenticateServlet class, the Servlet that generates the session bean with information on the logged in user extracted from the Users Table. If the 'Role' specified is not 'admin', 'read-only', or an exact match of one of the

entries in the program name of the Programs table, then the user will be forwarded to the error page with an appropriate message displayed.

Role-based authorization based on schema

The role-based nature of the application is central to the design and priorities as described so far. Since this functionality is dependent on the table schema as described in this chapter, I will also provide a brief general description of how the role-based scheme works here, providing technical details of actual implementation later in the Security chapter.

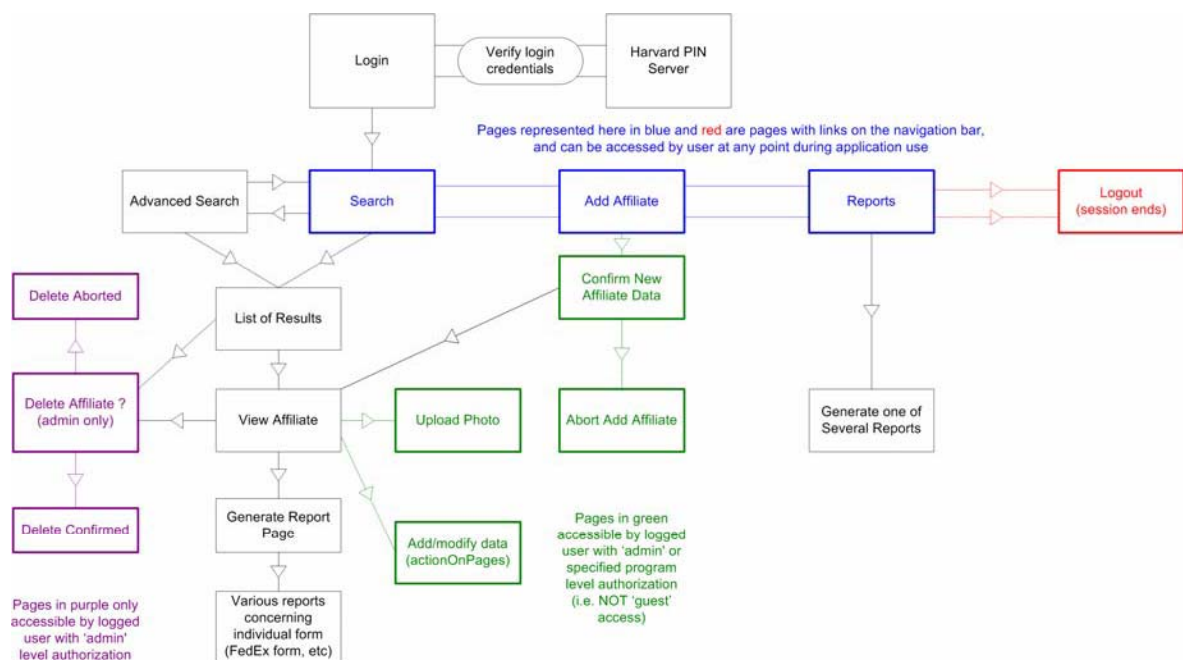
As we have seen, every affiliate in the database represented by a Census record has one or more Affiliation records, each with a 'WCFIA Program' field. Each logged in user has a role that is equivalent to one of the Affiliation program names, or is 'admin,' providing global authority on all records, or 'read-only.' Since an affiliate may have multiple past affiliations with various programs, it is important that logged in users (i.e. administrators from these multiple departments), each have the ability to edit the record. Technically, this is done by storing a list of the primary keys of the affiliates a logged in user should have edit access to in a list attribute in the session bean associated with the logged in user role. If a logged in user deletes the affiliation that associates that record with their role, they will lose their future edit access to that record.

AffilDB uses JavaBeans to encapsulate data from the tables, and make it available to the program logic. There are JavaBean classes that correspond to most of the tables used in the application. One such class is UsersBean which represents data on the current logged in user, and has attributes with data taken directly from the Users table. UsersBean keeps track of which CensusRecord (and related data) should be editable by the logged in user. In addition to the attributes found in the table, the UsersBean also has an attribute

called `accessList`, which contains a list of numbers that correspond to the `CensusIDs` of Census records in the database that have affiliations corresponding to the role of the user. Within the code of the `UserBean` a database connection is made and a query generates the `accessList`. In a JSP page, the `CensusID` of a Census record can be passed to the `UsersBean`, and a check can be made to see if it is a record that that the user should have the capability to edit.

Chapter 5 User Interface

Since “ease of use” is one of the core principles of the design, an easy to understand user interface that provides the user with quick access any of the major functionalities of the program has been implemented. The following flow chart (Figure 5-1) specifies the order of action that can be taken.

