HETAC Staff Development and Training Program

NetClasses: A lecturer's web site manager.

Presented for the MSc Degree in Computing

by

Jim Doyle

Department of Computing,

School of Computing and Mathematics,

Cork Institute of Technology

December 2004

Abstract

There are a number of Learning Content Management Systems on the market. All commercial ones are expensive and all require considerable training to use. A simple lecturer's web site management tool, **NetClasses,** which is both inexpensive and easy to use, is presented in this thesis.

NetClasses is cross platform. It is largely written from scratch and has a simple menu driven interface. Free open access and open source middleware is used to deploy the web pages.

This thesis describes how NetClasses is produced. The ease of use comes from the menu interface and the underlying tight control of progression from one web page to the next.

Acknowledgements

My greatest thanks goes to Derek O Reilly who had a number of parts to play in this project. Firstly he lectured one of the taught modules, Multimedia Programming, that I took in preparation for this undertaking. Secondly he agreed to be the project supervisor. Thirdly he came up with the idea for the project and lastly he steered me back on course whenever things started to go in the wrong direction.

I would also like to thank my wife, Marian, who had to put up with me being absorbed in the project, sometimes to the detriment of my role as husband.

Thanks also to Morris Rimbi who proof read the thesis.

Finally, thanks to my colleagues in Cork Institute of Technology who contributed their opinions on requirements and on partly completed modules.

Table of Contents

Chapter 1

Introduction And Overview

1.1 Introduction

1.1.1 Goals

The thesis describes the creation of a web site management tool called NetClasses. NetClasses can be used by both lecturers and students.

Lecturers can use Netclasses to:

- Post notes.
- Assign passwords.
- Post assignment instructions.
- Collect assignments.
- Present multiple choice quizzes.
- Operate a notice board.

Students can use NetClasses to:

- Read notes.
- Read assignment instructions.
- Submit assignments.
- Take multiple choice quizzes.
- Read notices on the notice board.

Both lecturers and students can:

- Log on using a password.
- Change a password.
- Log off.

NetClasses is a tool for managing lecturer's notes, student assignments and quizzes, and for broadcasting text messages to students. The intended users of NetClasses are students and lecturers of a course, therefore, access to the system is limited through passwords. The lecturer and students could be attached to any kind of educational institution. However, this thesis concentrates on a third level course where the facilities mentioned above are used routinely to exchange work between students and lecturers.

The system is multi-user in that there can be many lecturers with their own classes of students using the same web address to access material relevant to the course they are teaching or studying.

NetClasses is scalable. This means that the number of lecturers and students can increase without affecting the reliability of the system.

1.1.2 What The Project Entails

It is clear that this project requires web-based database access. This thesis describes research into technologies and environments that can be used to provide the service. The chosen technology (Java Server Pages) is examined and decisions are made as to how the application is deployed.

The project involves research into the area of Web Assisted Learning to get information on what is already available. This information is used in conjunction with the requirements for the proposed application.

There should not be too many mouse clicks required when performing any NetClasses task [GALI02]. The page design should be intuitive and, more importantly, the navigation should be learnable [HOLM02]. There are a number of different tasks to be performed by users of NetClasses. The flow control issues that arise when navigating through a web site that has many different pages are considered.

Individual components are researched in more detail. All components except the login are written from scratch. These programs are developed from specifications rather than by modifying existing code because nothing that meets all the requirements is available free of charge. This process helps in gaining an understanding of how to develop such programs and makes it easier to get NetClasses to meet the exact requirements.

1.2 Chapter By Chapter

1.2.1 Chapter 2

Sections 2.1.1 and 2.1.2 discuss how the user requirements are arrived at. The intended audience is identified. Component requirements are discussed in *Section 2.2.*

1.2.2 Chapter 3

In *Section 3.1*, Existing Technologies, is the research into various technologies used to create dynamic web pages and the reasons why Java Server Pages is chosen as the technology for NetClasses. *Section 3.3* discusses the choice of Database application. Some features from different applications that offer similar services to NetClasses are compared in *Section 3.4*.

1.2.3 Chapter 4

Sections 4.2 and 4.3, relate some of the decisions taken about how the site files are stored and how the web pages look. There is also a description of how access is restricted to certain parts of the site in *Section 4.4*. The final section, *Section 4.5*, discusses the most challenging aspect of NetClasses; how the progression from one web page to another is controlled within the site.

1.2.4 Chapter 5

The building of the major parts of the NetClasses application is reviewed in this chapter. *Section 5.1* discusses the login component which is the only one not written from scratch. The login is a standard login program largely taken from [FIEL02] although it is modified to take account of the restrictions on site access. The parts of NetClasses used by the lecturers are discussed in *Section 5.2*. Those used by students are in *Section 5.3*. Finally *Section 5.4* presents the administrator's components.

1.2.5 Chapter 6

Chapter 6 concludes the thesis and presents suggestions for improvements and further work.

Chapter 2

Analysis And Specifications

2.1 Analysis

2.1.1 General

The original broad specification for the project came from the Project Supervisor who had the idea to create the web site that would provide the facilities listed in *Section 1.1.1*. After initial consideration and discussion it was decided that NetClasses should be cross platform (usable on Windows and Unix based systems). It was also decided at this stage that the lecturers should be able to use their existing notes files, regardless of format, when using NetClasses.

For any application to provide web pages to multiple users, a web server is required. The cross platform web servers are mostly written in the Java programming language [TAYL98], therefore, NetClasses is developed and deployed using a Java web server.

Even though the application is cross-platform it was decided to write and test it on an office PC that runs Windows and then to test it in a Unix environment. If NetClasses uses Java throughout then it can work on UNIX based machines as well [COOK03].

