An Online Student Portfolio System

A Manuscript

Submitted to

the Department of Computer Science

and the Faculty of the

University of Wisconsin-La Crosse

La Crosse, Wisconsin

by

Steven E. Reich

in Partial Fulfillment of the

Requirements for the Degree of

Master of Software Engineering

March, 2008

An Online Student Portfolio System

By Steven E. Reich

We recommend acceptance of this manuscript in partial fulfillment of this candidate's requirements for the degree of Master of Software Engineering in Computer Science. The candidate has completed the oral examination requirement of the capstone project for the degree.

______________________________	____________________
Dr. Kasi Periyasamy, Ph.D Examination Committee Chairperson	Date
______________________________	____________________
Dr. David Riley, Ph.D Examination Committee Member	Date
______________________________	____________________
Dr. Thomas Gendreau, Ph.D Examination Committee Member	Date

Abstract

REICH, STEVEN, E., “An Online Student Portfolio System”, Master of Software Engineering, December 2007, (Dr. Kasi Periyasamy, Dr. David Riley).

Becoming a teacher is a lengthy process for students in Wisconsin. One major requirement for state certification is that students complete a portfolio relating to the ten standards set by the Wisconsin Department of Public Instruction. This document describes the development of a software system designed to assist students at the University of Wisconsin-La Crosse in maintaining these portfolios and allowing faculty members to easily view and comment on each student’s files independent of face-to-face communication. The software is a web application to be run from a university server that was developed using JSP pages, a SQL server, and a file server. Also examined are several important decisions involving the structure and functionalities of the program as well as the communication between the developer and the customer over the course of development.

Acknowledgements

I would first of all like to thank my project advisors Dr. David Riley and Dr. Kasi Periyasamy, who have proved their worth many times over both as guides and problem solvers. Without their help, it would not have been possible to complete this task. Secondly, I would like to thank the various people who provided input from the education department, most especially the project sponsor Dr. Ali Ahmed, who was instrumental in helping mold the product. Finally, I would like to thank all the people at IT that helped make it possible to deploy this product.

Table of Contents

List of tables/figures

Glossary

Apache Tomcat

Apache Tomcat is a web server application developed by the Apache Software Foundation that comes bundled with some versions of NetBeans.

API

An Application Program Interface (API) is an interface that provides access to the operating system and other services.

Cookies

Cookies are small files stored on the user's computer by web sites, typically used to maintain login information and user settings. Some cookies are stored permanently, while others are deleted when the browser is closed.

IEEE

The Institute of Electrical and Electronics Engineers is a professional organization that establishes some standards, publishes a variety of journals, and sponsors conferences.

JDBC

Java Database Connectivity (JDBC) is an API that provides Java applications with the ability to interface with standard database architectures.

JSP

Java Server Pages is a technology that allows for the use of Java code within web pages, which is compiled upon loading of the page to produce the end result visible to the user.

NetBeans

NetBeans is an integrated development environment created by Sun Microsystems for the creation of Java applications, both stand-alone and web-based. It includes tools for writing, compiling, and debugging Java code.

SQL

Server Query Language is a language used for accessing and modifying databases.

UML

Unified Modeling Language, currently maintained by the Object Management Group, is a collection of standard graphical metalanguages used across the software industry, allowing developers to design programming constructs before writing code for them. UML diagrams include class diagrams, use case diagrams.

1. Background Information

The process of becoming a professional educator in the state of Wisconsin is lengthy one, involving numerous steps and milestones, not the least of which are the ten standards set forth by the Wisconsin Department of Public Instruction, which students are required to complete in order to become certified to teach in the state. The collection of information stored in several files relevant to these standards for a given student is referred to as a portfolio.

There are two types of files involved in student portfolios: artifacts and reflections. Artifacts are files that represent a student's academic progress towards standards completion. Reflections include the student's comments on the artifact or artifacts submitted for that standard.

Currently, students at the University of Wisconsin – La Crosse (UW-L) are responsible for maintaining their own portfolios. They must keep a copy of all files and provide the appropriate files to their advisors for approval at various times. The approval process is performed on paper copies and requires face-to-face meetings between students and faculty. This process has three primary disadvantages: 1) Students must schedule meetings with their advisors to review these files to prior to approval, 2) it is virtually impossible for the student to make progress on a portfolio when the school is not in session, and 3) the lack of an available backup of files leaves students in a vulnerable position if something happens to their computer.

While several commercial packages were considered, the Education Department at UW-L decided to offer the task of developing a portfolio system as a capstone project. While buying a commercial product would likely have been a faster way to get what the department needed, it also would have cost more and might not be as well-suited as a product specifically designed for the university.

2. Life cycle models

A variety of development models exist for software development, each with its own advantages and disadvantages. The one chosen for this project was a waterfall model with some prototyping elements. The development process followed a prototyping model in that the first version was developed on a computer owned by the student and the final version was deployed on a campus server. In addition, the waterfall sequence (requirements gathering, design, and implementation) was used to develop each version. This approach allowed the developer to easily create the initial version on his own computer and then move it to the campus server for deployment.

The main advantage of the prototyping model is that it produces a visible product within a short period of time. This helps give the customer a better idea of what the final product will look and feel like before the project is truly complete. The major problem with the prototyping model is that the product is often shown in an incomplete or rough state, possibly negatively affecting the customer's opinion of the project.

By contrast, the waterfall model has been widely used for many years due to its proven record of success. By carefully completing each stage of development before moving on to the next, the model seeks to reduce the number of potential errors and defects within the project by catching them early in the development process before they have a chance to propagate through the system. Additionally, the waterfall model produces a group of documents that can be used for later reference when updating the product or even developing a new product. The main disadvantage with the waterfall model is that it has problems with projects where requirements shift rapidly.

3. Requirements

The initial stage of any project is requirements phase. In this phase, the customers (in this case, the UW-L Education Department) and developers establish what the software is supposed to do (but not how it is supposed to do it) in terms desired features. The phase typically begins with a meeting between the developers and the customers and then the developers write a requirements document detailing the features of the product.

Meetings were held in the spring of 2007 to determine the requirements for the portfolio manager. During a meeting with education faculty and students, a variety of functionalities were proposed, ranging from simple storage and retrieval of student files through highly advanced interaction between faculty and students.

The initial scope of the project was defined: The software needed to support archival of student files. However, consultation with Education Department faculty produced a wide array of additional ideas for features. Several specialized disciplines (namely Health and Physical Education) expressed a desire to tailor the application toward their programs. Additionally, in was requested that other academic data be included. It was clear that features would have to be scaled down to what could be completed on schedule and within a scope appropriate for an MSE capstone project.

After discussion with some of the computer science faculty, it was decided that the proposed project was too much for a single student capstone project and the first version should include just core functionalities. In this version, the product would be a basic archival system that allowed for storage and retrieval of files with basic access for faculty members allowing for simple approval and disapproval of files with a basic commenting architecture. This would create a program useful to the students and faculty but also allow the application to be completed within the specified time frame.

3.1 Requirements development (Napkins)

Over the summer of 2007, the formal requirements document was developed using a tool called Napkins created by former UW-L student Ben Garbers. The Napkins tool allows for the easy compilation of requirements-related data into a single document that concisely describes what the software is intended to do on both a procedural as well as on a GUI level. The software is designed to produce a document conforming to IEEE standard 830-1993.

The use case diagram in Figure 1 illustrates the main functions of the software:

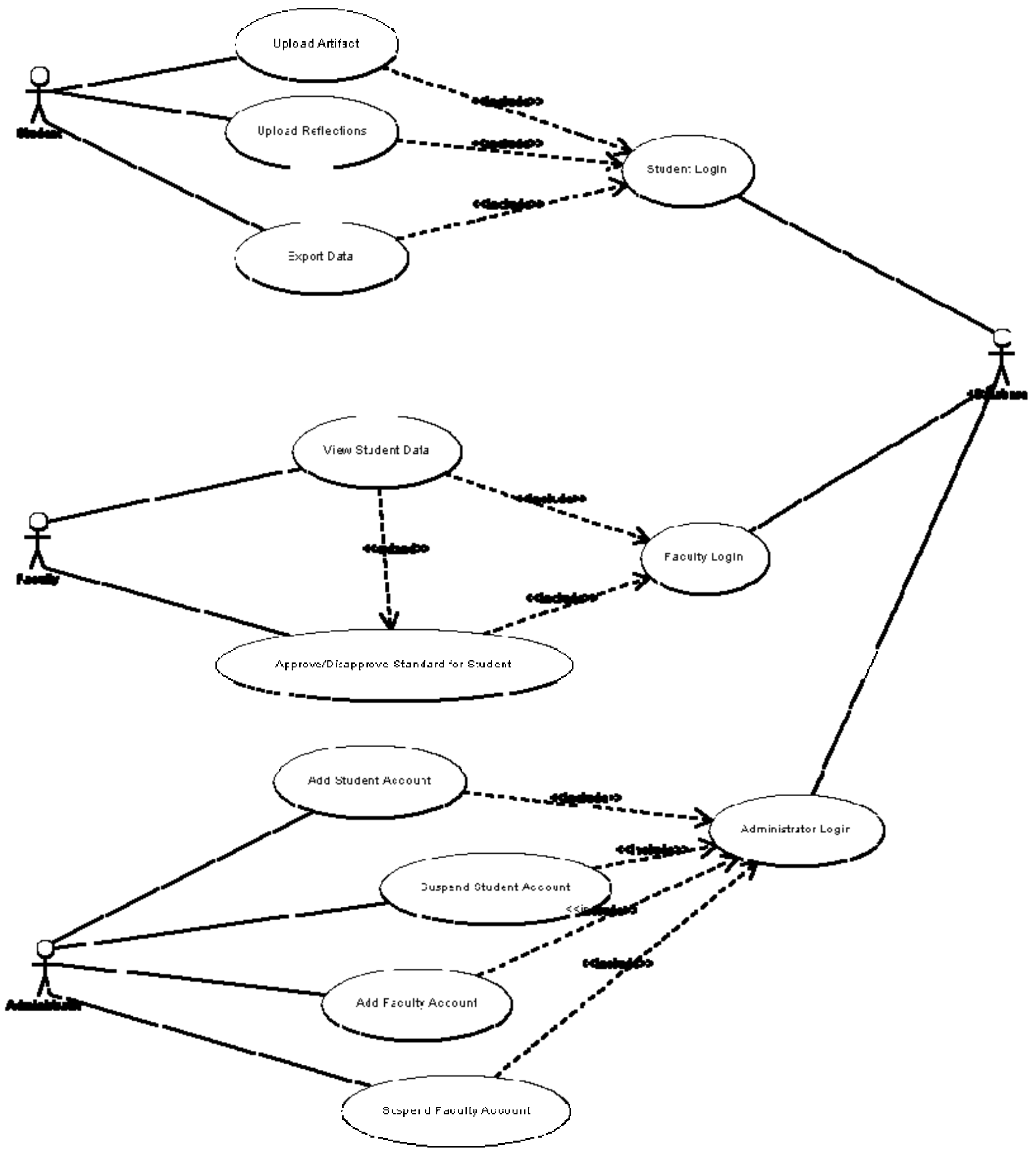


Figure 1. Use case diagram for the portfolio manager

The portfolio manager software has three user groups, each with its own set of privileges and responsibilities:

Students

Students are the main users of the system. They need to upload and download files as well as receive comments from instructors.

Faculty

Faculty members are responsible for approving or disapproving student files, as well as providing feedback to students about their files.

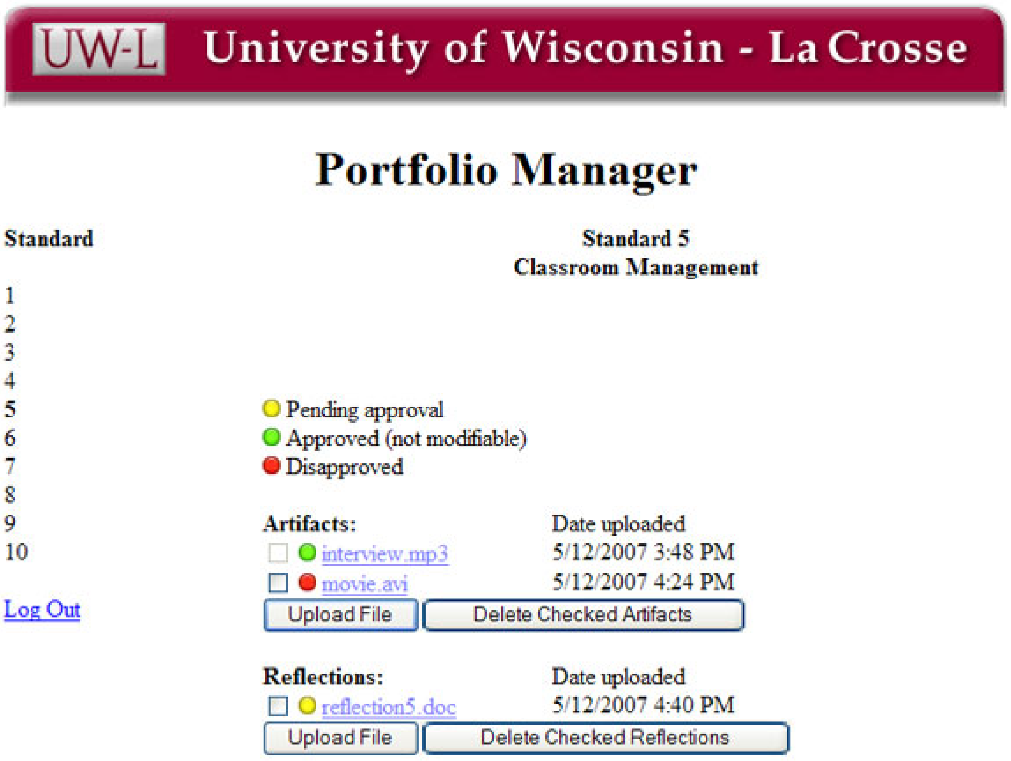
Administrator

The administrator must be able to activate and deactivate the system, as well as create and suspend student and faculty accounts.

3.2 User Interface

It was known from the start that the portfolio manager would be a web-based product, as it would have to be readily accessible from both on and off campus. The intended audience for the application could only be assumed to have beginning to intermediate computing knowledge, and therefore the interface would have to be kept relatively simple.