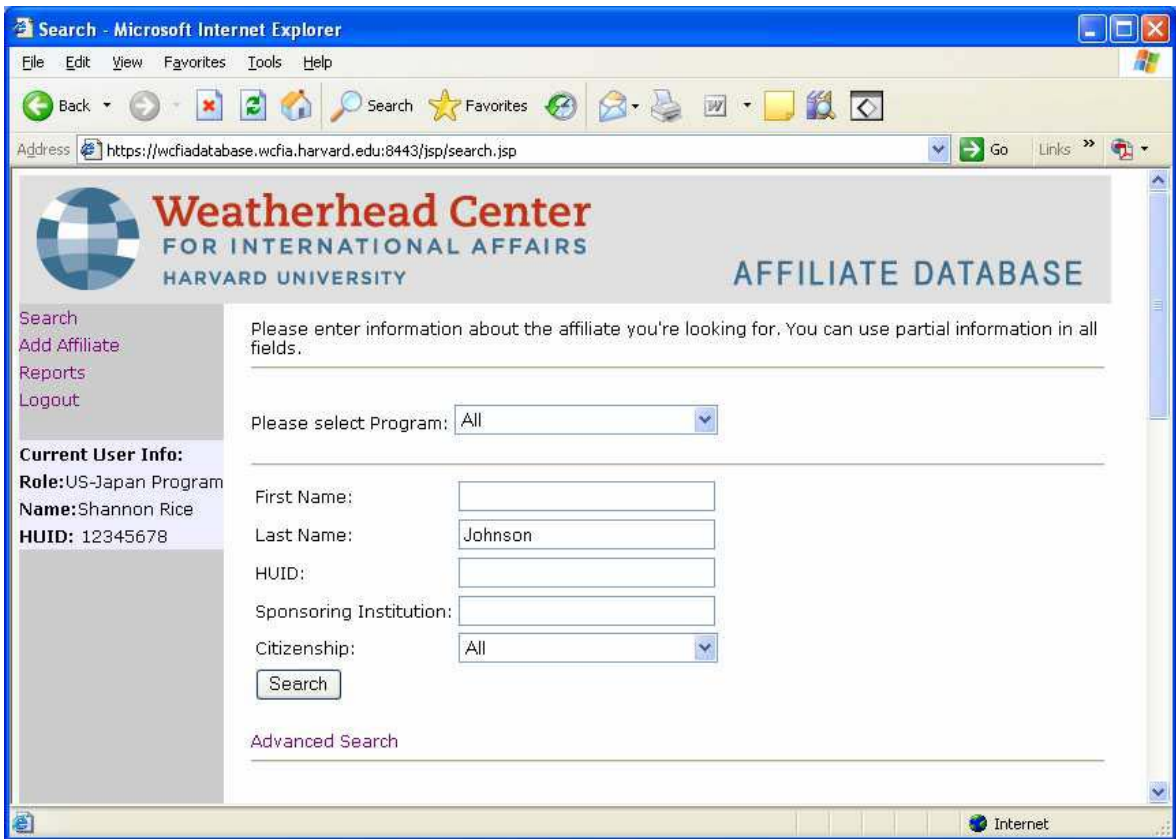


5-1 Chart of flow of control of AffilDB

In chart 5.1 above, the colors indicate the level of authorization, as well as possible paths a user may take to traverse the AffilDB. For example, only a user with an ‘admin’ role will be able to access the pages represented above in purple. Pages represented in green are accessible by logged users with ‘admin’ or specified program level authorization

(i.e. NOT 'read-only' level access). The pages represented in black are available to all users. Blue and red represent pages that are included on the navigation bar.

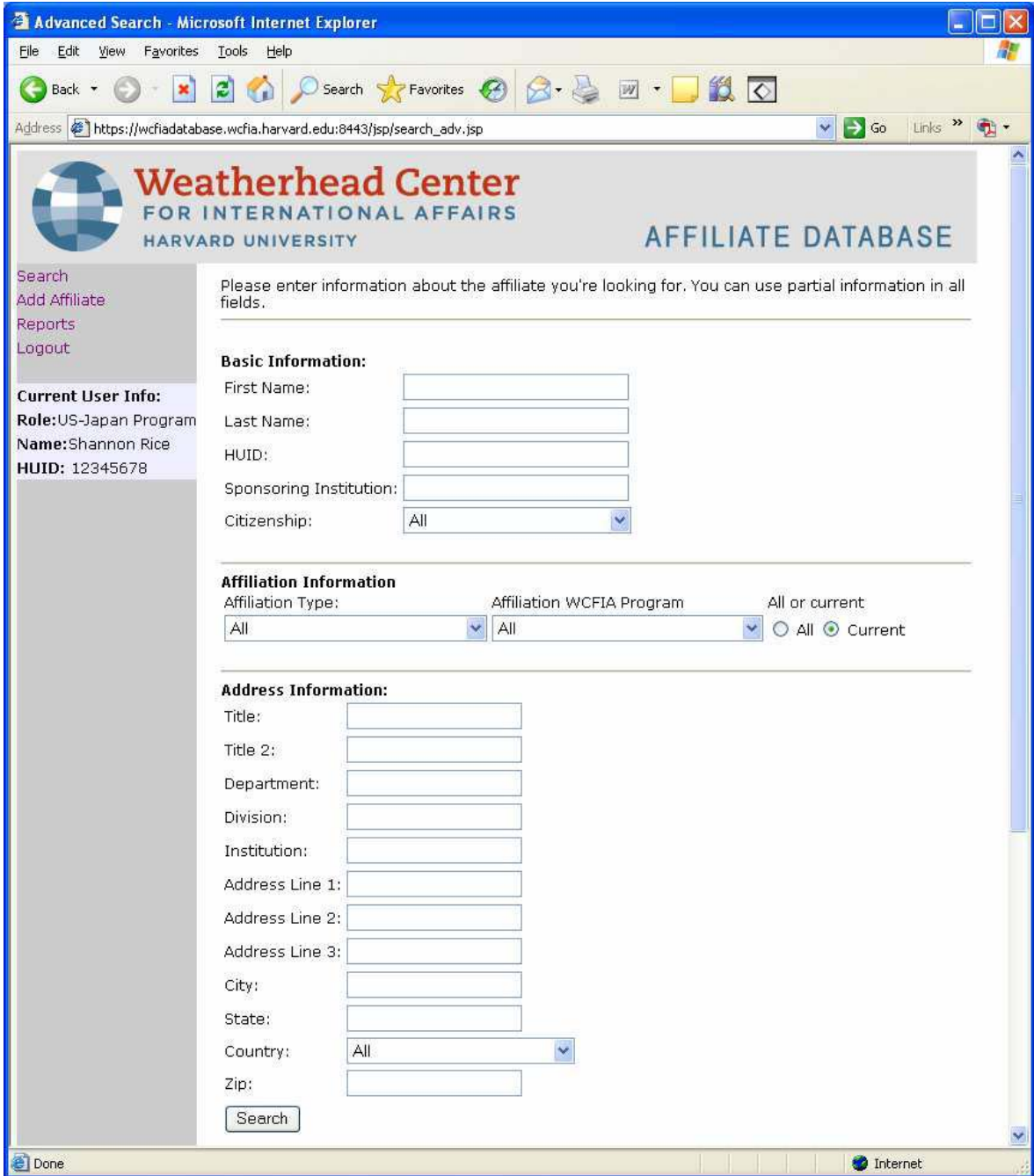
The screenshot in figure 5-2 illustrates how the application appears in a user's Web browser. There is a navigation bar on the left-hand side of the screen with the basic selections: Search, Add Affiliate, Reports and Logout. The navigation bar is always available, and the user may jump to any of these four major pages at any time. Information on the user currently logged in also appears on the shaded area in the navigation bar labeled "Current User Info." In the figure below, the logged in user has 'US-Japan Program' program privileges.