The application is meant to be accessible from the Internet as well as the college intranet. The speed of downloading web pages is slower on a home PC than on a college PC if the home PC is connected to the internet by a standard modem. Because of this network speed consideration, the size of the web pages should be kept small so as not to have unduly long waiting time for pages to change [NIEL00].

NetClasses should be able to run on various different kinds of machine. As mentioned before, it should be able to run in both the Unix and Windows environments. NetClasses is a distributed application; some of it runs on one machine and some on another. In practice this means that one machine, the server, holds the data and forms the web pages and sends them to the user. The users are on other machines, the clients, which are used to view the pages. It is a sensible requirement that no particular software is needed on the client machines other than freely available web browsers [ORFA99]. The industry standards are NetScape and Internet Explorer for Microsoft Windows and NetScape-Mozilla for Unix. The modern versions of those three are called Document Object Model (DOM) browsers [MARI02] and it is a NetClasses requirement that the client side should run on any machine with a DOM browser.

2.1.2 User Requirements

2.1.2.1 Audiences

There are three audience types that use the application and they need to be catered for in the design. They are administrators, lecturers and students.

2.1.2.1.1 Administrators

Initially it was thought that the administrator should have complete responsibility for creating and deleting class groups of students. Because of our personal experience with class lists and waiting for realistic listings to be available, NetClasses is now a lecturer based system. Administrators do not need to get involved in this end of the work. The creation of a college wide student list is not a trivial task for an institution the size of Cork Institute of Technology (CIT) [DOHE94]. The task of creating class lists is easier if it is spread among the lecturers concerned. NetClasses allows lecturers to do that.

Administrators should have the job of adding lecturers to the system. It makes sense that administrators can also add other administrators. This approach is taken by the commercial applications reviewed in *Section 3.4* (WebCt 3.6 Reference Manual for Designers – Module 1, page 8). This latter ability needs to be controlled or there may be more administrators than the original administrator intends. To control the proliferation of administrators, NetClasses has the requirement that there is one administrator who can create lecturers and administrators. Those administrators, once created, can only create lecturers.

Administrators should be able to delete lecturers and the main administrator should also have the ability to delete other administrators from the system. Again this is the approach taken by the commercial applications reviewed in *Section 3.4*. The main administrator should be able to delete any lecturer but other administrators should only be able to delete lecturers created by them. They should not be able to delete lecturers created by a different administrator. The same applies to editing lecturer records.

Lecturers can be administrators in real life [BAUER99] so in NetClasses, administrators also have the functionality of lecturers.

2.1.2.1.2 Lecturers

NetClasses is specifically aimed at lecturers but students are also a target audience. This thesis develops a tool used by lecturers to provide services to students.

Making NetClasses lecturer centred means that lecturers can add courses to the system. A lecturer might be teaching two different subjects to a class of students or could be teaching the same subject to two different classes. It is also normal for several lecturers to be teaching the same class. The combination of lecturer, class and subject identifies a course in NetClasses.

NetClasses is designed so that each lecturer is responsible for entering the names and passwords into their own classes. The task of entering the student details by hand can be tedious for a large class. NetClasses removes the tedium by allowing student details to be pasted in from columns in a data file, e.g. a spread sheet or text file. The lecturer must also have the ability to get rid of courses from the system.

Every user should have only one username and password in the system [JOHN03]. When a lecturer is entering a student into a class, a student ID number that is unique to that student has to be entered. This is not a difficult thing to do because colleges like CIT have unique student number for every student that ever attended the college [CENT04]. If a lecturer enters a student username that is already in the database, the clash needs to be resolved automatically.

Lecturers must be able to upload files of notes. As mentioned above, lecture notes written outside the system e.g. in Microsoft Word should be usable. This means that some system of transferring the file from the lecturer's computer to the web server is needed. The same thing applies for lecturers uploading files describing student assignments, the assignment instructions.

2.1.2.1.3 Students

Students are users of NetClasses who must be able to upload files from their own computers. As readers of the course notes and assignment instructions students need a sensible interface through which to access this material.

Each student should have only one username and password to log on to NetClasses [JOHN03].

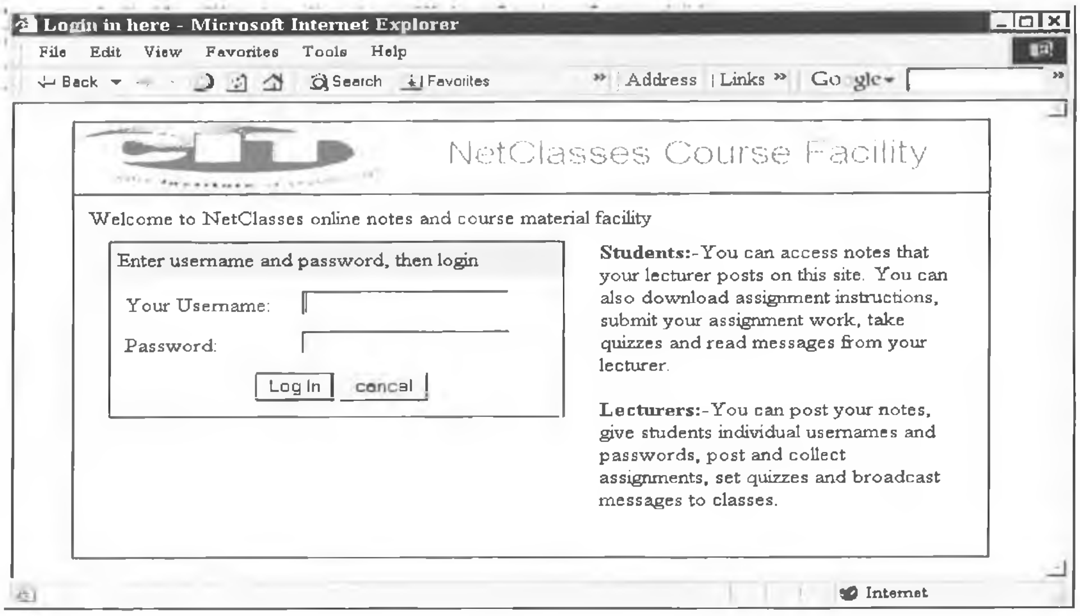
2.2 *Specifications*

The analysis of user requirements leads to the formulation of the following specifications. Following is a list is of the facilities to be supplied and some specifics of the functionality required.