The basic interface, as seen in Figure 2 and appendix B, consists of a table with two rows and one or two columns (depending on the type of user). The first row contains a logo for UW-La Crosse and the second row contains all data relevant to the given page. On the student pages, the second row is subdivided into a column for page data on the right and a column for quick access to individual standards. This layout achieves the goal of keeping the interface simple and avoids putting too much information on any single page.


UW-L University of Wisconsin - La Crosse

Portfolio Manager

Standard

1
2
3
4
5
6
7
8
9
10

Log Out

Standard 5
Classroom Management

● Pending approval
● Approved (not modifiable)
● Disapproved

Artifacts:	Date uploaded
☐ ● interview.mp3	5/12/2007 3:48 PM
☐ ● movie.avi	5/12/2007 4:24 PM

Reflections:	Date uploaded
☐ ● reflection5.doc	5/12/2007 4:40 PM

Figure 2. View standard files

4. Design

The overall architecture of the application can be divided into three parts:

- A web server that manages the online interface
- A database server that maintains records of uploaded files
- A file server that stores and retrieves files uploaded by the student

Users see only the web interface, effectively hiding the database and file server from students and faculty.

Design began with the creation of a class diagram. The initial diagram (see Figure 3) contained five student-centric classes and a main class. Most of the functions of the system are routed through the main class. Several things should be noted about this diagram. Although the artifact and reflection classes are quite similar (and indeed share quite a bit of code), they do not have a direct subclass or shared subclass relationship as the potential gains of subclassing seemed to be outweighed by the potential pitfalls resulting from confusion between the classes. In hindsight, it would probably have been better to use the subclass structure since many of the scenarios the developer sought to avoid could simply have been avoided by only using the subclass objects in actual code.

Also of note is the fact that both the faculty and student classes contain only a single attribute and no methods other than their constructor. Full classes (rather than just strings) are supplied for potential future work.

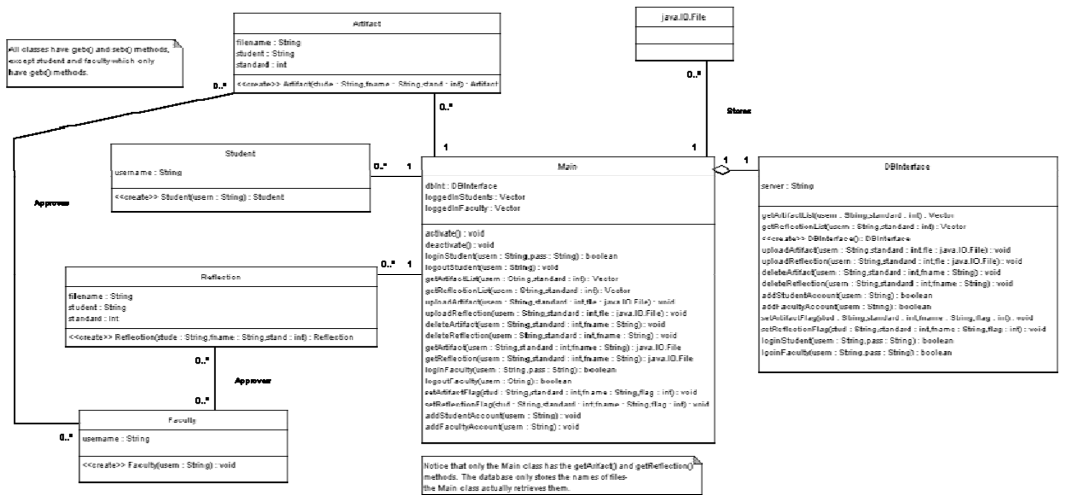

All classes have getx() and setx() methods, except student and faculty which only have getx() methods.
Artifact
filename : String
student : String
standard : int
<<create>> Artifact(stude : String,fname : String,stand : int) : Artifact
java.IO.File
Student
username : String
<<create>> Student(usern : String) : Student
Main
dbInt : DBInterface
loggedInStudents : Vector
loggedInFaculty : Vector
activate() : void
deactivate() : void
loginStudent(usern : String,pass : String) : boolean
logoutStudent(usern : String) : void
getArtifactList(usern : String,standard : int) : Vector
getReflectionList(usern : String,standard : int) : Vector
uploadArtifact(usern : String,standard : int,file : java.IO.File) : void
uploadReflection(usern : String,standard : int,file : java.IO.File) : void
deleteArtifact(usern : String,standard : int,fname : String) : void
deleteReflection(usern : String,standard : int,fname : String) : void
getArtifact(usern : String,standard : int,fname : String) : java.IO.File
getReflection(usern : String,standard : int,fname : String) : java.IO.File
loginFaculty(usern : String,pass : String) : boolean
logoutFaculty(usern : String) : boolean
setArtifactFlag(stud : String,standard : int,fname : String,flag : int) : void
setReflectionFlag(stud : String,standard : int,fname : String,flag : int) : void
addStudentAccount(usern : String) : void
addFacultyAccount(usern : String) : void
DBInterface
server : String
getArtifactList(usern : String,standard : int) : Vector
getReflectionList(usern : String,standard : int) : Vector
<<create>> DBInterface() : DBInterface
uploadArtifact(usern : String,standard : int,file : java.IO.File) : void
uploadReflection(usern : String,standard : int,file : java.IO.File) : void
deleteArtifact(usern : String,standard : int,fname : String) : void
deleteReflection(usern : String,standard : int,fname : String) : void
addStudentAccount(usern : String) : boolean
addFacultyAccount(usern : String) : boolean
setArtifactFlag(stud : String,standard : int,fname : String,flag : int) : void
setReflectionFlag(stud : String,standard : int,fname : String,flag : int) : void
loginStudent(usern : String,pass : String) : boolean
loginFaculty(usern : String,pass : String) : boolean
Reflection
filename : String
student : String
standard : int
<<create>> Reflection(stude : String,fname : String,stand : int) : Reflection
Faculty
username : String
<<create>> Faculty(usern : String) : void
Approves
Stores
0..*
1
Notice that only the Main class has the getArtifact() and getReflection() methods. The database only stores the names of files-the Main class actually retrieves them.

Figure 3. Class diagram from portfolio manager design document

4.1 User interface design

The user interface is relatively simple-a series of pages that handle interaction between the users and the rest of the application. The overall look and feel of the pages, see Appendix A, is intended to maintain consistency in appearance between each screen. This interface was fine-tuned through a series of meetings with Education Department faculty.

4.2 Database design

The relational database that stores data relevant to student files is very basic. The database uses a standard SQL query interface accessed using JDBC. The seven tables shown in figure 3 are used, and there are only four relationships. As shown in Figure 4, each table roughly corresponds to a class in the class diagram, and each relationship models a dependency within the system.

Table Name	Primary Key Field(s)	Class Represented
Artifacts	filename/student/standard	Artifact
Reflections	filename/student/standard	Reflection

eightDotFourStudent	username	Student
studentAccounts	username	Student
eightDotFourFaculty	username	Faculty
facultyAccounts	username	Faculty

Table Name: Name of the table in the database

Primary Key Field(s): Field(s) used as the primary key within the table. Each entry in the table must have a unique key field value (or combination of values if there is more than one primary key field).

Class Represented: The class for which the table is stores information.

Figure 4. List of tables and keys in the portfolio manager

The relationships (values from one table used in another) can be seen in Figure 4, where each database table is represented by a box and each relationship is represented by a black line (with cardinality on each end):

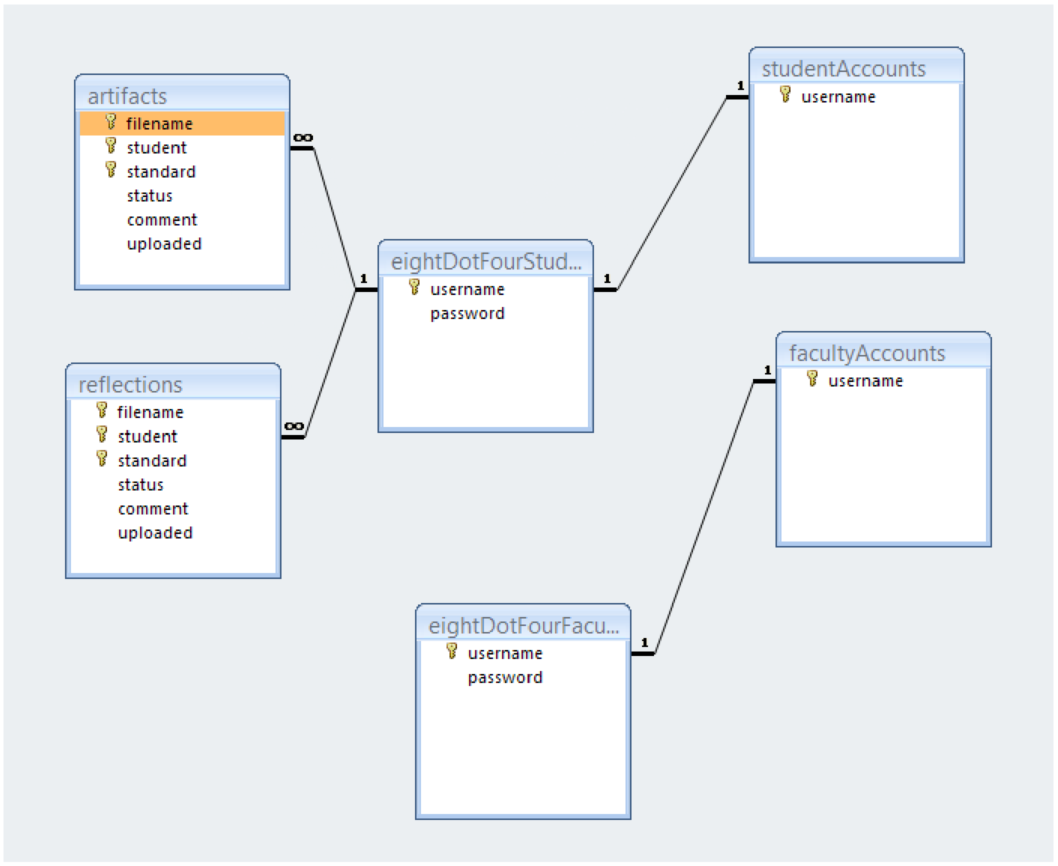


Figure 5 - Tables in the portfolio manager and their relationships

A relationship often indicates that a value from one field in one table should appear as a value in a field in another table. For example, every record in the artifacts table must have a value in the student field that is the same as a value in the username field in a record eightDotFourStudent table (thereby indicating that the artifact belongs to and was uploaded by that student). The numbers on then ends of the line indicate that each student may have an infinite number of artifacts stored in the system. Similarly, the relationship between eightDotFourStudent and student accounts is one-to-one, meaning that a username from the eightDotFourStudent table should appear no more than once in the studentAccounts table.

Figure 4 does not indicate the directionality (which tables' keys are being used as the foreign keys) of the relationships. For the portfolio manager, the eightDotFourStudent and eightDotFourFaculty tables are the originating tables in all relationships (that is, deleting an entry from the eightDotFourStudent table will also remove all entries using that username value in the artifacts, reflections, and studentAccounts tables).

Several significant details emerge from the relationships diagram. The relationship between students and their files is independent of the studentAccounts table. This decision is intentional as it makes it possible to suspend a student's account (preventing them from accessing or uploading files) without deleting all their files. Another significant point is that the artifact and reflection tables have the same fields and the same relationship with the eightDotFourStudent table. The two tables could have been combined with the addition of another key field to indicate whether the file is an artifact or a reflection, but it was decided to carry the division from the class diagram out to the database for the sake of consistency.

4.3 File server design

The final element of the software architecture is the file server, which stores and retrieves the actual artifact and reflection files. The file server module is also responsible for keeping track of student disk use.

As described in the requirements, each file must have a unique combination of filename, file type (artifact or reflection), standard number, and student username. To support this, the folder structure is as follows:

Drive:\portfolio\student username\standard number\artifact or reflection\filename.ext

Therefore, the artifact file "interview.txt" for standard 5 from student "doe.john" would be located at:

Drive:\portfolio\doe.john\5\artifact\interview.txt

When the file is copied onto the web server for download, it will be moved to:

```
root\TEMP\doe.john\5\artifact\interview.txt
```

where "root" is the main directory of the server. This structure avoids any issues relating to naming conflicts.

Keeping track of student use of disk space is another important function of the file server module. When a file upload is attempted, the server checks the amount of disk space the student is currently using and the size of the incoming file against the predetermined disk space limit (defined as a constant, in bytes, in the FileServer class). If the file fits within the allotted space, it is stored. If not, an error message is returned to the user.

4.4 Security design

While the portfolio manager dos not process highly sensitive information (as would be seen in a health care application), the data still requires a significant amount of safeguarding to maintain confidentiality. This need for security is carried over from the real-world domain where portfolio activities are generally private matters between students and faculty.

The most basic protection is the login system. At UW-L, each student and faculty member is uniquely authenticated by a username (also known as an 8.4, a rigid format containing the first eight letters of the user's last name and the first four letters of their first name) and password. In order to simplify the life of the users, many campus online services use this username for login purposes. However, simply knowing that a student or faculty member exists is not enough for the purposes of the portfolio manager, as the system must also know which students and faculty are authorized to use the system. These lists are maintained by the studentAccounts and facultyAccounts tables inside the database and may be edited by the administrator (as mentioned previously, deactivation of a student's account will only prevent the student from logging in-it does not delete the files they have stored). Every page checks if the user accessing the page is authorized, redirecting them to the login screen if he/she is not.

The second major concern is the security of artifact and reflection files. The system needs to keep these files in a secure location on the web server that is not accessible from other computers, moving them to the web server only when requested, and then removing them from the web server once they have been downloaded by the user. While the process sounds simple and straightforward, actually doing it proved quite difficult, as many attempts at solving the problem resulted in the file either being deleted before the user could access it or not being deleted at all. In the end, the most viable solution was to have the previous file deleted before the next one is retrieved, achieved by having each new file retrieval begin with a call the page’s jspDestroy() method, which was overridden in order to complete the task.

One last security feature has to do with the way the database is accessed. In most cases, an SQL statement is a string that the server parses and returns the requested material or performs the requested update. However, using appended strings in this manner permits SQL injection attacks. To mitigate against SQL injection attacks, the portfolio manager uses prepared statements with a specific number of parameters and expected parameter types, thus preventing unauthorized access and modification to database files.