5-2 Screenshot of search page

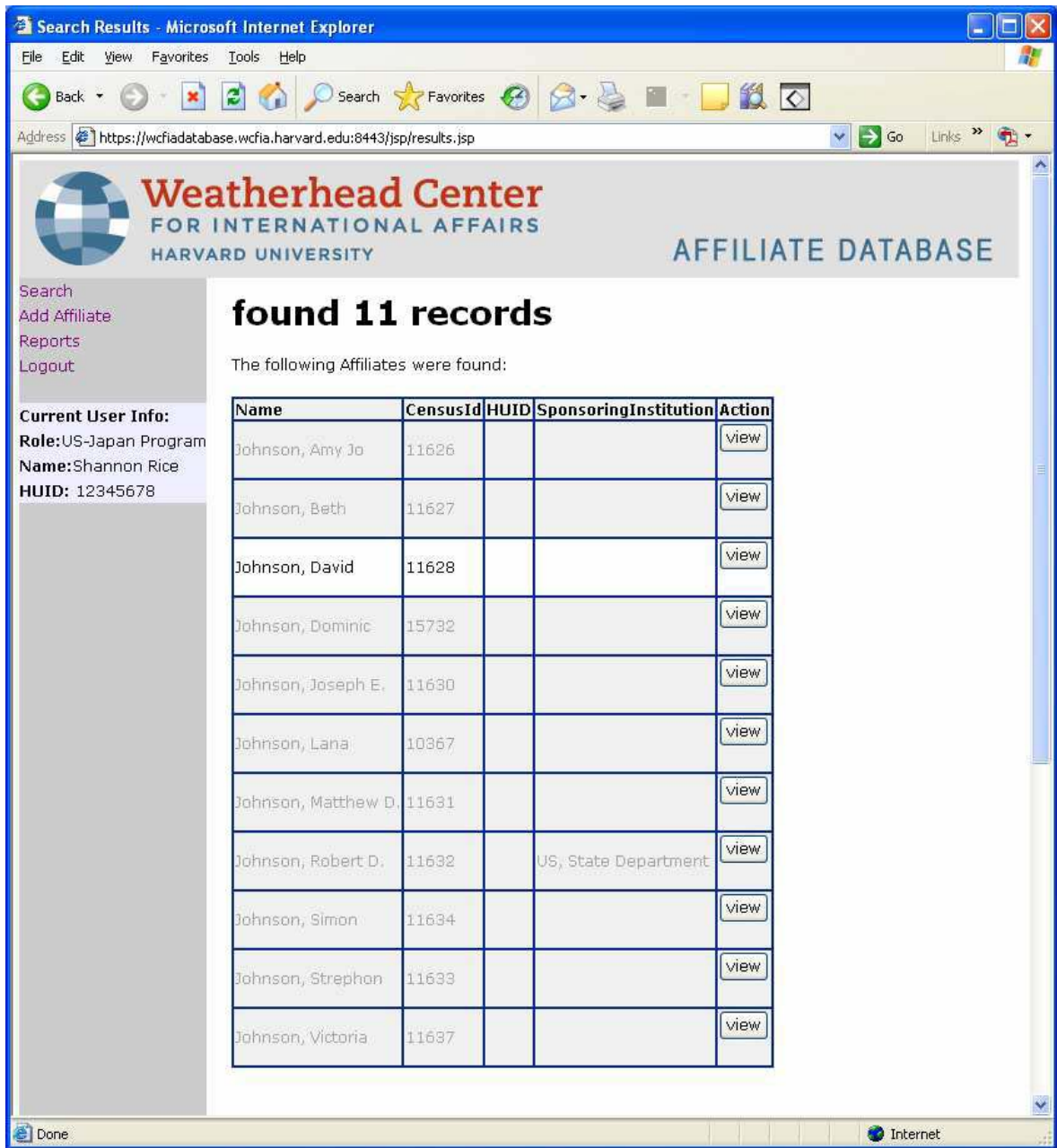
The search button on the navigation bar will bring you to the basic search page in figure 5.2. There is also an advanced search page, with a number of search parameter

options for the user. The purpose is to make advanced search functionality available in an unobtrusive way for the users that need to conduct more sophisticated queries. The user can access the advanced search page shown in figure 5-3 by following the link at the bottom of the basic search page.



5-3 Screenshot of advanced search page

Figure 5-3 shows the advanced search page that allows a user to make more sophisticated queries on the database.