2.2.1 Logging On

All users should see the same screen whether they are administrators, lecturers or students [JOHN03]. This is dictated by the fact that when a user first accesses the system, NetClasses has no way of knowing which category of user they are. As soon as the username and password are entered, NetClasses can identify his status and treat him appropriately. To facilitate this, every user should have a username, password and role.

In order to reduce the workload of the person logging on it makes sense not to ask for the user's role but to deduce it from the username and the course being logged on to. To ask for the role would be to ask for redundant information [BROW02]. A NetClasses login screen is shown in *Fig. 2.1*. The code for the login page is in the login.jsp file and is included in *Appendix 2*. All other pages are also stored as Java Server Pages (JSP) files.


Login in here - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Address Links Google
NetClasses Course Facility
Log In
cancel
Internet

Welcome to NetClasses online notes and course material facility

Enter username and password, then login

Your Username:	
Password:	

Students:-You can access notes that your lecturer posts on this site. You can also download assignment instructions, submit your assignment work, take quizzes and read messages from your lecturer.

Lecturers:-You can post your notes, give students individual usernames and passwords, post and collect assignments, set quizzes and broadcast messages to classes.

Figure 2.1 Login page.

A student would likely want to access material for several different subjects. It is not sensible to have students choose which subject they want to read notes for or submit an assignment for each time they choose a menu option [BROW02]. A more robust method is to have the student choose a specific course (lecturer, class and subject combination) at login time. Every action they take will then be directed at that section of the site. When a student or lecturer is associated with more than one course in NetClasses they are asked to choose a course to complete the login process.

When lecturers have no course (when they are first added to the system) they should still be able to log on so that they can create classes and courses.

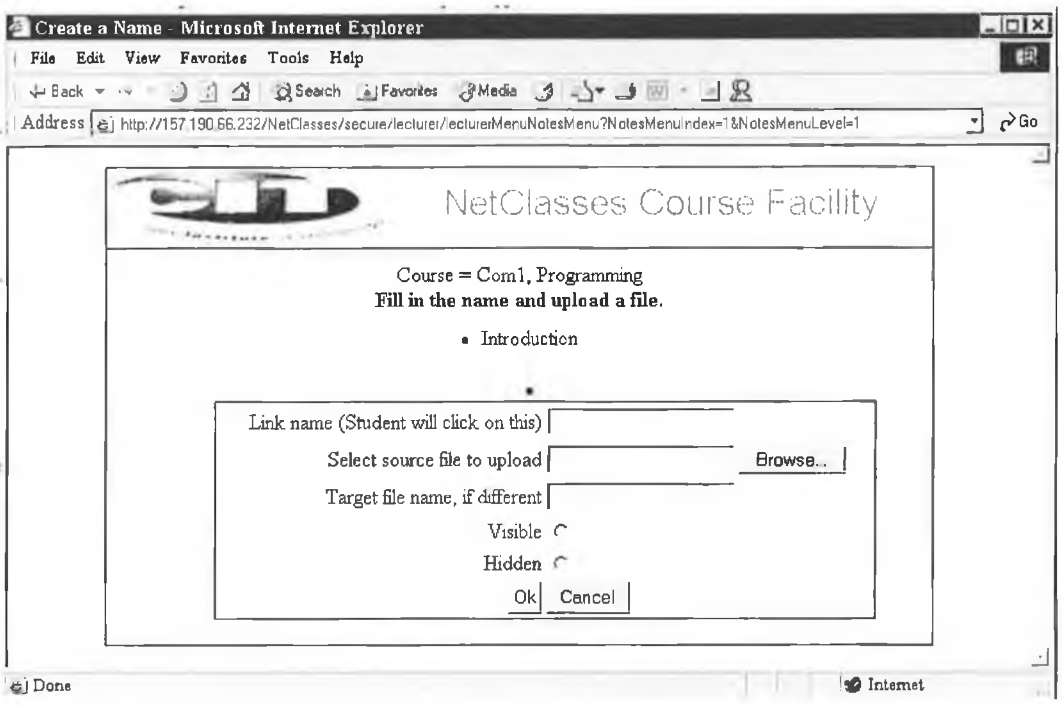
2.2.2 Posting And Accessing Notes

Both lecturers and students need to have a user-friendly method of accessing course notes. The lecturer needs to upload them and the student needs to view them. The web page that the student should see when viewing notes should be as intuitive as possible. If a person is viewing a web page to access notes then the web page should have a list of note names and each notes file should be accessible by making one selection on the page [BAXL02]. To help in selecting notes files, they should be grouped under logical headings on the page. There should be a facility for the lecturer to create headings on the notes page and to group files under these headings. It should be clear to the user of the notes web page which page-item is a heading and which is a notes link.

Students may want to read more than one file in a session. Therefore, the page where the file is opened from should be only one mouse-click away from the page the file is viewed on i.e. it should be easy to view the next file [BROW02].

When posting notes, the lecturer should be given a view of what the student will see when accessing the notes. It should be possible to submit notes of any type e.g. Word documents, text files, pictures or even computer programs that the students can download. It should also be possible to delete notes or overwrite old notes with updated versions.