4.5 Changes to the design

As actual coding began, changes to the class structure were required. One of the first changes was the conversion of the database interface class from an object class into one composed entirely of static methods. This permitted the functions inside the class to be implemented without initialization data from the calling code. The change to static methods also allowed for all of the database login code to be stored in a central location, which eased the transition from the prototype to the deployed version.

Another early design change was the elimination of the main class from the project. Due to the JSP architecture, each page effectively took care of the relevant actions previously assigned to the main class. As a result of this change, the system no longer kept track of which users were logged into the system at any given time. This was determined to be a reasonable change, as it was not necessary to keep track of this data for any mandated function of the product.

When implementation of the file server began, another class was included to manage storage, retrieval and deletion of files. The class also assisted in building links to files for the web pages and monitoring each student's use of disk space. Like the database interface class, this class contained only static methods, as no initialization parameters were required for operations.

After initial demonstrations of the web interface, several changes were made to the pages. In order to make the interface more usable, the font size was increased and various HTML elements were realigned. Additionally, several new features were added, including the ability for staff to view all of a student's files on one page.

A late major addition to the project was the ability to store three additional documents, including: A personal statement, an educational philosophy, and a resume. These documents were added as artifacts for DPI standards eleven through thirteen, avoiding the need to create a new set of tables in the database.

5. Testing

The portfolio manager offers several interesting challenges to the testing process. Unlike a stand-alone application, it has multiple interaction points and does not necessarily follow a predictable path of execution. It also frequently supports multiple users at any given time, raising many concurrency issues. Additionally, due to the open nature of the web architecture, careful testing must be done to ensure that the software blocks unauthorized access to the system. Finally, the disk space limit is difficult to test since it is difficult to find files of specific size.

While the portfolio manager poses some interesting problems as a result of being a web-based application, it does have some features that actually make it easier to test. Very little of the data sent into the system has to be interpreted in any way. Other than the login parameters and standard numbers, the software has only to take in data from the user and store it in the database or on the file server. Also, none of the tasks performed by the software requires more than two or three inputs to be directly supplied by the user, greatly reducing the number of required test cases for each action.

The portfolio manager was tested on two levels: First, unit testing was performed each time a new functionality was added to the project. Once all the major functionalities were added, the project was retested, also known as system testing, in its entirety to ensure that the project worked as desired.

Before the system tests, a full test document was created containing the details of every test to be performed. Each test case contained inputs as well as expected results. In all, over 250 cases were generated and compiled into a Microsoft Excel document, covering all functionalities implemented at the time. The tests were designed to cover at least single fault assumption (and where feasible, multiple fault assumption) for all functionalities within the product. [1]

6. Continuing work

6.1 Limitations

One key limitation of the current portfolio manager software is the lack of advanced communication between faculty and students. Currently, faculty members are limited to attaching brief comments to student files, and students are not able to directly respond to these comments through the system. Hopefully a later version will expand the program to allow for more complete communication support.

Another concern is the possible concurrency issues of the file retrieval portion of the software. In order to protect student privacy, files copied to the web server are deleted once they have been downloaded. This is achieved by deleting the old file before the new one is moved to the web server. Since the speedy deletion of the previous file cannot be guaranteed, there is a delay between when the new file is requested by the user and when the file begins to download. While this delay would likely be minimal for a small number of users, it is not known what it would be during peak usage times.

A side effect of the race condition potential is that files may occasionally not be deleted from the server after downloading. The file server cannot wait indefinitely to see if the previous file has been deleted before retrieving the new one, and so the possibility exists that the server may have to give up on deleting the old file entirely. Additionally, the old file will not be deleted if no new file request is made before the server crashes, is restarted, or is otherwise interrupted. As a result, the temporary directory on the web server may need to be periodically "cleaned out".

Another concern is the lack of full security on the faculty side. As it stands, any faculty member with an account may view any student’s files at any time. While this may prove helpful at times, it would be preferable to have multiple levels of faculty access so that only authorized faculty can view each student’s files.

One shortcoming of the current administrator interface is the lack of a mechanism to add student and faculty accounts in bulk. Currently, accounts can only be added one at a time. It would be preferable if the administrator could add a list of names, and even better if students could be added automatically to the system upon registration for a specific required class. Hopefully a later version will add this capability.

A related limitation of the administrator interface is the way the usernames of students and faculty are presented. Each list is presented as a pop-up menu (sorted alphabetically) on the administrator controls page. While this interface is fairly easy to implement on the programming side and should be sufficient for a small number of initial users, it will not scale effectively to large numbers of users. A later version of the product should redesign this portion of the interface.

One suggested change that was not completed involves the communication between students and faculty. As designed, the comments left by faculty members are associated directly with an artifact or reflection. As such, if the student is to delete the artifact or reflection file from the server, the comment is deleted with it. Although comments could be made more permanent, it would require a fundamental restructuring of the database and was deemed not worthwhile at this time.

It is hoped that eventually another graduate student will have the opportunity to extend the software to include a variety of communication features between students and faculty, possibly integrated with coursework and other academic items, transforming the product into a full student assessment tool.

6.2 Campus wide deployment

As a final step, the prototype will be adapted to run on the campus's web server. Several notable differences exist between the prototype version of the software and the final deployment platform, which will have to be smoothed out prior to making the product available to students. Most importantly, the database interface will have to be integrated into the existing campus database system. The campus uses an Oracle-based SQL client, as opposed to the Microsoft Access SQL client used in the prototype.

Like many large-scale projects, the portfolio manager will be rolled out in stages. Initially, the program will be used by a small group of education students. Eventually, the software will be rolled out to all students within the major. There are several reasons for not making the software immediately available to a large audience, one of the most important being the amount of resources the system uses. Although the total amount is not yet known, creating a brand new system and making it available to over one thousand users on a single day is sure to cause massive bandwidth usage with a massive detrimental effect on the campus network's performance.

7. Conclusion

The software product whose development is described in this paper is designed to simplify the process of creating and maintaining a portfolio for a student educator. By removing a substantial bottleneck in the portfolio-building process, both faculty and staff will be saved a significant amount of work. Additionally, students will be provided with a safe backup of their progress toward certification.

The portfolio manager system is a complete functional program. It uses a database, file server and JSP server to easily store and retrieve student files as well as allow faculty to comment on the files without requiring time-consuming face-to-face meetings. This is a significant step forward in usage of both time and resources.

The university will be moving the project onto the school's web servers for use, likely within the next year, eventually being used by over one thousand students. Moving the project to a deployment server should only require some modifications to the database interface and the file server, all of which should be minor.

8. Bibliography

1. P. C. Jorgensen, *Software Testing: A Craftsman's Approach, (2nd Edition)*, CRC Press, 2002.
2. S. Joshi, "SQL Injection Attack and Defense", SecurityDocs.com, http://www.securitydocs.com/library/3587, 2005.
3. M. Hall and L. Brown, *Core Servlets and Javaserver Pages: Core Technologies, Vol. 1 (2nd Edition)*, Prentice Hall, 2003.
4. Sun Microsystems, "Java^{TM} Platform Enterprise Edition, v 5.0 API Specifications", java.sun.com, http://java.sun.com/javaee/5/docs/api/, 2007.

9. Appendices

Appendix A. Screen shots from the portfolio manger interface

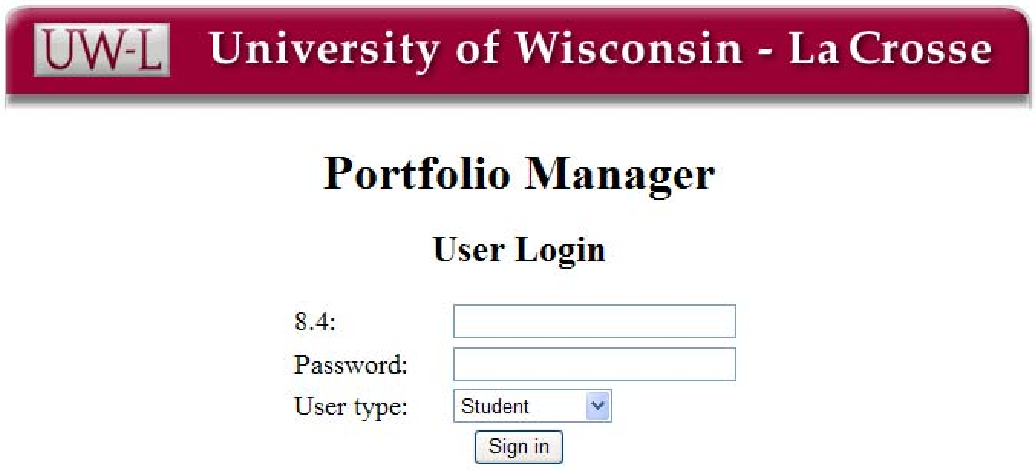
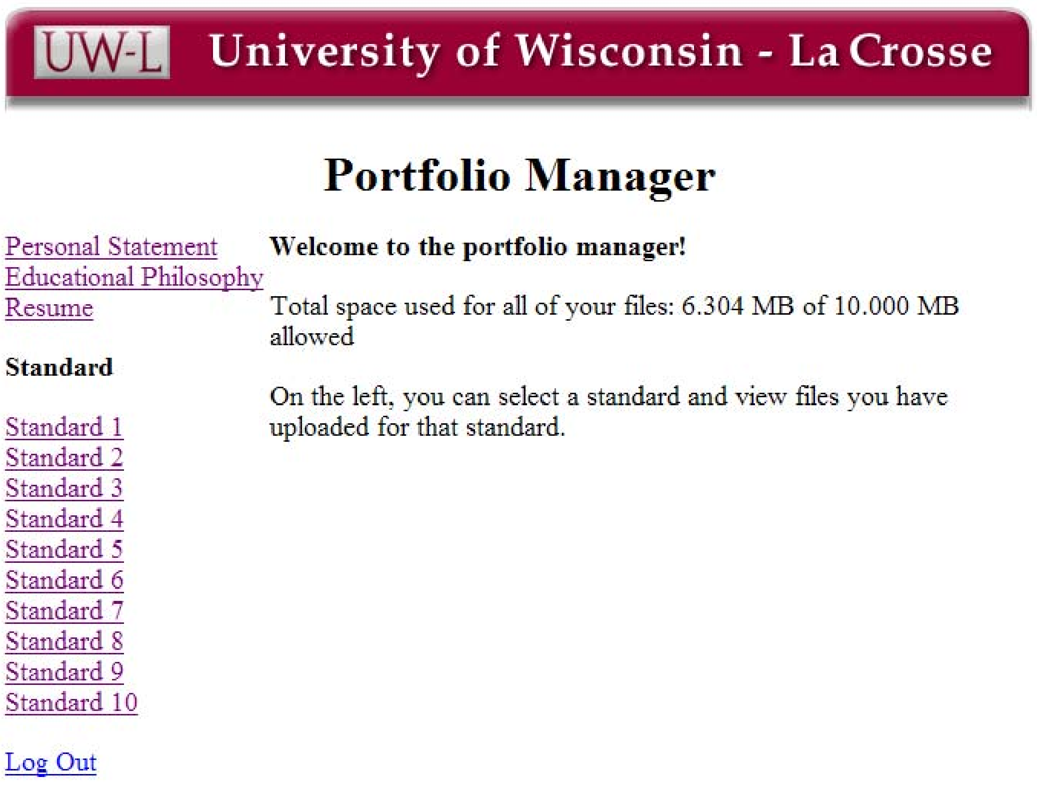


Figure 6. User login



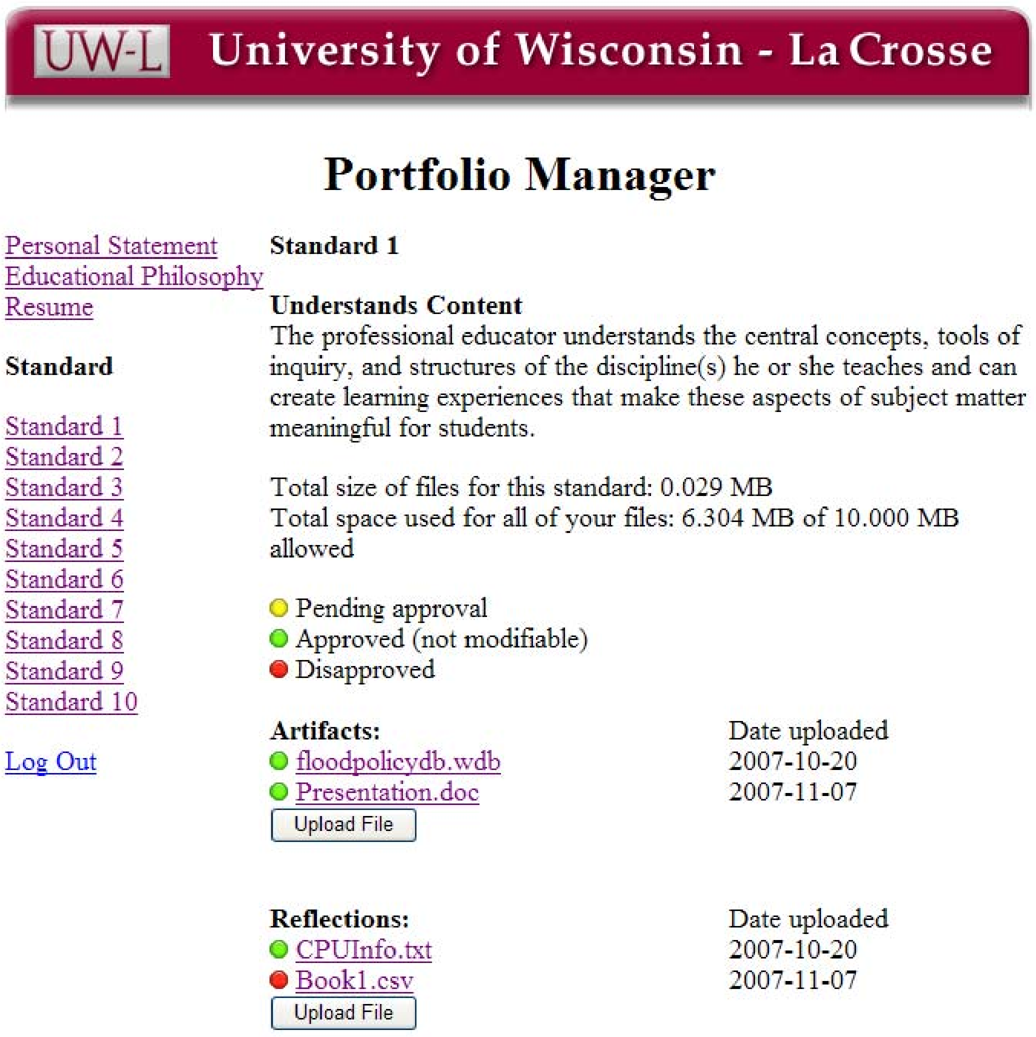
Portfolio Manager

Welcome to the portfolio manager!