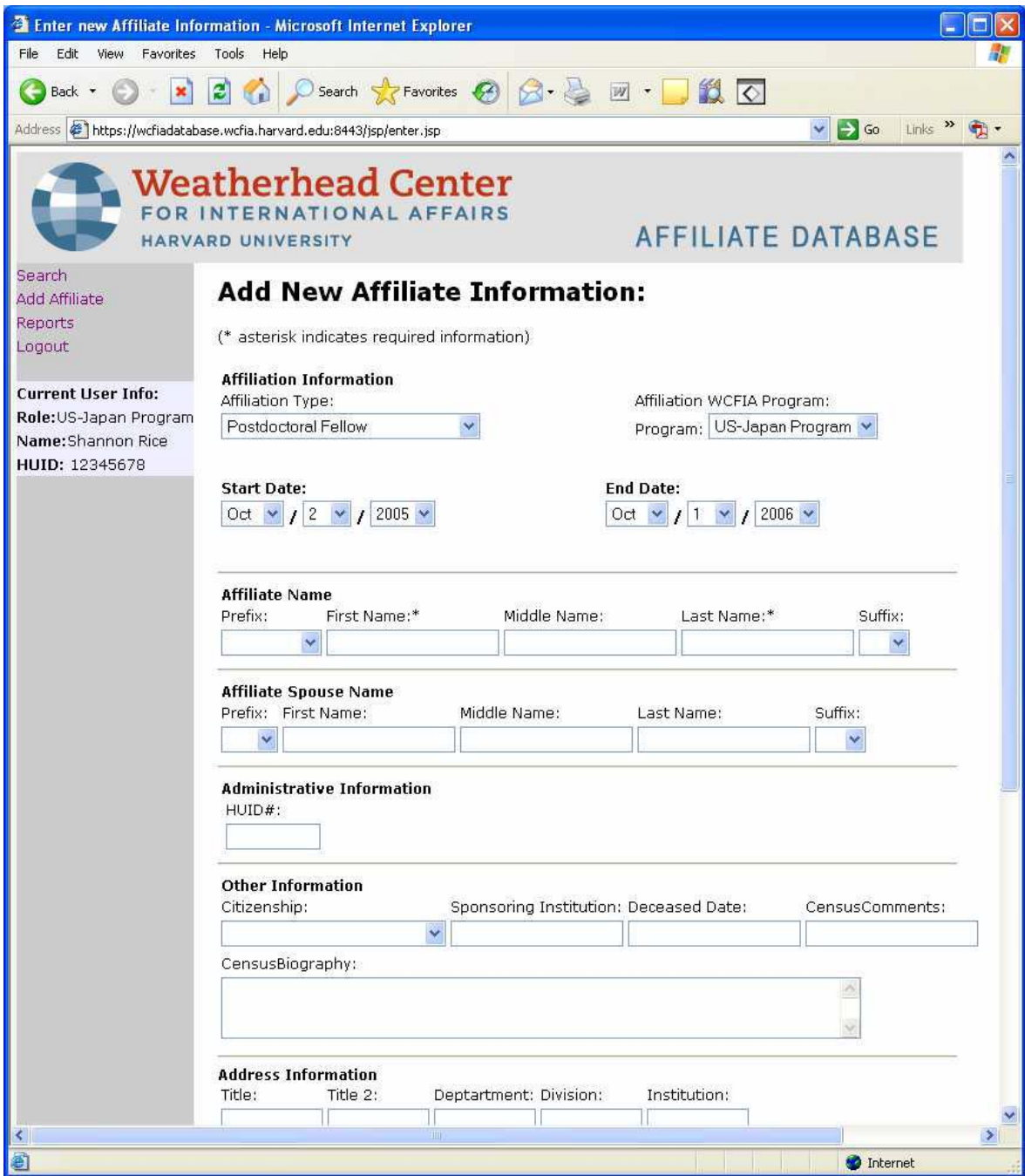


5-4 Screenshot of search results with both editable and non-editable results

Figure 5-4 shows the results of the search from the screenshot of the basic search of figure 5-2. The search was on all people in the database with “Johnson” in the last name.

Eleven affiliates (Census records) were pulled up. Since the user is of the role US-Japan Program, the affiliate that has an affiliation with the US-Japan Program, David Johnson, appears with a white background, and the affiliates without an affiliation of US-Japan Program appear with a gray background. The logged in user will have the option to edit all fields, as well as upload a graphic image that will be associated with the Census record associated with the affiliate by the name David Johnson should they click on the “view” button next to his name.

Figure 5-5 shows the add affiliate page. If a logged in user has a role that corresponds to a program, then the only types of affiliates that they are allowed to enter are affiliates with the WCFIA Program that is equivalent to their defined role. When the form that appears on this page is processed, a new record is added to the Census table, a new record is added to the Address table, and a new record is added to the Affiliation table. The start and end date of the affiliation, are selected on this page. The screenshot below was taken on October 2, 2005, so the default start date is October 2, 2005, and the default end date is October 1, 2006. Although the user may set any dates he or she chooses, the default start date will be the current date, and the default end date will be 365 days in the future from the current date.



5-5 Screenshot of add affiliate page

The view individual page in figure 5-6 is reached when the user does a search, and selects a “view” button that corresponds to a record that was retrieved from the database. In a sense, this page is the heart of the application as this is where updates and deletions to data will be made. A user who has edit access for the record being displayed, which

depends on the user's assigned role, will be given an option to upload a photo on this page, or to make edits to the record.

The functionality to upload an image file is based on code in the CensusBean. Actual images that are uploaded to the server are stored as files, with the name corresponding to the Primary Key (CensusCensusID) of the Census record of that affiliate. Code that will return the name if the photo associated with a Census record, if such a photo has been uploaded into the server, is in getPhotoName() method of the AffilBean class. This allows a JSP page to easily include an html tag, and display the photo associated with a Census record. If no file is available, the name of the default image, a .gif file that has the lettering "no photo available" is returned.

```
/**
 * This method will return the path to the photo associated
 * with this associate (including, possibly, the default "no_photo"
 * image)
 **/
public String getPhotoName() {

    String myFilename = this.getCensusCensusId()+".gif";
    String myPath = new String(image_path+myFilename);
    File imgFileObject = new File(myPath);
    if (imgFileObject.exists() && imgFileObject.isFile())
        return myFileName;
    else
        return "no_photo.gif";
}
```

By encapsulating the code within the JavaBean, it is easily accessible using JSP tags. Similarly, when a Census record is deleted by a user with 'admin' level privileges, AffilDB will use a JSP setProperty tag to trigger the code that will remove the image file from the file system, since the actual image is not stored directly in the database.

```
/**
 * This method is simply used to delete a photo associated with an
 * affiliate
 ***/
public void setDeleteImage(String foo) {

    String myFilename = this.getCensusCensusId()+".gif";
    String myPath = new String(image_path"+myFilename);
```

```
File imgFileObject = new File(myPath);  
  
if (imgFileObject.exists() && imgFileObject.isFile())  
    imgFileObject.delete();  
}
```

The choice of .gif as the default file type is based on the fact that the organization already has image files of all current affiliates stored in this format on the Web server. Were I to make the choice independently, I would have chosen the .png format, as it is designed to render images, such as photos, that have gradual variations of many colors. (The .gif format is more effective a compressing image files that have large blocks of pixels of a small number of colors.)


Weatherhead Center FOR INTERNATIONAL AFFAIRS HARVARD UNIVERSITY

AFFILIATE DATABASE

generate report upload photo <-- Back

Mr. Shannon Rice Basic Data

Update Basic Data

Name:	Mr. Shannon Rice	
HUID:		
Citizenship:	U.S.A.	
Sponsoring Institution:		
Spouse:		
Comments:		
Deceased:	FALSE	
Deceased Date:		
Record Last Modified:	Aug 18, 2005	

Addresses add Address

Address	About Address	Last Modified	Action
Title: Program Coordinator Title2: Program on U.S. - Japan Relations Dept: Weatherhead Center for International Affairs Division: Institution: ----- Line 1:(or rm#) 61 Kirkland Street Line 2:(or mailbox) Line 3: CityStateZip(or Univ) UNIVERSITY MAIL Country: ----- Address Tel: 617-495-1890 Address Fax: 617-495-4921 Address Email: srice@wcfia.harvard.edu	Primary IsGood:	Aug 21, 2005	update delete

delete all Addresses

Affiliations add Affiliation

Affiliation Type	Affiliation WCFIA Program	Start Date	End Date	Last Modified	Action
------------------	---------------------------	------------	----------	---------------	--------

5-6 Screenshot of view individual page

Figure 5.7 illustrates the reports page, which has links to other pages with forms that allow user to submit custom designed queries to the database and create reports in PDF format or tab delimited format based on selected parameters. The PDF files will print mailing labels in an Avery format. The tab delimited files provide an easy way for a user to

download a subset of data from the database which can be imported into mail-merge capable software to print a series of form letters or to perform other specific business functions.



5-7 Screenshot of reports page

Each of the links shown in the reports page shown in figure 5.7 refers to a URL of a Servlet that will take parameters and generate a PDF or tab delimited format. The 'What's this?' links refer to pop-up windows with a description of the report to be generated.

The links to various reports pages allow for extensibility should new report types be added.