In some circumstances it might be desirable to have all the notes available at the beginning of a course and allow the students to create individualised programs of study [SCHR98]. In cases where the lecturer does not want to do that, it should be possible to hide notes from students until they are relevant. The lecturer can load up all the notes at the beginning of the year and roll them out as he sees fit. *Fig. 2.2* shows the screen for uploading a file.


Create a Name - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://157.190.66.232/NetClasses/secure/lecturer/lecturerMenuNotesMenu?NotesMenuIndex=1&NotesMenuLevel=1
Go
NetClasses Course Facility

Course = Com1, Programming
Fill in the name and upload a file.

- Introduction

Link name (Student will click on this)	
Select source file to upload	Browse...
Target file name, if different	
Visible	○
Hidden	○
Ok	Cancel

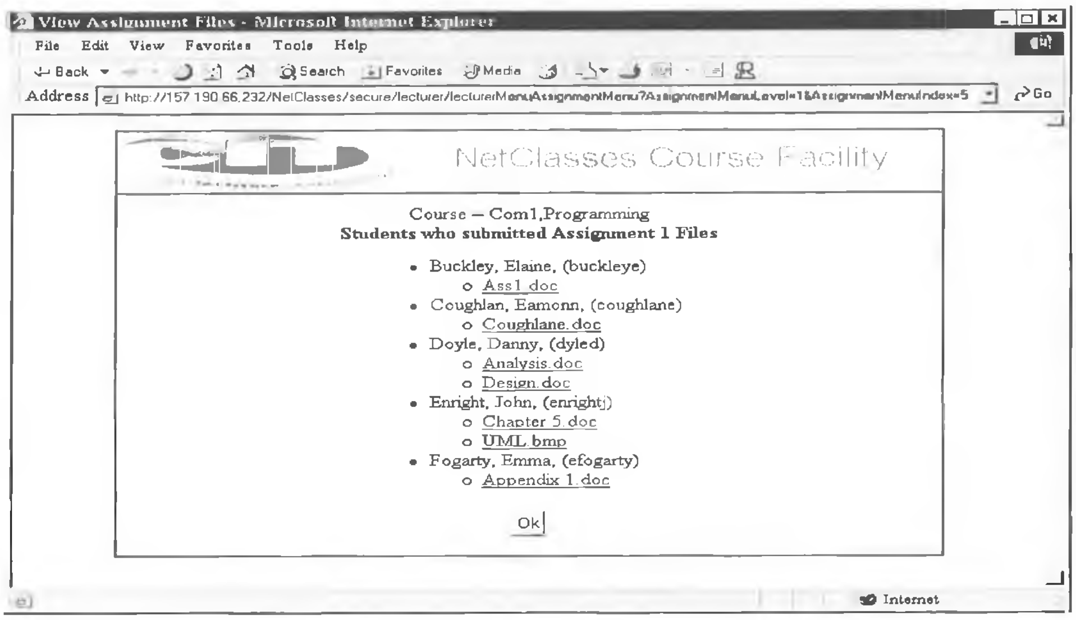
Figure 2.2 Uploading file page.

2.2.3 Posting And Submitting Assignments

Lecturers need to be able to post assignment instructions to the site. To do this might require more than one file. It should be possible to add a file to an assignment that already exists or to create a new assignment for the course. There needs to be a facility to create new assignments and, when doing this, the lecturer should be able to see the list of assignments that are already attached to the course. It is also required to have a date and time limit, past which, students cannot submit assignments. Creating a new assignment should be done on a page where a deadline can be defined. Uploading the students' file should be refused when the time is after the date and time limit set by the lecturer. Students could have more than one file to upload.

Students need to be able to view the assignment instructions. This involves selecting an assignment from the list of assignments associated with the course they are logged on to and then clicking on the file they want to open.

A lecturer should be able to see a page with a list of students who have submitted assignment files and should be able to view a file by making a selection on that page [BAXL02]. The list of available files should still be there, without the lecturer having to reopen the page, after viewing one file. *Fig 2.3* shows the screen with the list of submitted files.


View Assignment Files - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://157.190.66.232/NetClasses/secure/lecturer/lecturerMenuAssignmentMenu?AssignmentMenuLevel=1&AssignmentMenuIndex=5
Go
Internet

NetClasses Course Facility

Course – Com1,Programming
Students who submitted Assignment 1 Files

- Buckley, Elaine, (buckleye)
  - Ass1.doc
- Coughlan, Eamonn, (coughlane)
  - Coughlane.doc
- Doyle, Danny, (dyled)
  - Analysis.doc
  - Design.doc
- Enright, John, (enrightj)
  - Chapter 5.doc
  - UML.bmp
- Fogarty, Emma, (efogarty)
  - Appendix 1.doc

Ok

Figure 2.3 Collecting assignments page.