Total space used for all of your files: 6.304 MB of 10.000 MB allowed

On the left, you can select a standard and view files you have uploaded for that standard.

Figure 7. Main student



Portfolio Manager

Personal Statement
Educational Philosophy
Resume

Standard

Standard 1
Standard 2
Standard 3
Standard 4
Standard 5
Standard 6
Standard 7
Standard 8
Standard 9
Standard 10

Log Out

Standard 1

Understands Content
The professional educator understands the central concepts, tools of inquiry, and structures of the discipline(s) he or she teaches and can create learning experiences that make these aspects of subject matter meaningful for students.

Total size of files for this standard: 0.029 MB
Total space used for all of your files: 6.304 MB of 10.000 MB allowed

● Pending approval
● Approved (not modifiable)
● Disapproved

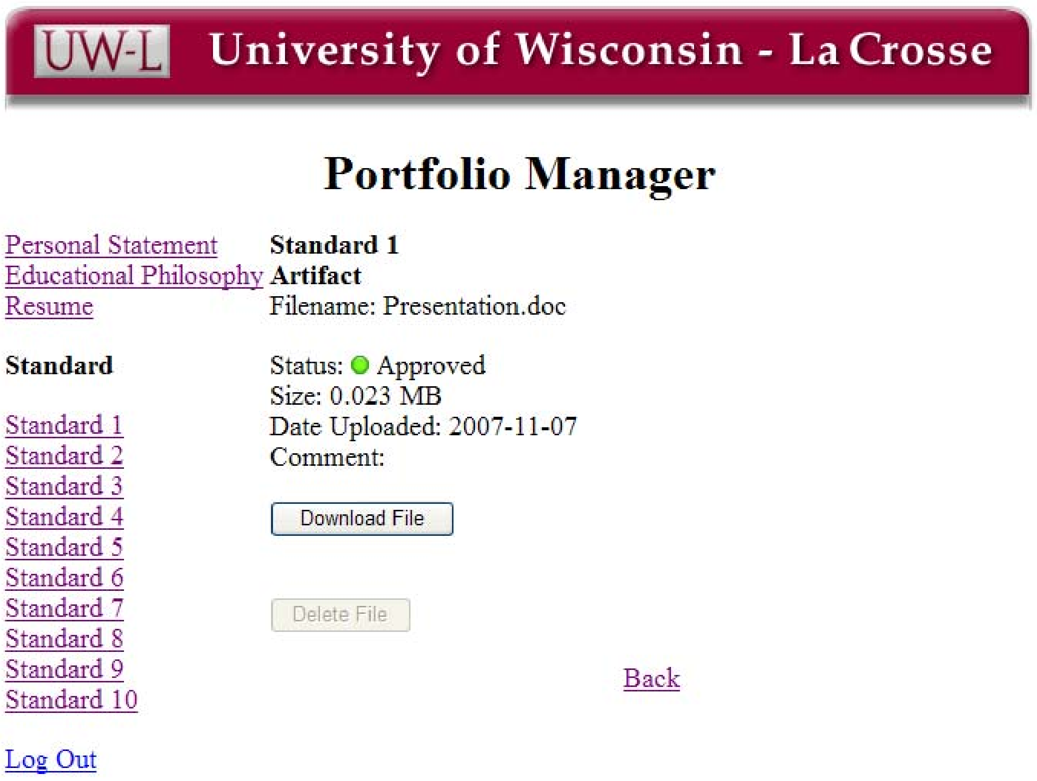
Artifacts:	Date uploaded
● floodpolicydb.wdb	2007-10-20
● Presentation.doc	2007-11-07

Upload File

Reflections:	Date uploaded
● CPUInfo.txt	2007-10-20
● Book1.csv	2007-11-07

Upload File

Figure 8. Student view files for standard

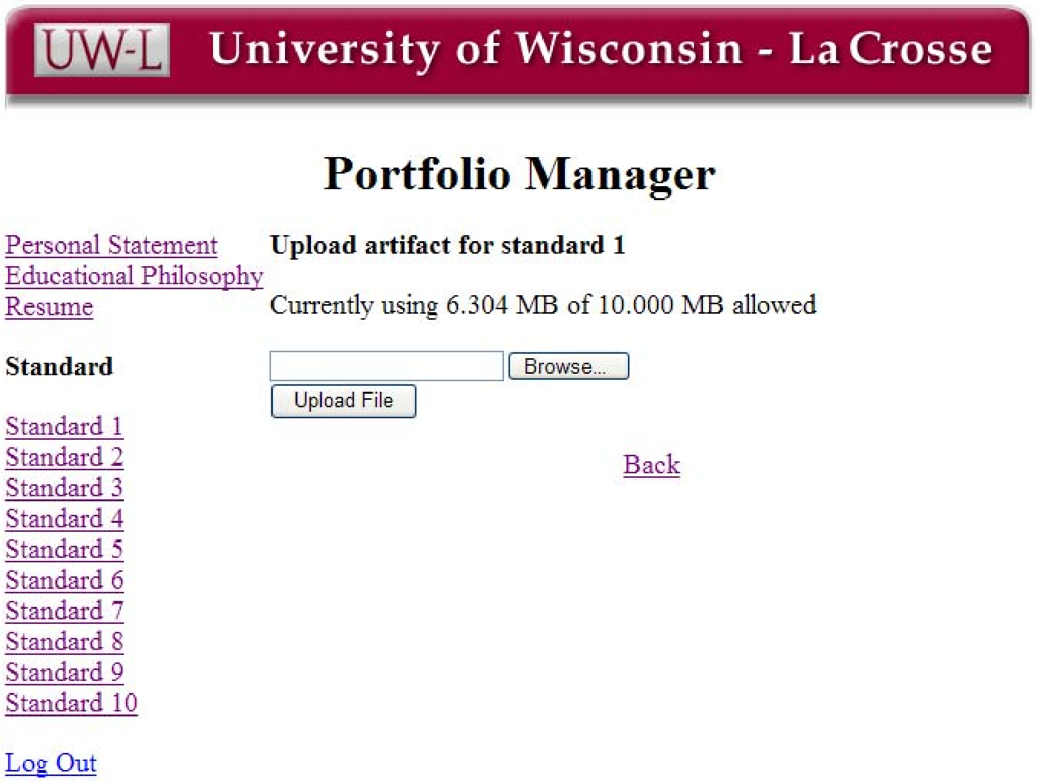


University of Wisconsin - La Crosse

Portfolio Manager

Personal Statement	**Standard 1**
Educational Philosophy	**Artifact**
Resume	Filename: Presentation.doc
Standard	Status: ● Approved
	Size: 0.023 MB
Standard 1	Date Uploaded: 2007-11-07
Standard 2	Comment:
Standard 3	
Standard 4	Download File
Standard 5	
Standard 6	
Standard 7	Delete File
Standard 8	
Standard 9	Back
Standard 10	
Log Out	

Figure 9. Student view artifact information


UW-L University of Wisconsin - La Crosse

Portfolio Manager

Personal Statement
Educational Philosophy
Resume

Standard

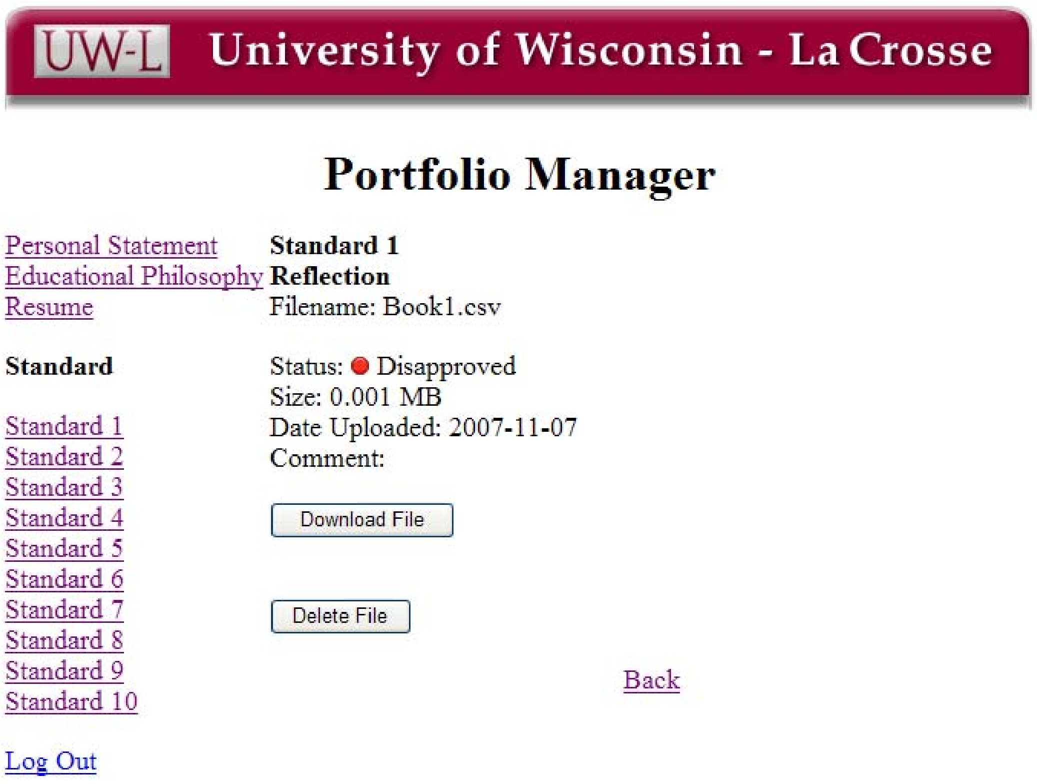
Standard 1
Standard 2
Standard 3
Standard 4
Standard 5
Standard 6
Standard 7
Standard 8
Standard 9
Standard 10

Log Out

Upload artifact for standard 1

Currently using 6.304 MB of 10.000 MB allowed

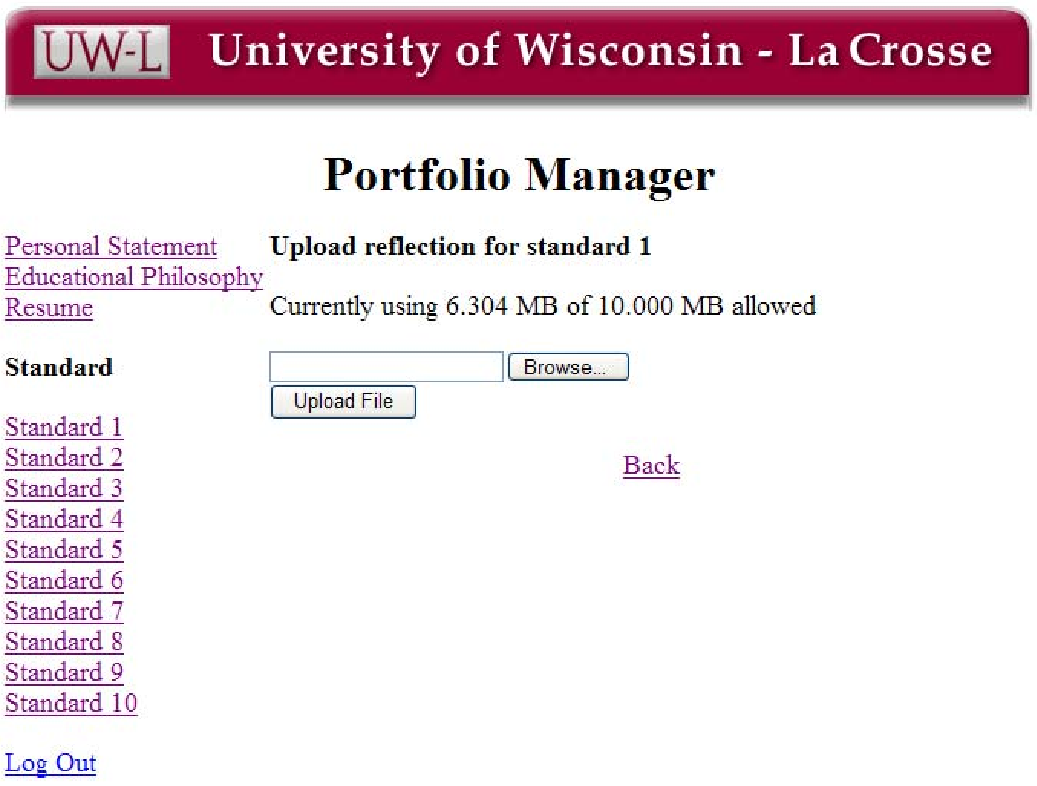
Figure 10. Upload artifact


University of Wisconsin - La Crosse

Portfolio Manager

Personal Statement	**Standard 1**
Educational Philosophy	**Reflection**
Resume	Filename: Book1.csv
Standard	Status: ● Disapproved
	Size: 0.001 MB
Standard 1	Date Uploaded: 2007-11-07
Standard 2	Comment:
Standard 3	
Standard 4	Download File
Standard 5	
Standard 6	
Standard 7	Delete File
Standard 8	
Standard 9	Back
Standard 10	
Log Out	