Chapter 6 Technology Stack

Due to the high level of performance many open source software projects have achieved, as well as due to WCFIA's financial constraints, an open source technology stack was selected for this project.

Operating System – Red Hat Linux 9 ¹

The Red Hat 9 distribution of Linux was our choice for the operating system. This decision was made by the director of information technology at WCFIA, who is familiar with this version of Red Hat. Any Linux distribution, or even a proprietary OS, would have worked sufficiently well. Since my application runs on top of the JVM, it is relatively independent of the OS. Other secondary reasons for not using a Microsoft OS was the cost of licensing, as well our perception that Microsoft operating systems tends to require more frequent security patches and updates than Linux.

DBMS – MySQL v. 3.23.54 ²

MySQL version 3.23.54 was selected as the open source DBMS to store and manage the data. MySQL is robust, functional and meets the needs of the AffilDB application. Since version 4.0 of MySQL is quite stable an upgrade to MySQL 4.0 is desirable. This would allow use of the MySQL administration client GUI tool, which requires version 4.0. Currently MySQL is administered using the command line tools

¹ Red Hat Website <<http://www.redhat.com>>

² MySQL Website <<http://www.mysql.com>>

provided MySQL 3.23.54. The queries used in my application are all supported sufficiently by version 3.23.

API - Java 2 Platform Enterprise Edition, v 1.4 ³

The version of Java used is the Java 2 Platform Enterprise Edition, v 1.4, the most recent stable version at the start of the project. It was important that I use version 1.4, as this is the first version of Java that supports the JavaServlet Pages 2.0 specification.

AffilDB utilizes key JSP 2.0 features, such as the Expression Language (EL) and JavaServer Pages Standard Tag Library (JSTL). I also developed a Custom Tag Library for some functionality in the application, such as handling some of the dropdown lists and formatting dates. These were also developed according to the JSP 2.0 specifications.

Java technology can run on all major operating systems, and using JDBC drivers, can easily interface with most database management systems. Since it is possible that WCFIA management may decide to use other platforms in the future, the choice of J2EE is appropriate. For example, if in the future there is a need to host on a Microsoft OS, or move our data over to an Oracle server, the portability provided by the JVM should simplify such migration.

Web Server – Tomcat Version 5.0

Tomcat 5.0 was used for the Web container and Web server for AffilDB. Tomcat is very easy to download and configure. The main reason for selecting Tomcat is that it is widely used, and is the de facto standard open source Servlet container.

“Apache Tomcat is the Servlet container that is used in the official Reference Implementation for the Java Servlet and JavaServer Pages technologies.”⁴

³ Java website: <<http://www.javasoft.com>>

AffilDB does require version Tomcat 5.0 or higher, as the specifications defined in JSP 2.0 are not implemented in versions below 5.0. As mentioned above, the main functionality I used was the Expression Language, as well as Custom Tags. (Although Custom Tags are available in JSP 1.2, I implement the Simple Tag interface introduced with the JSP 2.0 specification.

It is worth reiterating that although I chose this technology stack as described above, I could replace any layer in the stack with an alternative, as long as the alternative supports the JSP 2.0 specification, should the need arise.

⁴ Apache Jakarta Project Tomcat website: <<http://jakarta.apache.org/tomcat/>>

Chapter 7 Security

AffilDB contains sensitive information such as personal telephone numbers, postal and email addresses. While this data is not as highly sensitive as social security or credit card numbers, security is a very important concern. Users logging in to AffilDB are authenticated via services provided by the Harvard University PIN server. Once a user is authenticated as being a person with a Harvard ID, the AuthenticateServlet grants them role-based authority to access the application by checking their Harvard ID with entries in an internal table that contains information on who has been set up for access. A third security measure is the implementation of SSL to provide encryption to prevent a third party running packet sniffing software somewhere on the Internet from viewing the data.

Authentication

The authentication service that verifies users who log into AffilDB is provided by the Harvard PIN server, which is administered by Harvard University. AffilDB has been registered as an application with the University PIN administrators, and PIN services are integrated into AffilDB. The first page that a user selects to access the AffilDB has a link with the URL of the PIN server and when the user visits that page they are asked to enter their Harvard ID and their PIN. If they do so correctly, which will happen if a person with a valid HUID enters their registered PIN, the server will forward the request with an appended “ticket” in the query string containing several pieces of information, along with a “signature” that was created using the PGP encryption algorithm, encrypted with Harvard University’s private key to AffilDB. Note that any user with a valid Harvard ID, and who

uses their correct PIN number will be authenticated via this system. Whether or not they are granted authorization to use AffilDB after they are forwarded to the AffilDB application, is determined by AffilDB. These pieces of information are passed as part of the query string when the request is forwarded to the AffilDB AuthenticateServlet. AuthenticateServlet handles the verification of the ticket by referring to the Harvard public key, and testing whether the signature was created by the Harvard system. This prevents people from spoofing someone else by simply creating their own URL query string with someone else's Harvard ID.

If the user has a valid ID, and the ticket has been verified, then a query is done on the Users table in the database, to see if they have been granted a role. If they are not in the database, then they are forwarded to an error page and are not given any access to the application. If they are in the database, then a session bean is created, and they are given role appropriate access (i.e. they will be able to edit affiliate data for people who are in their department, if they are a department administrator). Details on the PIN application system can be found at the Harvard PIN Website⁵.

Authorization

The largest technical challenge was to develop an application that would allow users the flexibility needed update and modify data relating to affiliates that they are responsible for within their departments, but to restrict them from making changes to data related to affiliate records in the database associates with affiliates who are not part of their department. If an affiliate has multiple affiliations, that record can be modified by users with a role that corresponds to any of the affiliations.

⁵ http://www.huid.harvard.edu/reference/pin/app_usepin_overview.html

As you may recall from the discussion on the database schema, each affiliate (or Census record – recall that the table containing basic information pertaining to an individual is called the Census table) in the database may have one or more affiliation records. Each affiliation has a ‘WCFIA Program’ field, as well as a start date and an end date for that affiliation. AffilDB uses the data encapsulation capabilities provided by JavaBeans to implement the authorization functionality. As we mentioned, when a user logs in, based on their designation in the Users table of the database, they are assigned a role. This role does not change for the entire session that the user is logged in. The JSP pages check the value of the user role and display functionality as appropriate to the role. For example, the search page will make a check, and if a role is ‘admin’ then a delete button will be displayed, as illustrated in the code snippet below:

```
<c:if test="\${loggedUser.role == 'admin' }" >
    code that displays form and submit button to delete goes
    here...
</c:if>
```

Similarly, JSP pages that should only be accessed only by users with specific roles do a similar check, (using a server side include statement), which redirects unauthorized users to an error page.