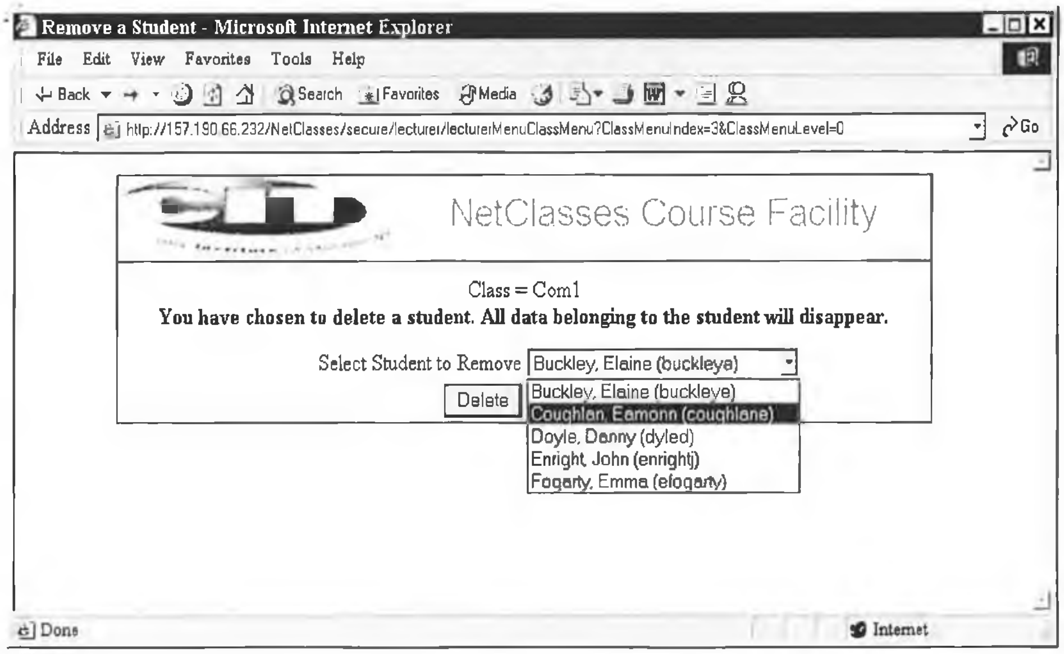
2.2.4 Creating And Updating A Class

Classes are collections of students. In the *Section 2.2.2* it is stated that the lecturer has the job of entering the students' usernames and passwords to create a class. Some constraints need to be placed on the values that can be entered for student usernames. It is a requirement that each student username be unique, so the system has to edit the usernames entered by the lecturer if that username already exists. NetClasses appends a number on to the end of a username if there is a clash and informs the lecturer with a pop-up window.

NetClasses should remember previously entered students. When a lecturer is adding students to a class they should be asked for the unique student IDs. If a student is in the system, but not already in this class, that student should be automatically added to the class list. The lecturer should only be asked to enter details for students not already in the database.

It is possible that a lecturer teaches more than one subject to a class, so it should be possible to add a subject to a class. Similarly it should be possible to delete a subject. Classes might also need to be deleted. Students need to remain in the system when the class is deleted because they may be in another class.

A student may leave a class for one reason or another, so NetClasses needs to cater for that. *Fig. 2.4* shows the screen for choosing which student to delete.


Remove a Student - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://157.190.66.232/NetClasses/secure/lecturer/lecturerMenuClassMenu?ClassMenuIndex=3&ClassMenuLevel=0
Go
NetClasses Course Facility

Class = Com1
You have chosen to delete a student. All data belonging to the student will disappear.

Select Student to Remove: Buckley, Elaine (buckleye)

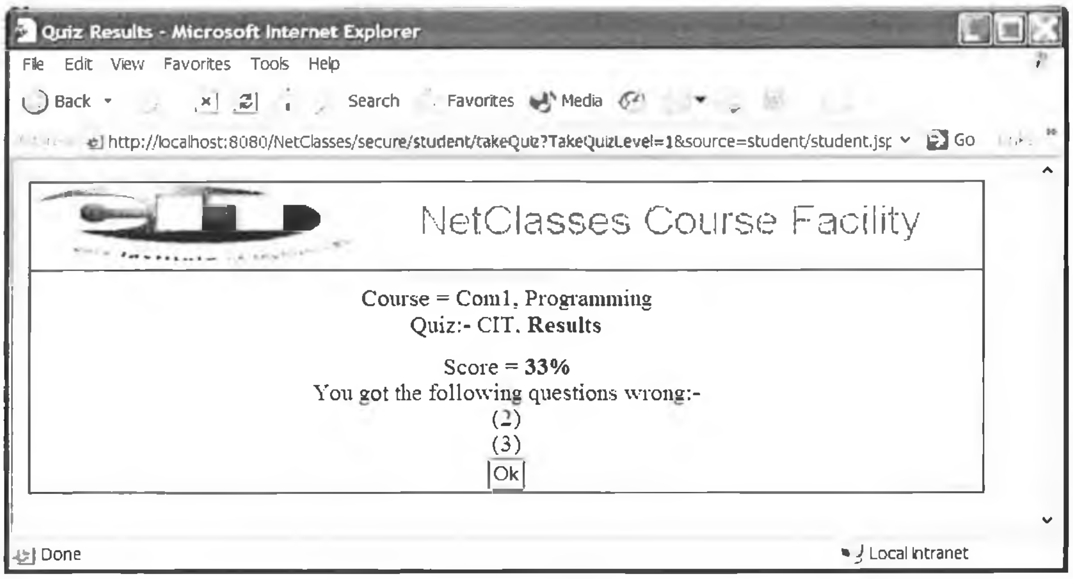
- Buckley, Elaine (buckleye)
- Coughlan, Eamonn (coughlane)
- Doyle, Danny (dyled)
- Enright, John (enrightj)
- Fogarty, Emma (efogarty)

Figure 2.4 Deleting a student page.

2.2.5 Handling Quizzes

Lecturers need to be able to create multiple-choice quizzes and store the correct answers in the system. They also need to be able to edit, add and delete questions.

Students need to be able to take the quizzes and view their score. The marking of the quizzes should be done by NetClasses based on the answers supplied by the lecturer. Students should be able to take quizzes either by clicking on a link or by selecting a menu item. *Fig. 2.5* shows the results screen that is shown after a quiz is marked.


Quiz Results - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
http://localhost:8080/NetClasses/secure/student/takeQuiz?TakeQuizLevel=1&source=student/student.jsp
Go
NetClasses Course Facility
Ok
Done
Local intranet

Course = Com1, Programming
Quiz:- CIT. **Results**

Score = **33%**
You got the following questions wrong:-
(2)
(3)

Figure 2.5 Quiz results page.

It should be possible to place links to quizzes on the notes page so that a reader can review the notes for a particular section of the course and then take a quiz related to that section. It should also be possible to create quizzes and store them in the system without necessarily putting links to them on the notes page.