Figure 11. Student view reflection information


University of Wisconsin - La Crosse

Portfolio Manager

Personal Statement
Educational Philosophy
Resume

Standard

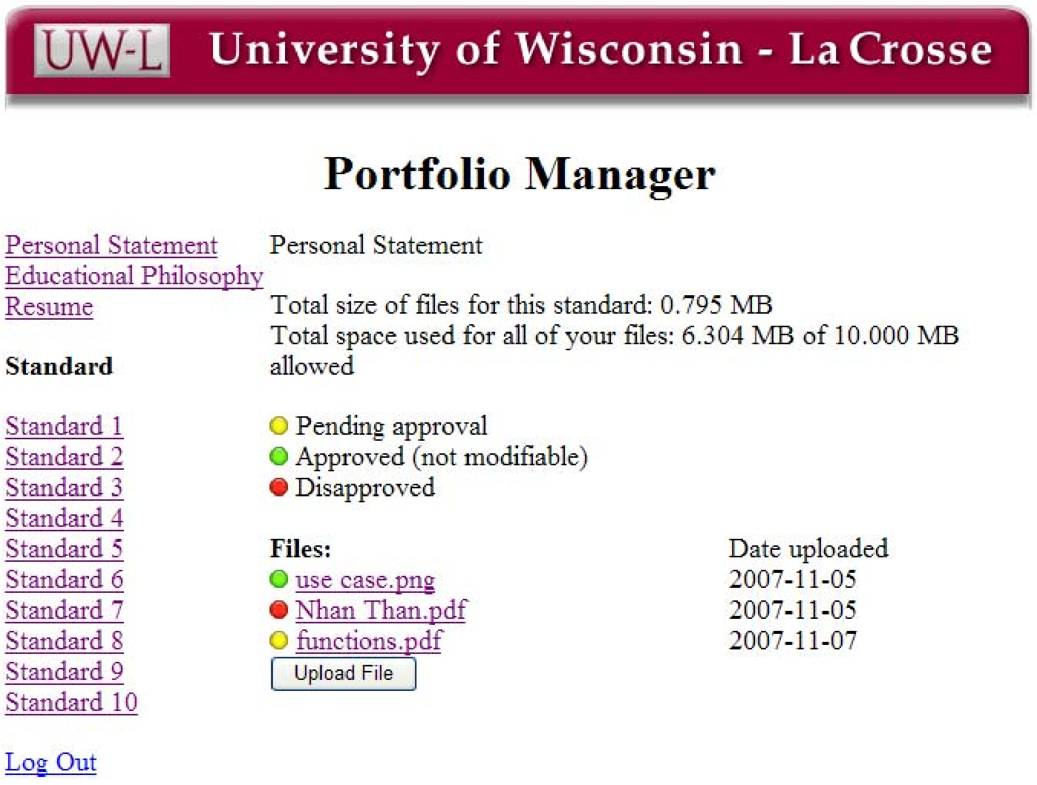
Standard 1
Standard 2
Standard 3
Standard 4
Standard 5
Standard 6
Standard 7
Standard 8
Standard 9
Standard 10

Log Out

Upload reflection for standard 1

Currently using 6.304 MB of 10.000 MB allowed

Figure 12. Upload reflection



UW-L University of Wisconsin - La Crosse

Portfolio Manager

Personal Statement
Educational Philosophy
Resume

Standard

Standard 1
Standard 2
Standard 3
Standard 4
Standard 5
Standard 6
Standard 7
Standard 8
Standard 9
Standard 10

Log Out

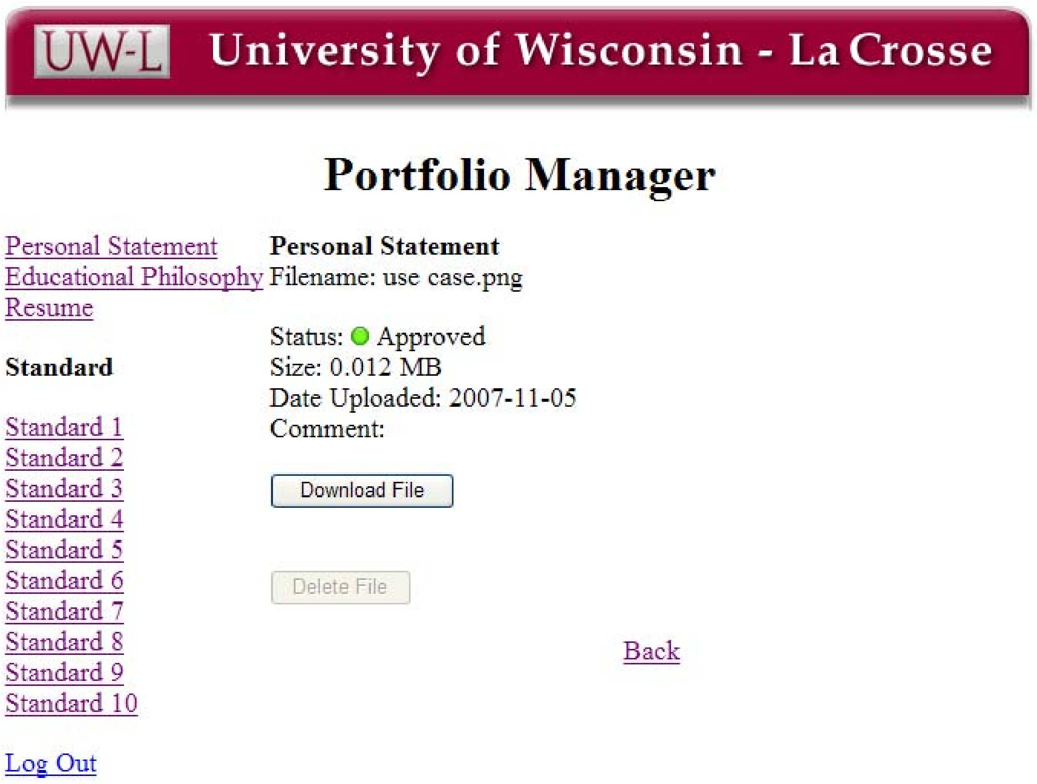
Personal Statement

Total size of files for this standard: 0.795 MB
Total space used for all of your files: 6.304 MB of 10.000 MB allowed

🟡 Pending approval
🟢 Approved (not modifiable)
🔴 Disapproved

Files:	Date uploaded
🟢 use case.png	2007-11-05
🔴 Nhan Than.pdf	2007-11-05
🟡 functions.pdf	2007-11-07

Figure 2. Student view special files


University of Wisconsin - La Crosse

Portfolio Manager

Personal Statement
Educational Philosophy
Resume

Standard

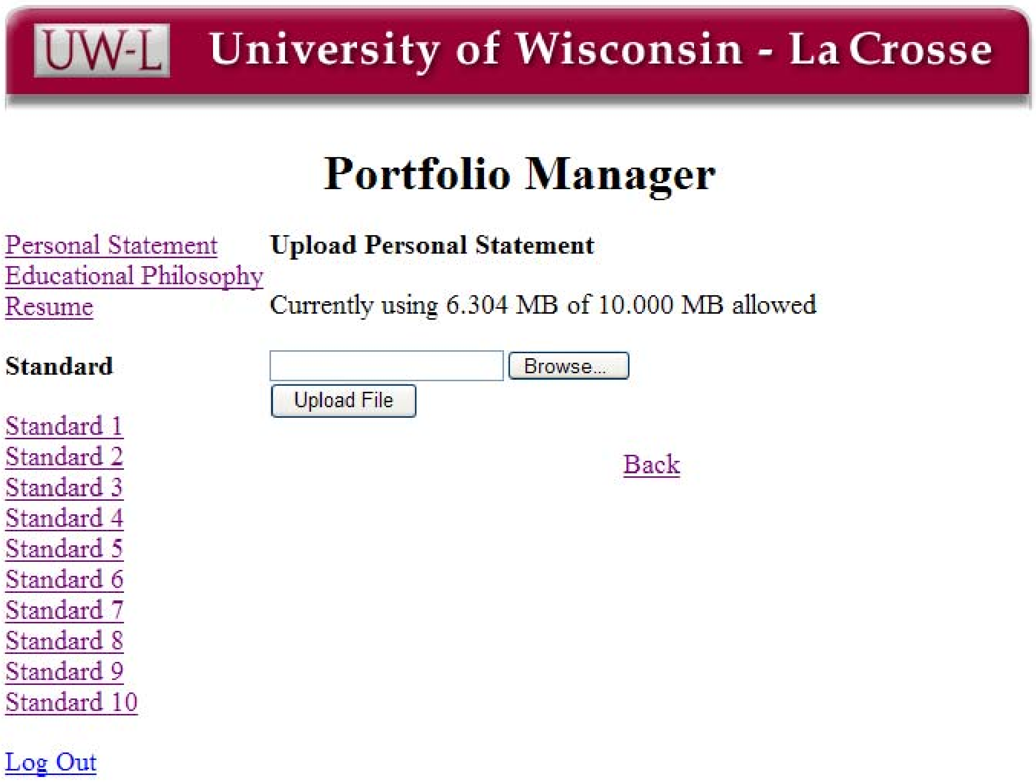
Standard 1
Standard 2
Standard 3
Standard 4
Standard 5
Standard 6
Standard 7
Standard 8
Standard 9
Standard 10

Log Out

Personal Statement
Filename: use case.png

Status: ● Approved
Size: 0.012 MB
Date Uploaded: 2007-11-05
Comment:

Figure 14. Student view special file information


UW-L University of Wisconsin - La Crosse

Portfolio Manager

Personal Statement
Educational Philosophy
Resume

Standard

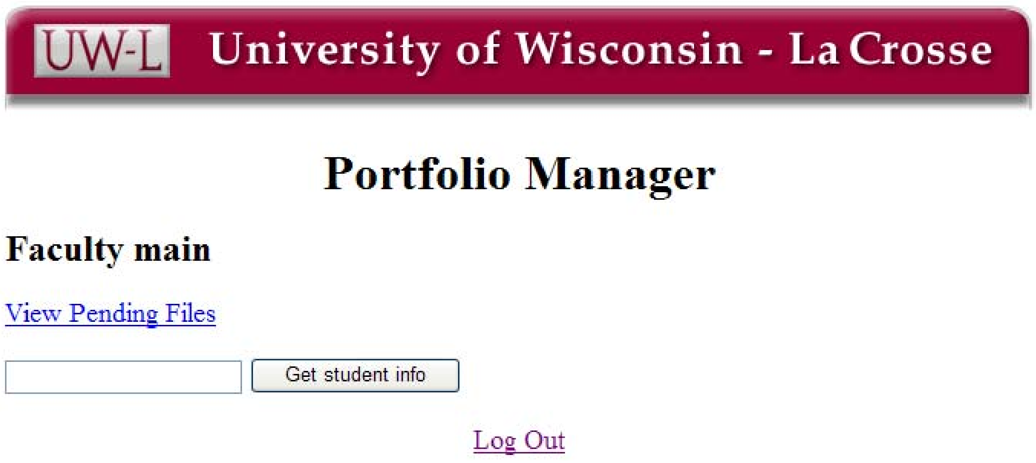
Standard 1
Standard 2
Standard 3
Standard 4
Standard 5
Standard 6
Standard 7
Standard 8
Standard 9
Standard 10