There are cases where simply checking the ‘role’ of a user to determine whether access should be granted to that page is not sufficient. This would occur when the check to be made was dependent on the qualities of data involved. For example, a user with a role defined as ‘Program A’ should be allowed to access a page where they can update an address on Affiliate Mr. Smith, if and only if Mr. Smith has an affiliation (past or present) with Program A. If Mr. Smith doesn’t, and the staff member goes to a page and attempts to modify the data on Mr. Smith, the user should be redirected to an error page. (The above

scenario could happen if a sophisticated user attempted to manually pass as a parameter the CensusID of an affiliate for whom they do not have access as part of the query string.)

From a programming standpoint, one of the most interesting challenges of the project was developing the algorithm to provide role-based authorization. The solution I came up with was to store a list indicating which records should be editable by the user in that user's session bean. Each user session bean has an accessList, which I implemented as a Tree Set. The TreeSet, which holds items in order, was a good choice as speedy lookups are very important, since AffilDB would be making many lookups.

Below is the code in the UsersBean to set the editable list. In order for the access list to be able to be set from within a JSP, a string property must be specified, but since the string is never used, it is a "dummy" field.

```
public void setEditableList(String str) {
    /*****
    str is a dummy field
    To allow list to method to be called by JSP set property
    Command
    *****/
    accessList = new TreeSet();
    String sql ;
    if (this.role.equals("admin"))
        sql = "select CensusCensusId FROM Census";
    else
        sql = "select CensusCensusId, AffiliationType from Census,
            Affiliation "+"WHERE AffiliationCensusId = CensusCensusId
            AND AffiliationWCFIAProgram= '"+this.role+"'";

    Connection conn = null;
    java.sql.Statement stmt = null;
    ResultSet rs = null;
    ResultSetMetaData rsm = null;

    try{

        Class.forName("com.mysql.jdbc.Driver");
        String url="jdbc:mysql://localhost:3306/WCFIA";
        conn=DriverManager.getConnection(url,"user", "password");
        stmt = conn.createStatement();
        rs = stmt.executeQuery(sql);

        while( rs.next()){
            accessList.add((Object)rs.getString(1));
            System.out.println("*"+rs.getString(1)+"*");
        }
    }
}
```

```

        } //end while

    } finally {
        try{
            stmt.close();
            conn.close();
        }
        catch (SQLException sqle){ }
    } //end finally
}

```

Each time a search is made the list is repopulated using the following JSP snippet:

```

<jsp:setProperty name="loggedUser" property="editableList"
    value="anything" />

```

By repopulating the list each time a search request is made, real-time accuracy is guaranteed. If there were thousands of users, this approach would incur a significant performance cost. If that were the case, it would be preferable to track a list for each role in application scope. However, since the subsidiary departments are quite small, each with only one or possibly two departmental administrators likely to login, and be assigned that role, there was not much cost to storing these lists in session scope for each user.

When a search query has been sent, and list.jsp is outputting results, it does a check on each record retrieved and checks if it is editable and formats it accordingly (i.e. it is slightly grayed out, as in figure 5-4, if the user does not have authorization to modify the record).

The code in list.jsp that does the check is as follows:

```

<c:forEach items="${affilList.rows}" var="row">
    <%--
        -- The below code will use the loggedUser bean
        -- to check each entry to see whether it is a
        -- value that is a "constituent" of the logged in
        -- user.
        --%>
    <jsp:setProperty name="loggedUser" property="idToBeChecked"
        value="${fn:escapeXml(row.CensusCensusId)}" />
    <c:choose>
        <c:when test="${loggedUser.editable != true}" >
            <c:set var="type" scope="request" value="uneditable" />
        </c:when>
        <c:otherwise>

```

```

        <c:set var="type" scope="request" value="editable" />
    </c:otherwise>
</c:choose>
    <tr class="\${type}" >
    <td><font class="\${type}" >
        \${fn:escapeXml (row.CensusLastName) },
        \${fn:escapeXml (row.CensusFirstName) }
        \${fn:escapeXml (row.censusMiddleName) }
    </font></td>

    <td><font class="\${type}">
    {fn:escapeXml (row.CensusCensusId) } </font></td>
    <td><font class="\${type}" >
    \${fn:escapeXml (row.censusHUID) }</font></td>
    <td><font class="\${type}" >
    \${fn:escapeXml (row.censusSponsoringInstitution) }</font></td>
    <td>
        <form action="view_individual.jsp" method="POST">
            <input type="hidden" name="ccid"
            value="\${fn:escapeXml (row.CensusCensusId) }">
            <input type="submit" value="view">
        </form>
    </td>
<c:if test="\${ loggedUser.role == 'admin' }" >
    <td>
        <form onSubmit='return confirmDelete(this) '
        action="/jsp/delete.jsp" METHOD="POST" >
            <input type="hidden" name="confirmMsg" value="Are you
            sure you want to delete this record">
            <input type="hidden" name="ccid"
            value="\${fn:escapeXml (row.CensusCensusId) }">
            <input type="submit" value="delete">
        </form>
    </td>
</c:if>
</font>
</tr>
</c:forEach>

```

As the results are being generated, a check is made against the list, using the UsersBean, and the appropriate formatting is determined by whether or not a record is flagged as a record that the user has the right to edit. Additionally, users with the role of 'admin' are allowed to delete records.

Encryption

Although the Harvard PIN Server provides protection from unauthorized users from logging on and accessing data, what has been described so far does not prevent unauthorized people monitoring the Internet from viewing data passed in requests and

responses that are sent across the wire. In order to prevent this, I implemented SSL (Secure Sockets Layer) a standard protocol that allows data to be encrypted before it is sent over the network. Implementing SSL on the Tomcat Web-server involved using the Java cryptography tools to generate a keystore and certificate, and then modifying my server.xml configuration file to enable SSL connections.⁶

At the time of this writing, after a user logs in via the Harvard PIN application, the PIN server forwards them to an http page. This regular http page then automatically redirects the user to an https page, and all subsequent communication takes place over an SSL connection. Once the Harvard PIN Server configuration is updated to forward a user who successfully provides credentials to a SSL page, I will disable non-SSL communication on the Web Server entirely, which will disable the port handling unencrypted http requests, and will only accept encrypted requests using the https protocol.