If a lecturer decides to put a link to a quiz on a notes page, he may want to hide the quiz from the students until such time as it becomes relevant to the material they are learning. There needs to be a facility for hiding quizzes in such cases.

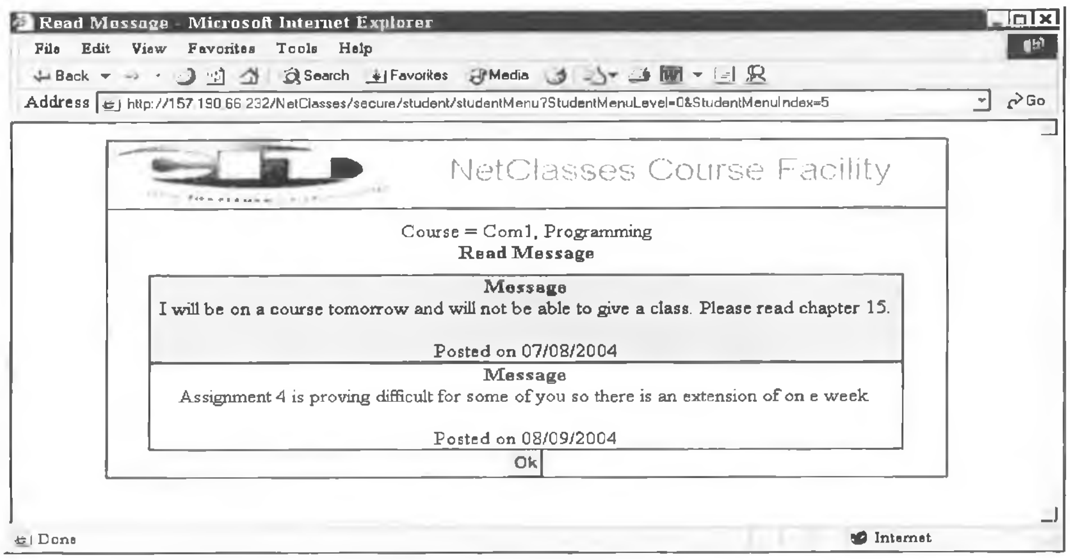
2.2.6 Posting Messages

Lecturers need to be able to type short notes into a web page and the notes should be available to the students.

When students log on and there are new messages, they should be informed about them on the welcome page. There should also be a button on the same page to allow the messages to be read.

A message should be marked as old in the database if a student has read it. There should also be a facility to allow the student to review all old and new messages.

Lecturers need to be able to delete messages so they are no longer visible to the students. *Fig. 2.6* shows a message screen.


Read Message - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://157.190.66.232/NetClasses/secure/student/studentMenu?StudentMenuLevel=0&StudentMenuIndex=5
Go
NetClasses Course Facility
Ok
Done
Internet

Course = Com1, Programming
Read Message

Message
I will be on a course tomorrow and will not be able to give a class. Please read chapter 15.

Posted on 07/08/2004

Message
Assignment 4 is proving difficult for some of you so there is an extension of on e week

Posted on 08/09/2004

Figure 2.6 Reading messages page.

2.3 Conclusion

This chapter analysed the problem area and sets out several requirements that NetClasses must satisfy. The modules of the system are identified and the functionality expected from each module is discussed.

Chapter 3 looks at solutions that are already available and chooses the technologies that are used to build NetClasses.

Chapter 3

Existing Work

3.1 *Existing Technologies*

This section is intended for readers who do not have expertise in the area of web based applications.

Using the Internet for most people today means opening a web browser on a computer and using that to read a web page from another computer. To understand this thesis it is necessary to know a little bit about the technologies that make web browsing possible.

3.1.1 Web Servers

For web browsing to work as described above, one computer has to be the web server [ORFA99]. That computer provides the web pages for those other computers that receive them. A computer that requests the web page from the server and then displays that page for the user to read is called the client computer. Any computer can become the web server if the correct program is running on it. A computer can also be a client if it has a suitable web browser on it.

3.1.2 HTTP

If the web server is providing the web pages and the clients are receiving those pages, then there needs to be certain rules about how requests and responses are exchanged between the computers. The set of rules that are the industry standard for exchanging information between computers on the Internet is called Internet Protocol (IP). The set of rules specifically for requesting web pages and for responding to those requests is called Hypertext Transfer Protocol (HTTP) [FORO03].

3.1.3 HTML And XML

Hypertext Mark-up Language (HTML) is a set of predefined tags that are used to describe what a part of a web page should look like. For example <center> *any text or other page element* </center> will put the text between the two <center> tags in the middle of the section of page it occurs in. The tags are in the web page that is sent to the browser and are interpreted by the browser. They do not themselves appear on the web page but influence how the web page material is presented to the user. XML is like HTML except that the tags are defined in a separate file. Without access to the file, the browser will not know the meaning of the XML tags. For an introduction to HTML and XML see [MORR01].

3.1.4 Dynamic Web Pages

Static web pages are the simplest web pages for a web server to handle. The information that describes a web page is kept in a file on the web server and when the client web browser requests that page the file is sent to the client computer. The web browser interprets the information in the file and creates a page on the client computer screen.

Dynamic web pages are different because the data that is sent from the web server to the client may not be the same every time the web page is requested. For example, a web page that has today's date and time as part of the page would be different at different times. A page that has variable information put into it at the time of request is known as a dynamic web page. NetClasses uses dynamic web pages.

3.1.5 DBMS

The data that is used to assemble dynamic web pages can be kept in a database. When there are multiple users reading and writing that data, a Database Management System (DBMS) is needed. The DBMS is a program that manages the data in a database and makes it available for other programs to use [ROBP01]. NetClasses has a DBMS and it runs on the web server.