Log Out

Upload Personal Statement

Currently using 6.304 MB of 10.000 MB allowed

Figure 15. Upload special file

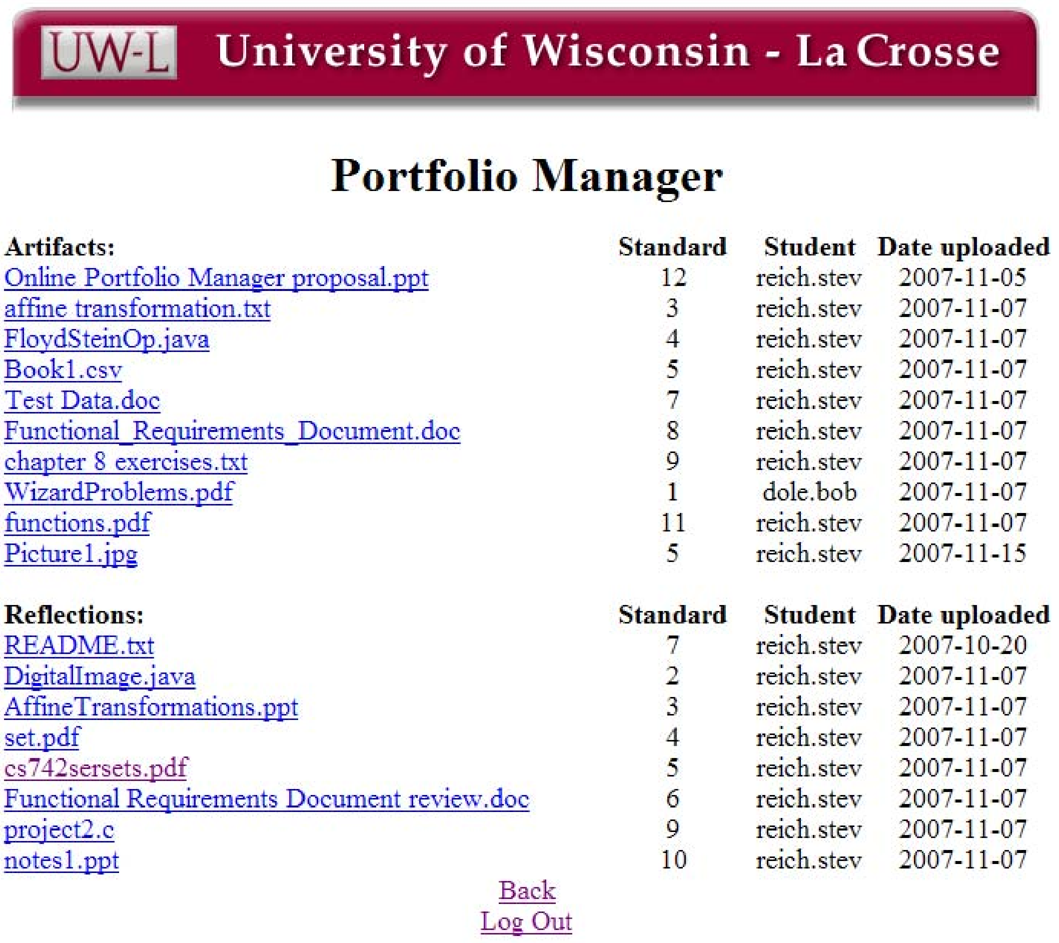

UW-L University of Wisconsin - La Crosse

Portfolio Manager

Faculty main

View Pending Files

Figure 16. Faculty main

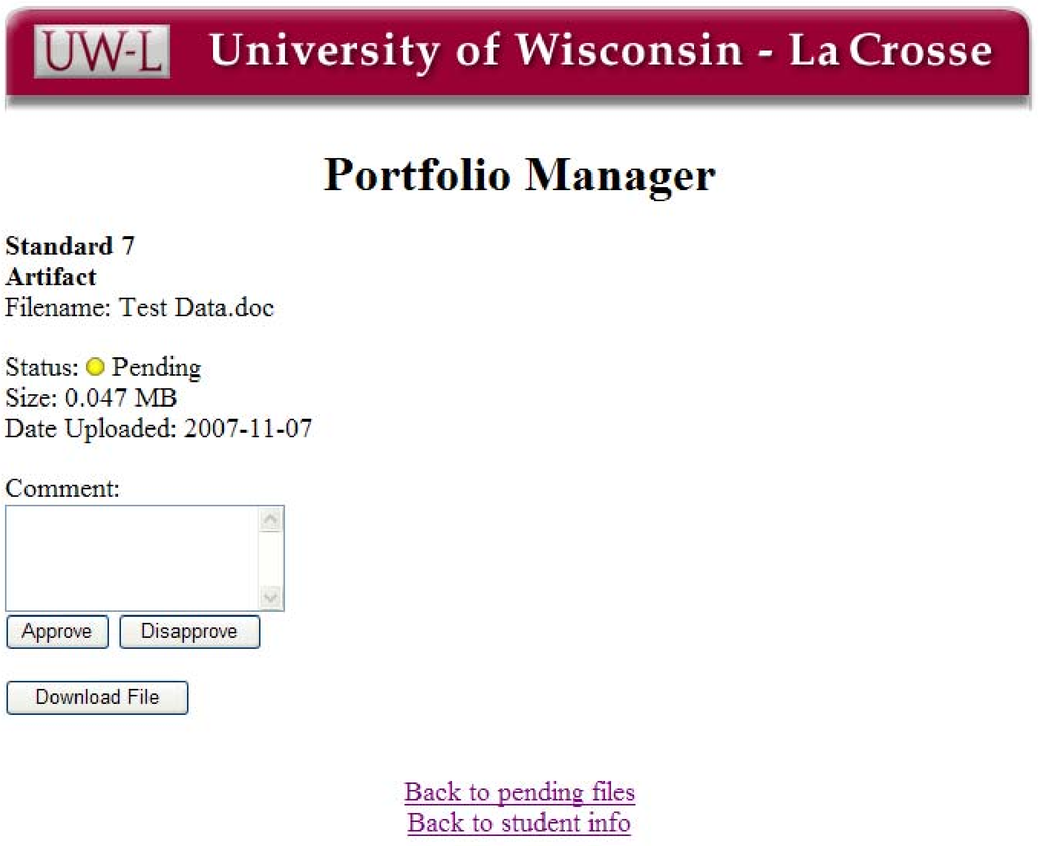


Portfolio Manager

Artifacts:	Standard	Student	Date uploaded
Online Portfolio Manager proposal.ppt	12	reich.stev	2007-11-05
affine transformation.txt	3	reich.stev	2007-11-07
FloydSteinOp.java	4	reich.stev	2007-11-07
Book1.csv	5	reich.stev	2007-11-07
Test Data.doc	7	reich.stev	2007-11-07
Functional_Requirements_Document.doc	8	reich.stev	2007-11-07
chapter 8 exercises.txt	9	reich.stev	2007-11-07
WizardProblems.pdf	1	dole.bob	2007-11-07
functions.pdf	11	reich.stev	2007-11-07
Picture1.jpg	5	reich.stev	2007-11-15

Reflections:	Standard	Student	Date uploaded
README.txt	7	reich.stev	2007-10-20
DigitalImage.java	2	reich.stev	2007-11-07
AffineTransformations.ppt	3	reich.stev	2007-11-07
set.pdf	4	reich.stev	2007-11-07
cs742sersets.pdf	5	reich.stev	2007-11-07
Functional Requirements Document review.doc	6	reich.stev	2007-11-07
project2.c	9	reich.stev	2007-11-07
notes1.ppt	10	reich.stev	2007-11-07

Figure 3. View pending files


University of Wisconsin - La Crosse

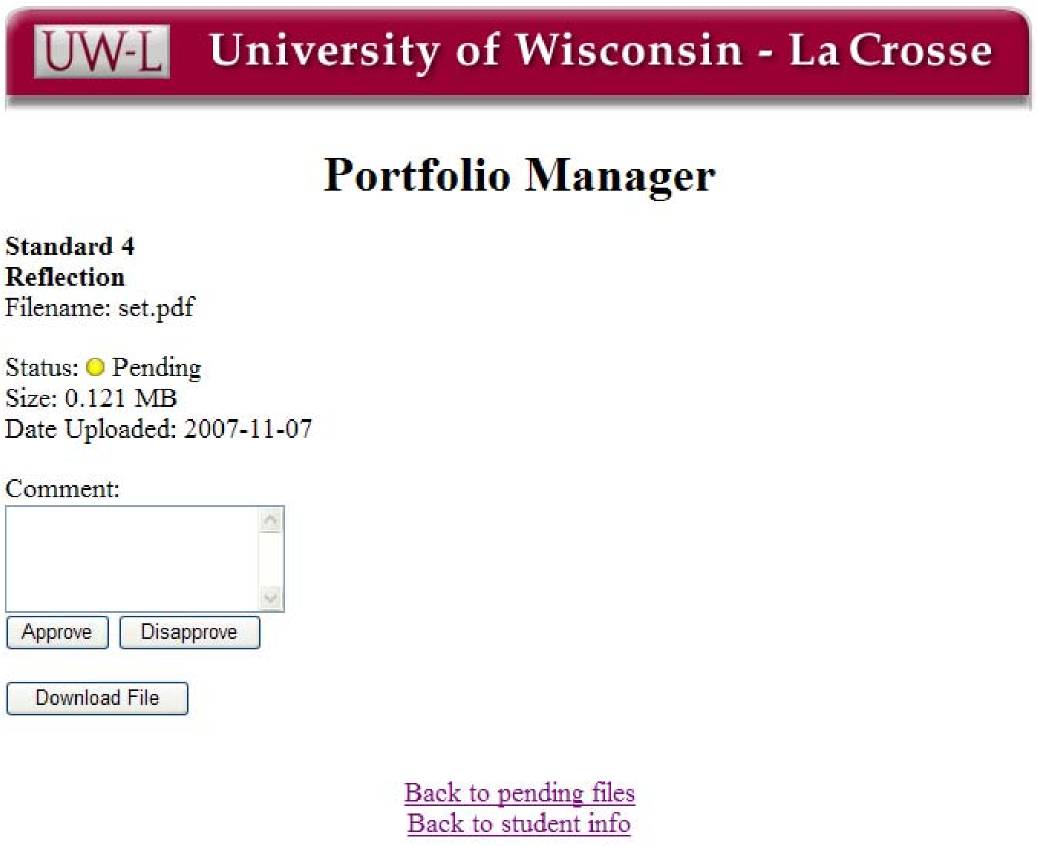
Portfolio Manager

Standard 7
Artifact
Filename: Test Data.doc

Status: ● Pending
Size: 0.047 MB
Date Uploaded: 2007-11-07

Comment:

Figure 4. Faculty view artifact information



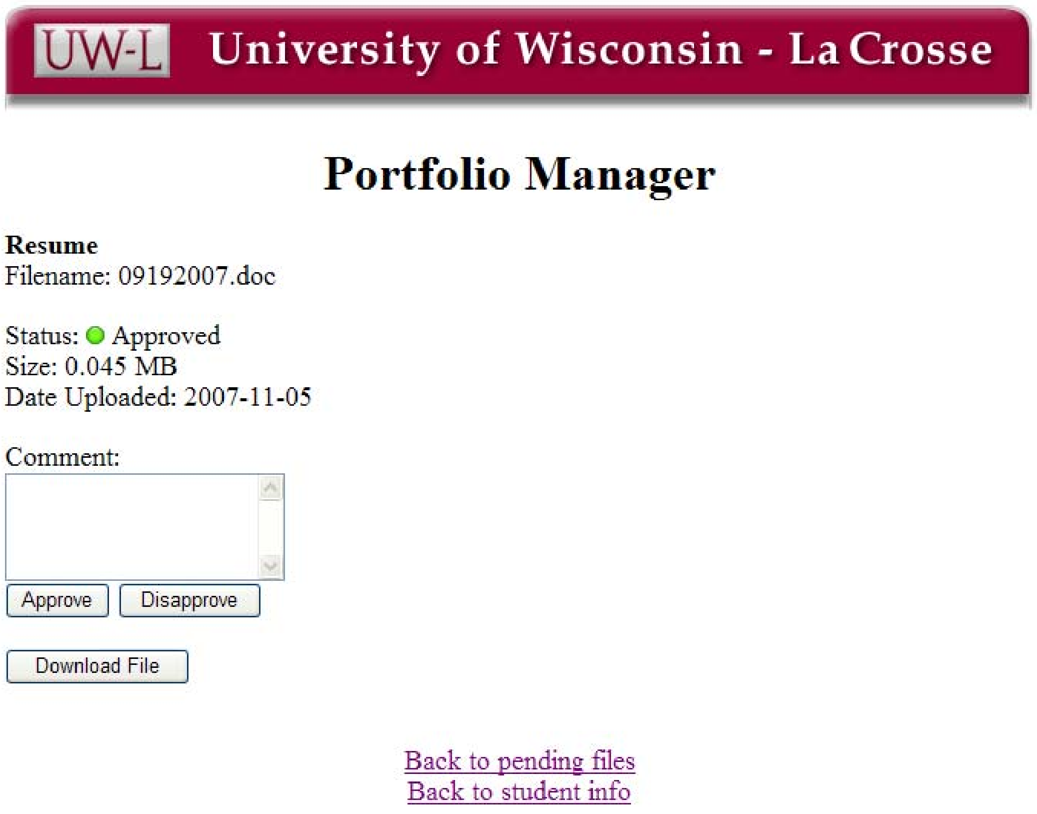
Portfolio Manager

Standard 4
Reflection
Filename: set.pdf

Status: ● Pending
Size: 0.121 MB
Date Uploaded: 2007-11-07

Comment:

Figure 5. Faculty view reflection information


University of Wisconsin - La Crosse

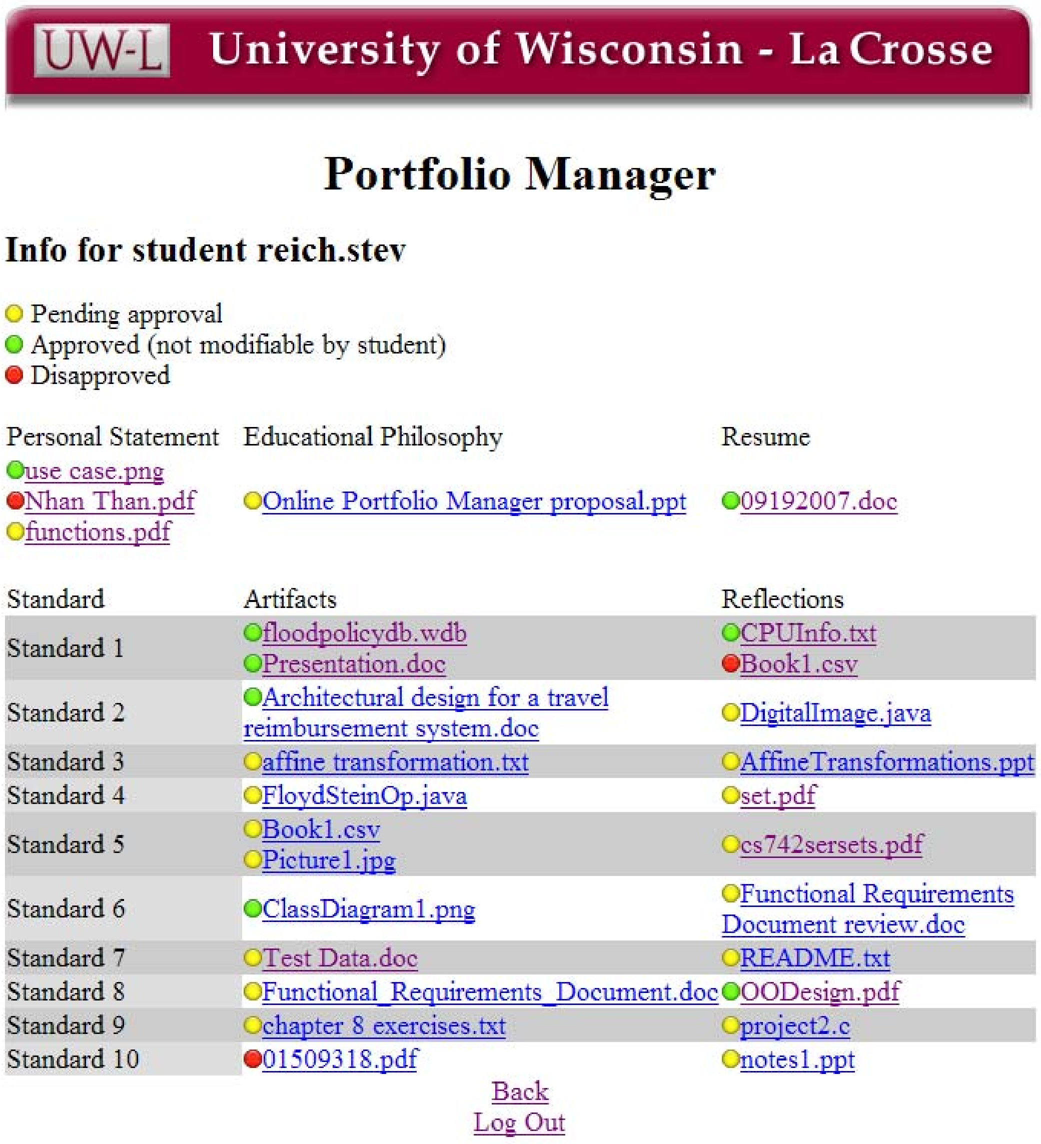
Portfolio Manager

Resume
Filename: 09192007.doc

Status: ● Approved
Size: 0.045 MB
Date Uploaded: 2007-11-05

Comment:

Figure 20. Faculty view special file information



UW-L University of Wisconsin - La Crosse

Portfolio Manager

Info for student reich.stev

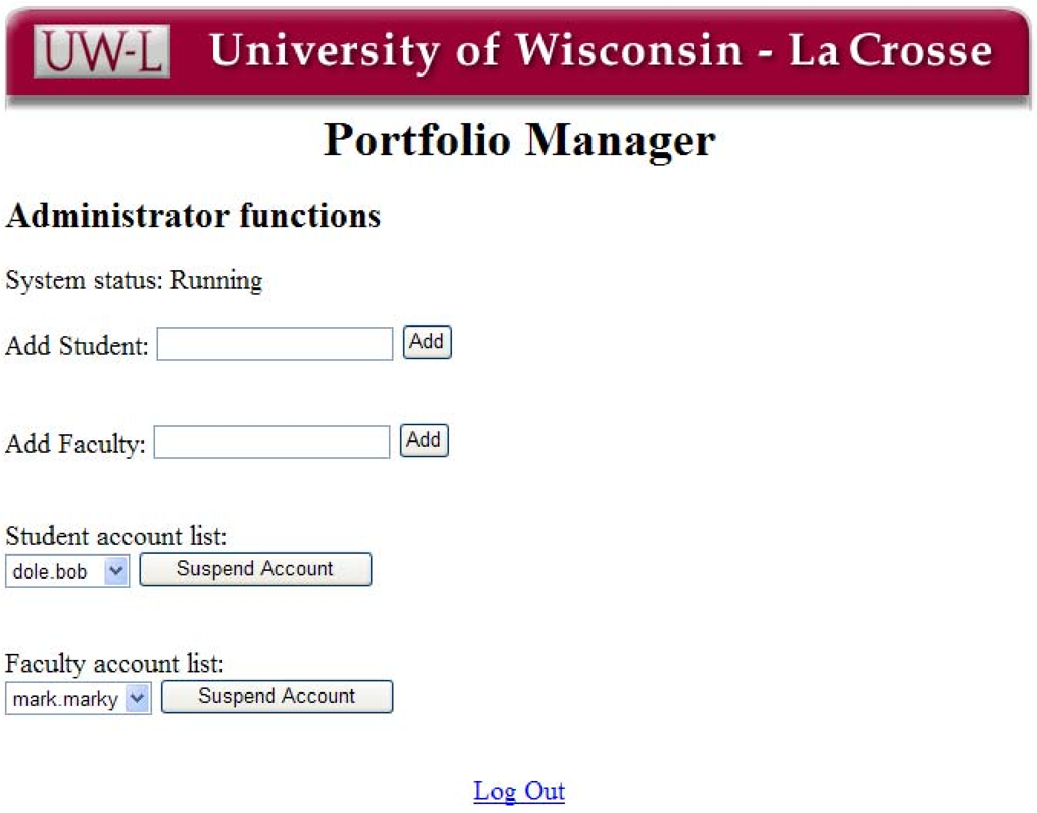
🟡 Pending approval
🟢 Approved (not modifiable by student)
🔴 Disapproved

Personal Statement	Educational Philosophy	Resume
🟢use case.png 🔴Nhan Than.pdf 🟡functions.pdf	🟡Online Portfolio Manager proposal.ppt	🟢09192007.doc

Standard	Artifacts	Reflections
Standard 1	🟢floodpolicydb.wdb 🟢Presentation.doc	🟢CPUInfo.txt 🔴Book1.csv
Standard 2	🟢Architectural design for a travel reimbursement system.doc	🟡DigitalImage.java
Standard 3	🟡affine transformation.txt	🟡AffineTransformations.ppt
Standard 4	🟡FloydSteinOp.java	🟡set.pdf
Standard 5	🟡Book1.csv 🟡Picture1.jpg	🟡cs742sersets.pdf
Standard 6	🟢ClassDiagram1.png	🟡Functional Requirements Document review.doc
Standard 7	🟡Test Data.doc	🟡README.txt
Standard 8	🟡Functional_Requirements_Document.doc	🟢OODesign.pdf
Standard 9	🟡chapter 8 exercises.txt	🟡project2.c
Standard 10	🔴01509318.pdf	🟡notes1.ppt

Back
Log Out

Figure 6. Faculty view student info


UW-L University of Wisconsin - La Crosse

Portfolio Manager

Administrator functions

System status: Running

Add Student:

Add Faculty:

Student account list:

dole.bob

Faculty account list:

mark.marky

Figure 22. Administrator main

Appendix B. Sample screen shots from the requirements document

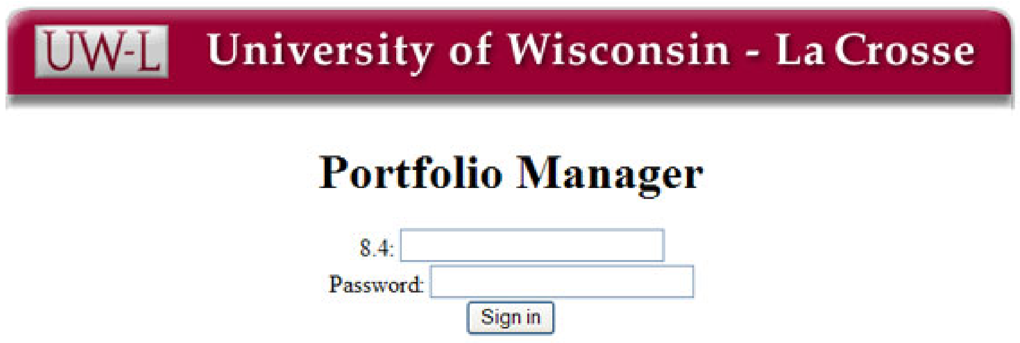
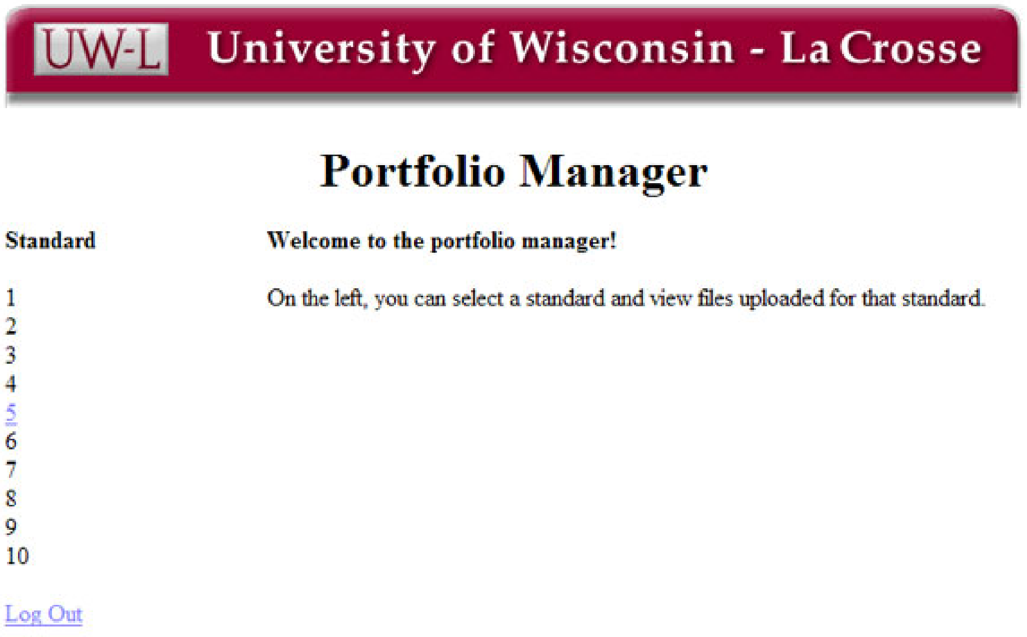


Figure 23. Student login screen


UW-L University of Wisconsin - La Crosse

Portfolio Manager

Standard

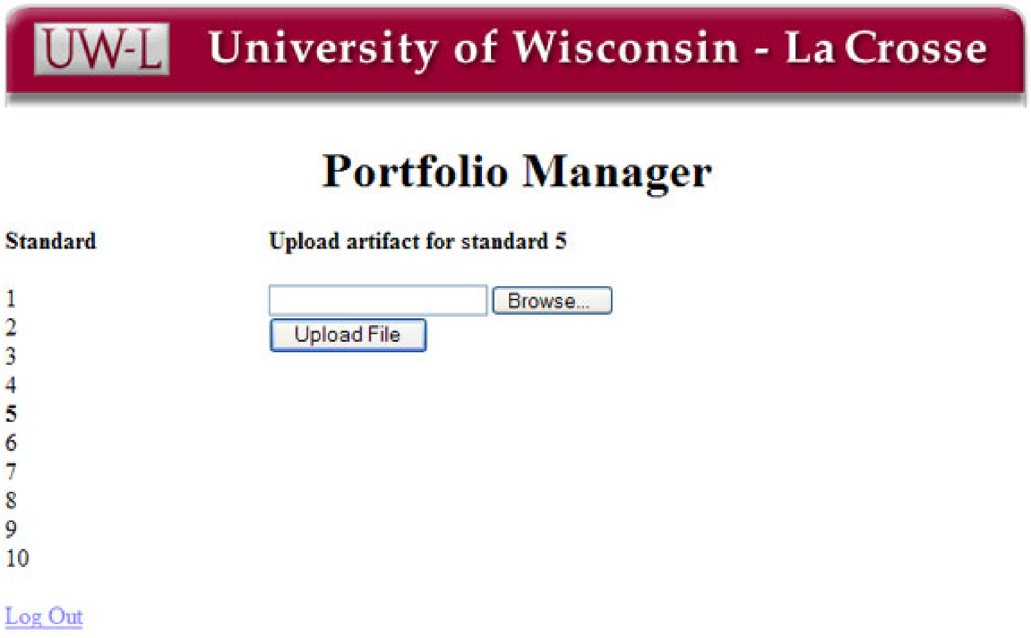
1
2
3
4
5
6
7
8
9
10

Log Out

Welcome to the portfolio manager!

On the left, you can select a standard and view files uploaded for that standard.