⁶ Instructions taken from <<http://jakarta.apache.org/tomcat/tomcat-5.0-doc/ssl-howto.html> >

Chapter 8 Release Management Process

The release management process was a cycle of development, testing, and finally deployment in a production environment. After the initial design of AffilDB was completed on paper, there was a long development phase that lasted several months. Development was done on the Linux box running Red Hat 9 that would eventually become the production machine. After most of the required functionality was integrated into the software and it was deemed stable, key users who would eventually use the system were presented with the system, invited to test it, and initial comments were received. After the suggestions that were accepted were implemented, and minor bugs were fixed, data from the existing database was imported into the new AffilDB.

The next phase was to integrate the PIN server services into AffilDB, and adding the code to restrict access to machines running on the three WCFIA subnets. After this was set up, then the core users were again given access, and tested the system. (This code was later removed after SSL was implemented and there was no longer a need to limit access from the local network.) During this testing phase, reported bugs were fixed. Once the product was deemed stable, the data from the previous MS Access database was exported into tab delimited text files, and then imported into the MySQL database using shell scripts.

Since the existing data was in a database with a schema relatively similar to the schema of the new database, the process of data migration was not as cumbersome as is typically the case in such projects. I created Excel files with the columns of data to be transferred, converted these to tab delimited flat-files, then used shell and SQL scripts to

load the new data into the MySQL database. Some of the fields that were in the original database had been changed or removed in the AffilDB schema, so some data cleansing was necessary, which was done with Perl scripts. (However, one of the biggest problems with the data was simply that much of it was incorrect or outdated as a result of the database not being kept up to date.) After importing the data, from the existing database, making all of the data easily accessible to department administrators, and giving them the ability to update information, the situation should improve as people notice old data, and as they transfer existing data from their current systems into the new system

Once the new data was imported, key users were given access again. Minor tweaks and modifications of the AffilDB were conducted during weekend hours when the application was not in use. After all data was successfully transferred, the old Access database was adjusted to be 'read only' and users were notified of the change.

Currently, backups of the data are done twice a week, using mysqldump, a utility tool provided with MySQL, that creates a very large text file with the sql script needed to restore the data to the original state. This file is then transferred to a remote machine for storage. In the event of a hardware failure, or data corruption on the server, this file will be retrieved, and all the data will be imported.

Packaging of application

I was the sole developer working on a single machine running Tomcat 5.0 as a standalone Web server. Everything was packaged in a WAR file following J2EE convention. I did all of my development on a Tomcat 5.0 server. The AffilDB now in production is running on this same Linux box. Were this to be an application that would be widely distributed to multiple people who would be running it on their servers, more time

would have been invested in developing release packaging. In such a case, it would be worth using a packaging tool, such as Ant, to simplify the process for the people who need to configure the application in different environments.

Chapter 9 Third Party Software Libraries

Three features in AffilDB require the use of external libraries. Users requested ability to print labels for mailings, so a library that could generate PDF documents was needed. Users wanted to be able to upload photos of affiliates onto the server. This would require a library to handle the uploading functionality. Since AffilDB uses the Harvard PIN system, which depends in PGP encryption algorithms, a library that could decrypt/and decrypt was necessary to handle the PGP encryption. Here is a description of the libraries that I found to meet these needs, a brief description of the services they provide, and a brief description of the license of each.

Library for PDF File Generation

The iText library⁷ is an open source java library that provides the ability to print reports in PDF format. Report generating Servlets in AffilDB import classes from the iText library.

The iText library is available under the GNU Library General Public License.

Library for uploading files

The Java library used for uploading files was one I found referenced in the book, 'Java Servlet & JSP Cookbook' (1st Ed.) by Bruce Perry, and downloaded at the companion site to that book.⁸

⁷ <<http://www.lowagie.com/iText/>>

⁸ <<http://servets.com/cos>>

The library was written by Jason Hunter, and is available under the following license:

<http://servlets.com/cos/license.html>.

Library for encryption/decryption

The Cryptix library⁹ was used to handle the PGP encryption and decryption required to take advantage of the services offered by the Harvard PIN server.

The Cryptix library is available under the following license:

<http://www.cryptix.org/LICENSE.TXT>

Potential Risks with using third party libraries:

Integrating third party libraries into an application can be a risky endeavor, in that as the application is developed and changes over time, it may change in such a way that the third party library loses compatibility. For AffilDB, this risk is relatively low. The functionality that is provided by these libraries is relatively isolated, and they provide fairly straightforward capabilities, that, if necessary, could be replaced with other third party options. The Cryptix library is only used in the AuthenticateServlet class, and it simply does PGP encryption. This is a straightforward task, and should a compatibility problem arise, it the library could be replaced.

Similarly, the iText Library to produce the PDF reports are only used in two report generating Servlets, the program mailing Servlet and the internal mailing Servlet. If this library should have compatibility issues in the future, it would still be relatively easy to

⁹ <http://www.cryptix.org/>

replace the library with another PDF generating library, (either open-source or commercial).

Finally, the Servlet for uploading the photos is also only used in the UploadServlet class, and could easily be replaced by another vendor's product, or a Servlet written specifically for this application.

Chapter 10 Conclusion

AffilDB started as a concept, and finished according to plan as production software for Harvard's Weatherhead Center for International Affairs. During the year that it took to develop the product there were many challenges. The early phase of the project involved honing my skills in the technologies of J2EE and MySQL. Fortunately, I found helpful documentation. Hans Bergsten's excellent book, "JavaServer Pages" provided a useful roadmap for getting up to speed. Similarly, Paul DuBois's "MySQL", along with the documentation on the MySQL Website, provided well-written, in-depth technical documentation.

In addition to implementing the role-based functionality of the application, another significant technical challenge faced during this project was integrating the external services of the Harvard PIN server. Initially, I attempted to integrate a library developed by Harvard's software development team that can be imported into a J2EE application that provides much of the needed authentication and authorization services. However, integrating this library, which has many more features than my application needed, turned into quite a complicated project in itself. It did not work "out of the box" so to speak on my particular system. After doing a little research, I realized that the Harvard library was a wrapper for the Cryptix library, an open source Java library that provides PGP encryption and decryption which is what was required by AffilDB. I realized that it would be more effective for me to integrate Cryptix directly into AffilDB, and not spend time trying to get the configuration of the Harvard library set up. Reflecting on the decision to use the Cryptix library, I realized that one skill that I developed during the course of the project

was my ability to step back and assess whether I was effectively using my time, and change course if necessary.

Rather than having a small number of major technical challenges, this project had a very large number of minor challenges, and took much more time than I could have imagined at the beginning. From administering the MySQL DBMS, to working with Tomcat, and developing CCS and HTML pages with an interface that would be comfortable for the users, there were dozens of technical challenges, not to mention the human challenges of working with users and trying to strategically integrate their wishes and needs into AffilDB.