3.2 Dynamic Web Pages Technologies

Dynamic web pages and a database are needed to implement NetClasses. This section discusses choosing the technology to provide the dynamic web pages.

There are several environments that can be used to implement dynamic web pages. The best known are [RAJA02]:

- Common Gateway Interface (CGI).
- Personal Home Page (PHP).
- Active Server Pages (ASP).
- Java Server Pages (JSP).

These environments are presented in *Sections 3.2.1, 3.2.2, 3.2.3 and 3.2.4*.

3.2.1 CGI

CGI is the oldest of the technologies listed above. It uses a collection of environment variables and a few other strategies to provide the gateway between the language that the application logic is expressed in (often C or Perl) and the HTTP protocol. This gateway approach is not particularly object-oriented because of the use of environment variables that are in effect global variables. Large projects can be difficult to manage with this old style of modelling the data.

In earlier versions of CGI there was a further drawback. This related to the scalability of CGI applications. Each HTTP request spawned a new process with its own memory and other resource requirements [COLE00]. Any application that processed a lot of requests was inefficient as a result.

The two drawbacks mentioned above led to the decline in use of CGI in favour of more modern technologies. The problem of having separate processes for every request has now been addressed [FIEL02]. Newer CGI implementations use threads but even this has not helped CGI to regain its position as the technology of choice for dynamic web page deployment.

3.2.2 PHP

PHP works by including code script in the web pages. This is known as server side scripting because the code script is interpreted and executed on the server side. The effect of executing the server side script is to generate the HTML stream that is sent to the client.

PHP is both open-source and cross platform. As a server-side scripting language PHP must be written in the web page. This means that the application logic is included in the web page file on the server. A three-tier architecture, presented in *Section 4.1*, can still be used because the logical separation of the processing layer and data layer does not mean that they have to be physically separated. However, if the job of page designing is to be separated from that of application programming, it is better that the scripted logic and the static page layout descriptors be separated [GANG02]. It should be easy for the page designer to see which parts of a page have to be altered to get the design and layout required. This is not possible with a pure server-side scripting language like PHP because the code and HTML tags are interspersed [MELO03].

PHP does everything that needs to be done by NetClasses and would have been used had a better alternative not been found.

3.2.3 ASP

ASP is Microsoft's technology for producing dynamic web pages. As such it is not cross platform and only works with Microsoft web servers.

ASP supports many server-side scripting languages [MARA00(a)], of which Visual Basic Script (VBScript) is the most popular. Used with VBScript the business logic can be expressed in the script or in ActiveX controls. ActiveX controls can be referenced in dynamic web pages. This facilitates the separation between the code describing the page and the code describing the dynamic content.

Using ActiveX controls, the business logic can be implemented in an Object Oriented solution.

ASP does everything NetClasses requires in a modern object oriented way but is not cross platform and so would only be used if nothing else would work.

3.2.4 JSP

JSP allows both scripting in Java and referencing Java objects in the web page. The programmer has access to the complete set of Java Application Programmer Interfaces (APIs) for use in accessing databases and processing the data. Of course all the data processing is done on the server side before the page is created and sent to the client. The extensive Java Graphical User Interface (GUI) objects such as are available in javax.swing are of no use to the programmerat in that case.

HTTP requests and responses are scheduled and dealt with by the Servlet Container and this runs on the Java Virtual Machine (JVM) on the server side. The Servlet Container supplied free by Sun Corporation is called Tomcat and needs no other web server to handle HTTP requests. However Tomcat can be made to work with several of the most popular web servers including Microsoft's Internet Information Services (IIS) and Sun's Apache.

XML type tags on the web page that reference Java beans can achieve separation of business logic and page descriptors. Beans are simply Java objects that follow certain naming convention of the get( ) and set( ) methods. The logic is encapsulated in the beans. Very little code appears on the web page other than tags that access bean methods.

It might be expected that JSP would be slower than the other alternatives because Java is an interpreted language. Java is compiled to machine independent byte code and is interpreted by the JVM during the execution phase. PHP, Perl and VBScript are interpreted also so there is no speed disadvantage against these.

There is a speed problem the first time any JSP page is requested. Every JSP is compiled into a servlet [FIEL02]. The servlet has both a .java and a .class file. The first time the page is accessed (assuming it is already interpreted, compiled and deployed as a servlet) the .class file must be loaded into memory on the server. This takes a considerable time. All subsequent requests for that page by any client are answered quickly because the class is either in memory or virtual memory.

JSP does everything NetClasses requires in a modern object oriented way. It is also both cross platform and open source so it is used to implement NetClasses.

3.2.4.1 Java Server Pages Environment

Java Server Pages technology is an extension of servlet technology. Both of them are from Sun Corporation. JSP pages are compiled to produce servlets. According to [INCE02], "Servlets are snippets of Java code which are loaded into a Web server and are executed when an HTTP command is processed by the server".

The JSP pages contain Java code and can contain references to Java objects that are defined in other Java code files in the same application. The Java code is separated from the rest of the page material by surrounding the Java section with <% and %> markers. The creator of a JSP page has access to all the facilities in the Java programming language. Good web based application design dictates that the amount of Java code in a Java Server page is kept to a minimum [FIEL02]. The code that prepares the information to go on the web page is kept in other Java files in the application. The prepared information is passed to the JSP page before it is sent to the client. A certain amount of Java coding is required in the page to collect the information and put it on the page.

For JSP to work there must be a Java servlet container running on the web server. The one supplied free by Sun Corporation, Tomcat, works well for NetClasses. Each JSP page is compiled to produce a corresponding servlet. This is done by the Java servlet container and needs no input from the application developer. The Java servlet container looks after all the scheduling of HTTP requests and responses for the application. An advantage of servlet technology is that the container keeps information about the HTTP session on the server side. The information is available within the application from the time the client opens the browser and makes the first request until the user logs off or closes the browser session [FIEL02]. This facility was not available with older technologies and developers had to use cookies or other strategies to keep track of the session information on the client side and continually sent it to the server. The Java servlet container used to develop NetClasses can even resume a session after shutting down and restarting the web server.