Figure 24. Main student page


UW-L University of Wisconsin - La Crosse

Portfolio Manager

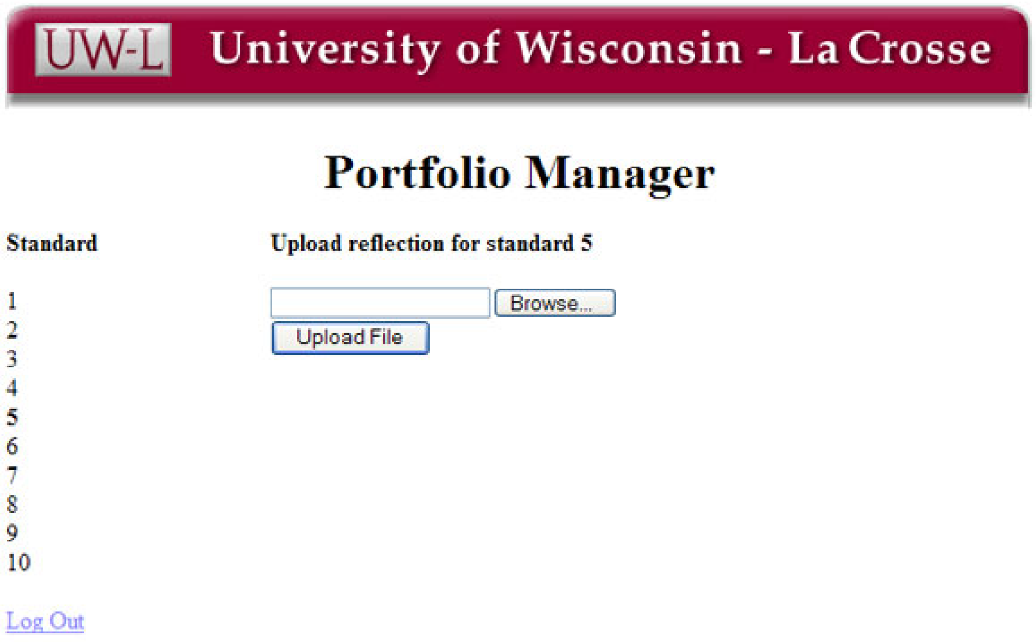
Standard

1
2
3
4
5
6
7
8
9
10

Log Out

Upload artifact for standard 5

Figure 7. Upload artifact


University of Wisconsin - La Crosse

Portfolio Manager

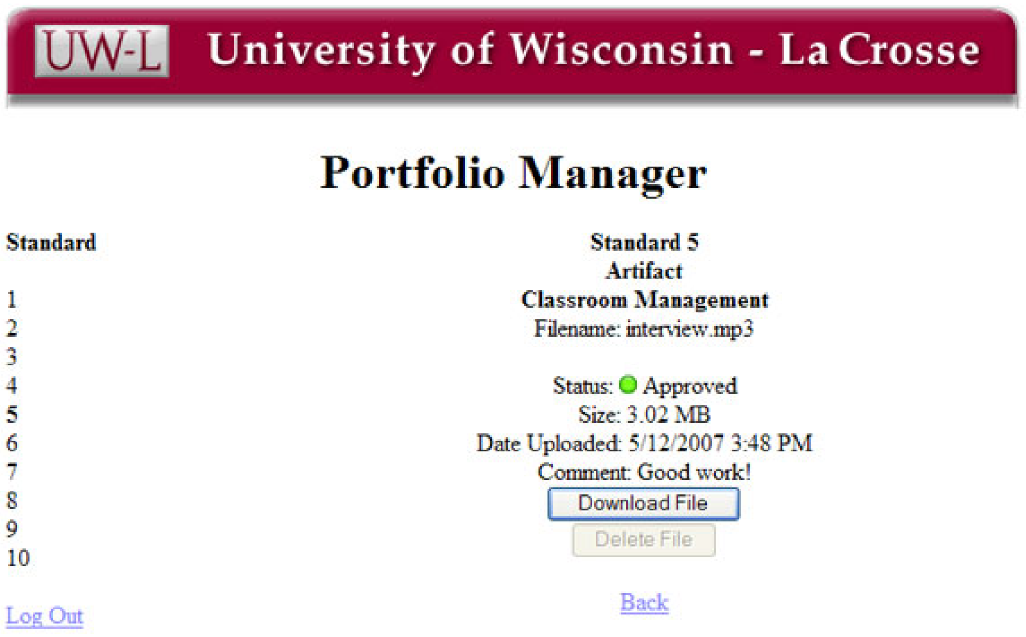
Standard

1
2
3
4
5
6
7
8
9
10

Log Out

Upload reflection for standard 5

Figure 8. Upload reflection


University of Wisconsin - La Crosse

Portfolio Manager

Standard

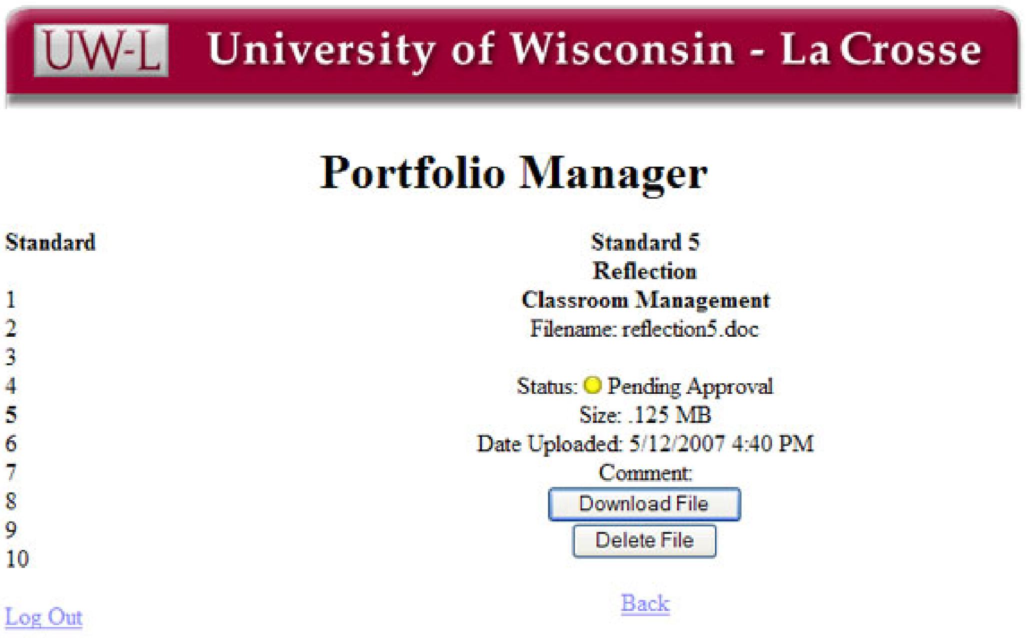
1
2
3
4
5
6
7
8
9
10

Log Out

Standard 5
Artifact
Classroom Management
Filename: interview.mp3

Status: ● Approved
Size: 3.02 MB
Date Uploaded: 5/12/2007 3:48 PM
Comment: Good work!

Figure 9. Student display artifact information


University of Wisconsin - La Crosse

Portfolio Manager

Standard

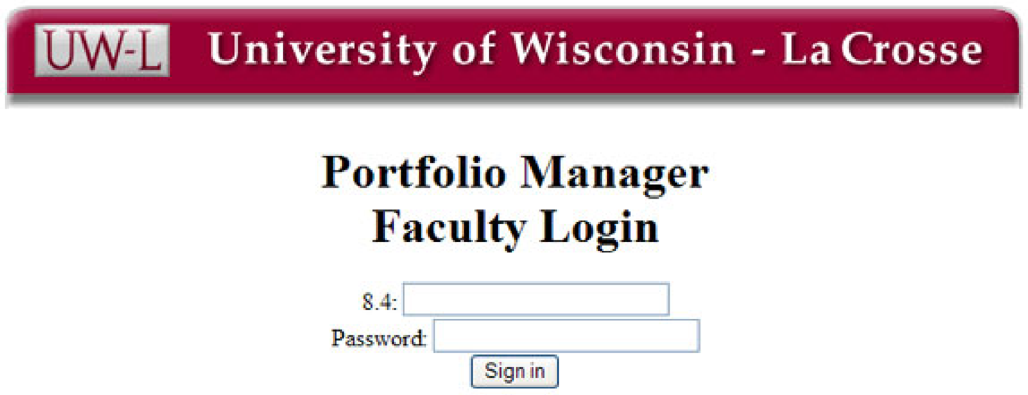
1
2
3
4
5
6
7
8
9
10

Log Out

Standard 5
Reflection
Classroom Management
Filename: reflection5.doc

Status: ● Pending Approval
Size: .125 MB
Date Uploaded: 5/12/2007 4:40 PM
Comment:

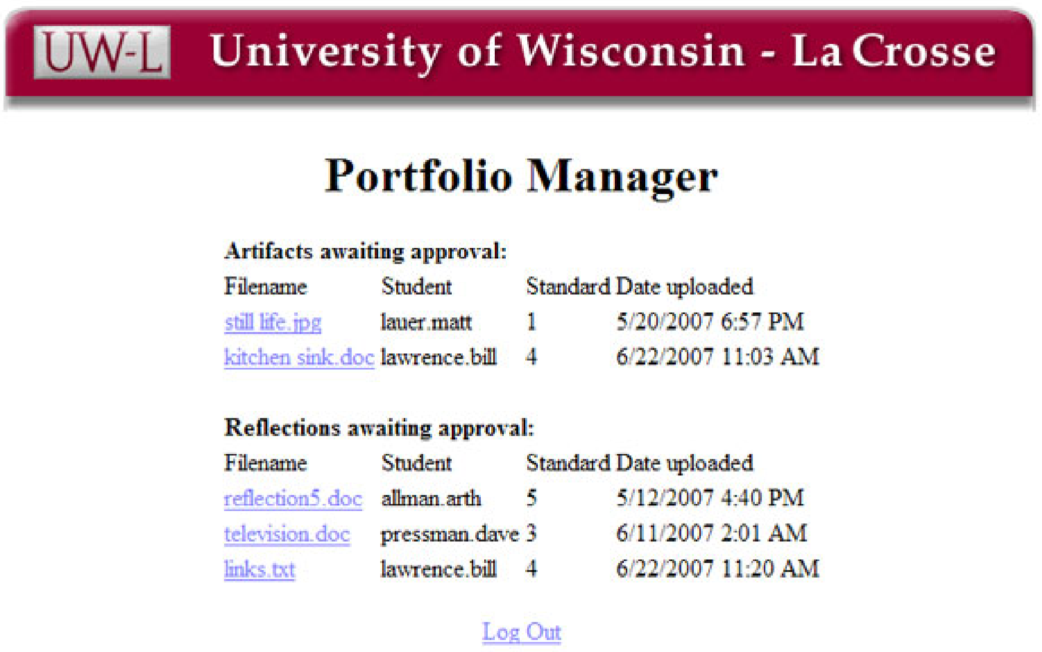
Figure 10. Student display reflection info


University of Wisconsin - La Crosse

Portfolio Manager
Faculty Login

8.4:
Password:

Figure 11. Faculty login screen


University of Wisconsin - La Crosse

Portfolio Manager

Artifacts awaiting approval:

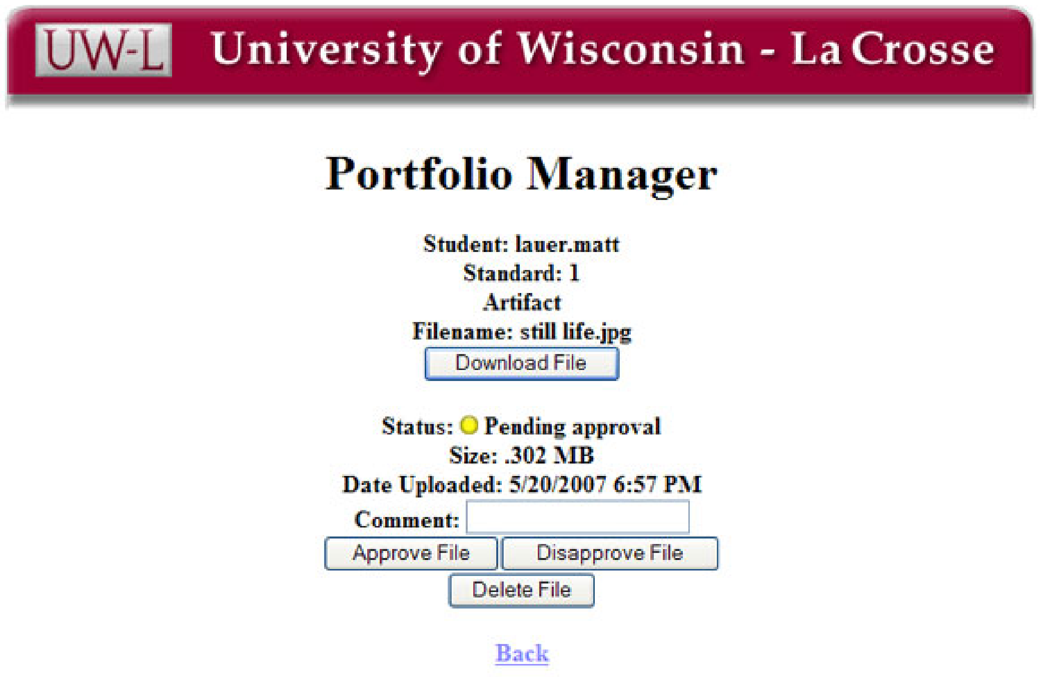
Filename	Student	Standard	Date uploaded
still life.jpg	lauer.matt	1	5/20/2007 6:57 PM
kitchen sink.doc	lawrence.bill	4	6/22/2007 11:03 AM

Reflections awaiting approval:

Filename	Student	Standard	Date uploaded
reflection5.doc	allman.arth	5	5/12/2007 4:40 PM
television.doc	pressman.dave	3	6/11/2007 2:01 AM
links.txt	lawrence.bill	4	6/22/2007 11:20 AM

Log Out

Figure 12. Faculty view pending files

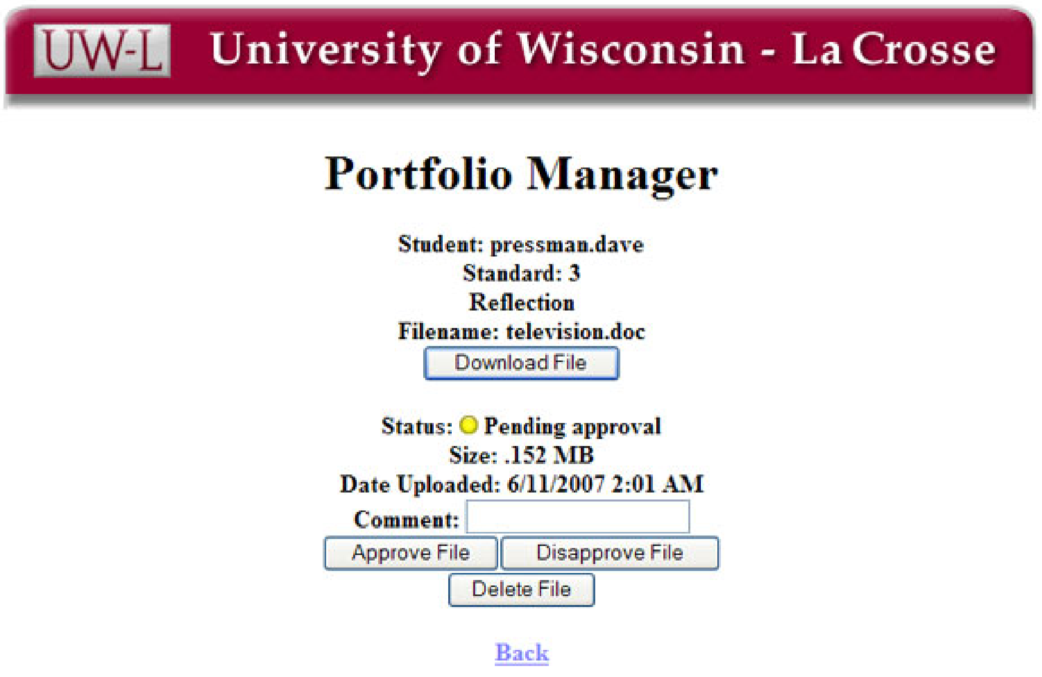

University of Wisconsin - La Crosse

Portfolio Manager

Student: lauer.matt
Standard: 1
Artifact
Filename: still life.jpg

Status: ● Pending approval
Size: .302 MB
Date Uploaded: 5/20/2007 6:57 PM
Comment:

Figure 13. Faculty view artifact information



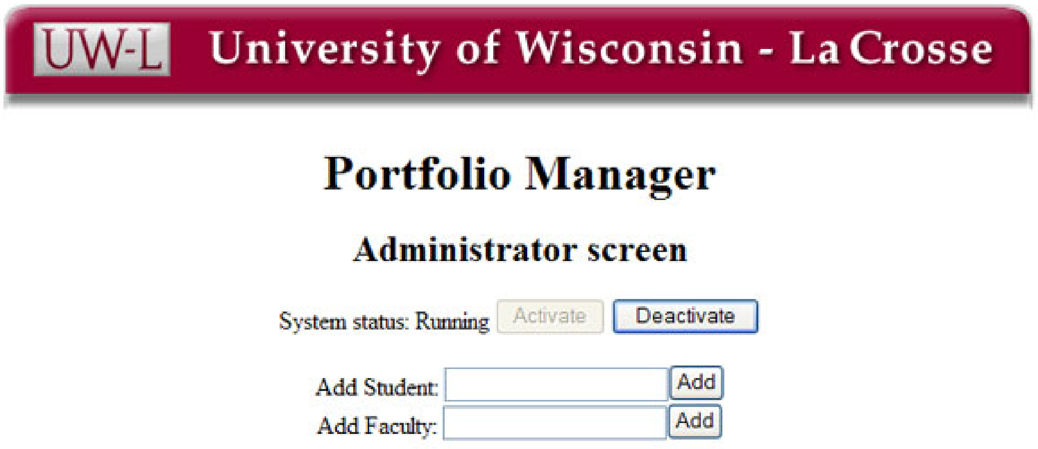
University of Wisconsin - La Crosse

Portfolio Manager

Student: pressman.dave
Standard: 3
Reflection
Filename: television.doc

Status: ● Pending approval
Size: .152 MB
Date Uploaded: 6/11/2007 2:01 AM
Comment:

Figure 14. Faculty view reflection information



University of Wisconsin - La Crosse

Portfolio Manager

Administrator screen

System status: Running

Add Student:
Add Faculty:

Figure 15 - Administrator controls