Most changes to the original design plan that were made during development were responses to users' requests. For example, the advanced search functionality and many of the reports were designed based on users' comments.

Another request that was made by users during development, and subsequently implemented, was the capability to have a dropdown list with a number of predefined selections, but also allow the user to type in data if none of the selections matches the data the user intends to enter. This functionality can be seen when adding a new address, or updating an existing one. In address line one there are three choices. The reason for this request is that most office addresses for new affiliates at WCFIA are at one of the three locations. However, not all of them will be. By searching on the internet I found and modified JavaScript code to provide this functionality, but it only works on Internet Explorer. Since this functionality is important, for now, only IE is supported.

Future Enhancements:

Although AffilDB meets the specifications laid out at the beginning of the project, there is room for further enhancements. What I have developed is a basic structure that can

and will grow with the needs of the organization. I integrated Cascading Style Sheets into AffilDB, and the result is fairly clean. However, a professional web designer could improve on some aspects. Drop-down lists could be improved by adding JavaScript to dynamically make selections based on user selections in other fields, for example.

One desirable feature would be a Web interface to be available to a user who logs in with the role 'admin' so that they could manipulate the data in the Users table, so that they can add or remove users as well as assign roles. Currently, this is must be done by accessing the MySQL database via the command line, or another tool that interfaces with MySQL. Having this capability would simplify the administration process.

A second desirable enhancement would be to allow self-update of information. For example, affiliates who have left the program could login, and update their address information.

I used Log4j¹⁰, an open-source Java library that provides logging capability. Implementing logging capability that would make it easier to analyze use patterns of AffilDB is a third enhancement that would improve AffilDB's ability to meet the needs of WCFIA.

Working on AffilDB was a great learning experience. While a relatively small scale, as far as software projects go, it was a large project considering that I acted as the project manager, database/systems administrator, software developer, and marketed the project to employees at WCFIA.

The experience of developing a full fledged J2EE Web application from the ground up was a great learning experience. I am interested in continuing to develop my skills in this area. The experience of designing the software and going through the process of

release management broadened my skills as a software engineer. I am proud of the product and that I was able to assess, and I dare say, meet the needs of colleagues at the WCFIA with a robust, extensible, and maintainable Web application.

¹⁰ <<http://logging.apache.org/log4j/docs/>>

References

Perry, B.W. (2004 1st Ed.) Java Servlet & JSP Cookbook. O'Reilly & Associates, Sebastopol, CA.

Apache Jakarta Tomcat 5 Servlet/JSP Container - SSL Configuration
<<http://jakarta.apache.org/tomcat/tomcat-5.0-doc/ssl-howto.html>> (cited September, 2005)

Apache Jakarta Project Tomcat website: <<http://jakarta.apache.org/tomcat/>> (cited September, 2005)

Cryptix <<http://www.cryptix.com>> (cited August, 2005)

Harvard PIN <http://www.huid.harvard.edu/reference/pin/app_usepin_overview.html> (cited August, 2005)

iText <<http://www.lowagie.com/iText>> (cited August, 2005)

MySQL <<http://www.mysql.com>> (cited August, 2005)

Servlets.com <<http://www.servlets.com/cos>> (cited August, 2005)

Appendix 1 Users Guide

Welcome the AffilDB database application. AffilDB (short for Affiliate Database Application), is a Web-based application designed to provide WCFIA employees an easy to use method to interact with data about the thousands of affiliates of the WCFIA.

I. Logging into the application:

To use the AffilDB you must have a Harvard ID and PIN. You must also be registered by the AffilDB administrator to use the application before you will be able to successfully log in. Every user that is given access to the application is assigned a role. After you successfully log in, you will see on the navigation bar on the left side of the screen a box labeled “Current User Info” that displays the name, Harvard ID and the role that has been granted to the user. The ‘role’ refers to the use’s level of authorization as described below.

admin:

A person with ‘admin’ level authorization is the ‘super user’ of the system, and may modify any data, add new records, or delete records.

program level:

A user with a program level authorization may view all records, and may modify records of affiliates that have one or more affiliations with that program. For example, a user with an ‘Olin Insitute’ role may modify any data on an affiliate who has a current or past affiliation with the Olin Insitute.

read-only:

A person with ‘read-only’ level authorization may view all records, but may not modify data.

II. Searching the database

The first page that a user sees when they log into the application is the basic search page. One may search on one of several parameters. Using a dropdown menu at the top of the page, a user may also choose to search from the entire database, or search from a subset of the records, based on the program affiliation. Since many program administrators are primarily interested in working with the data from the current group of affiliates in their respective program, if one is a program administrator, the default is that they will be searching from a subset of their current affiliates.

There is also an advanced search page, that allows users to perform more sophisticated queries on the data, including searching by affiliation type, and affiliation program.

III. Modifying data on an affiliate

Users with a Program level role may modify data on records that have an affiliation that is the same program as their role

IV. Deleting an affiliate record

Since AffilDB is meant to keep a record of all people who have had affiliations with the WCFIA, records are not meant to be deleted. If you have entered an affiliate, and have reason to have it deleted, then you must email a delete request to the database administrator. Please include the first and last name as well as the censusID number, and the reason for the delete request. (Reasons for a delete request might include a duplicate entry, or a mistaken entry.)

To prevent duplicate entry of records, if a user attempts to add a new affiliate that has both the first name and last name of a record that already exists in the database, then a notification message appears, along with a link to view that record.

V. Adding an affiliate

Users with program level authorization may add affiliates to the database, but they will be assigned a program affiliation equivalent to the user's role by default. In other words, a user with 'Olin Institute' as a role, will only have the option of 'Olin Institute' for the field AffiliationWCFIAProgram when adding a new affiliate.

If a user attempts to add an affiliate that shares both a first name and a last name with an affiliate that is already in the database, a notification message will display, and a link to view the information on the affiliate in the database. If they indeed want to add a new user with the same first and last name, they will need to contact the database administrator.

VI. Generating Reports

The AffilDB application allows generation of several customizable reports, in both PDF and tab delimited formats. Descriptions of the various reports are available by clicking on the 'reports' link of the navigator bar, and then clicking on "What's This" listed next to the name of each report.

VII. Known issues

Currently, only IE v. 6.0 or higher is supported.

VIII. Logging out

To end a session, a user should click 'logout' on the navigation bar. This will end the session.

Appendix 2 – Application Code

(Since this is an online version of my thesis which details software that is in production, I have not included code for security purposes. If you have questions, please contact Shannon Rice at shannon@thericestuff.com.)