3.3 The Database Management System

The data server and the web page server need to be both Java based applications so that they are cross platform. If the classes that make up the DBMS application are made available to the servlet container, then NetClasses can use the facilities directly. This method of using the DBMS classes is called running in embedded mode. One widely used Java database is Cloudscape but this is no longer available free of charge. Therefore, another DBMS that supports both Structured Query Language (SQL) and Java Database Connectivity (JDBC) is used. It is called McKoi, is free, open source and is available at http://mckoi.com/database/. It can also be embedded in another Java application.

Not all DBMSs are entirely thread safe. However, as long as an instance of the ResultSet or Statement objects are used by only one thread at a time they will not interfere with different ResultsSet or Statement objects used in another thread. This can be achieved by using a Statement as synchronised and opening and closing the ResultSet in the same thread. The ResultSet objects can also be created in insensitive mode, that is the results are a snapshot of the data at the time of reading and do not change when the database is updated.

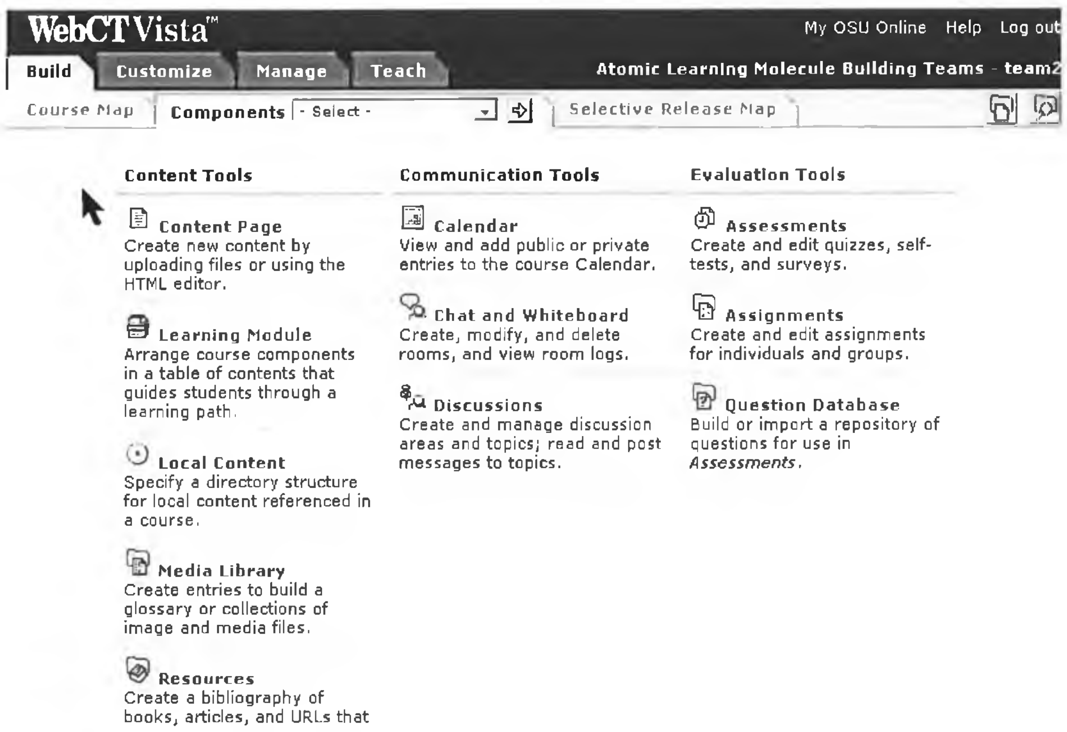
Any database supporting SQL and JDBC can be used in NetClasses. If it not used in embedded mode then it must support client-server architecture. McKoi supports client-server architecture but not in embedded mode. In NetClasses, it is used in embedded mode but not in a client-server architecture. This might seem strange at first because NetClasses is essentially a client-server application. The Java Servlet container, however, is the actual application that accesses the database facilities supplied by the DBMS. Therefore, the clients of NetClasses are clients of Tomcat and not of the DBMS. The design of NetClasses ensures that there are only two connections to the database opened at any time by Tomcat. One connection is for group transactions that must all be completed before the changes are committed to the database and the other one is for simple transactions. There can be several SQL statements requesting access to the database at the same time but each SQL statement is synchronised so there can be only one instance of each SQL Statement accessing the data at any time.

3.4 *Existing Applications*

This section looks at some of the big commercial web content management tools, one free open source application and two custom built applications used in CIT and University College, Cork.

3.4.1 WebCT (www.webct.com)

WebCT is a commercial e-learning tool used in several third level colleges. CIT has this system on campus. There are facilities in the enterprise edition for integrating with the college Student Records and other college wide applications. It is also possible for students to modify their view of the notes and even to annotate them. The functionality of WebCT that falls within the specifications of NetClasses is considered in this thesis. *Figs. 3.1a and 3.1b* show some of the options available to a lecturer using WebCT and NetClasses.



Content Tools

Content Page
Create new content by uploading files or using the HTML editor.

Learning Module
Arrange course components in a table of contents that guides students through a learning path.

Local Content
Specify a directory structure for local content referenced in a course.

Media Library
Create entries to build a glossary or collections of image and media files.

Resources
Create a bibliography of books, articles, and URLs that

Communication Tools

Calendar
View and add public or private entries to the course Calendar.

Chat and Whiteboard
Create, modify, and delete rooms, and view room logs.

Discussions
Create and manage discussion areas and topics; read and post messages to topics.

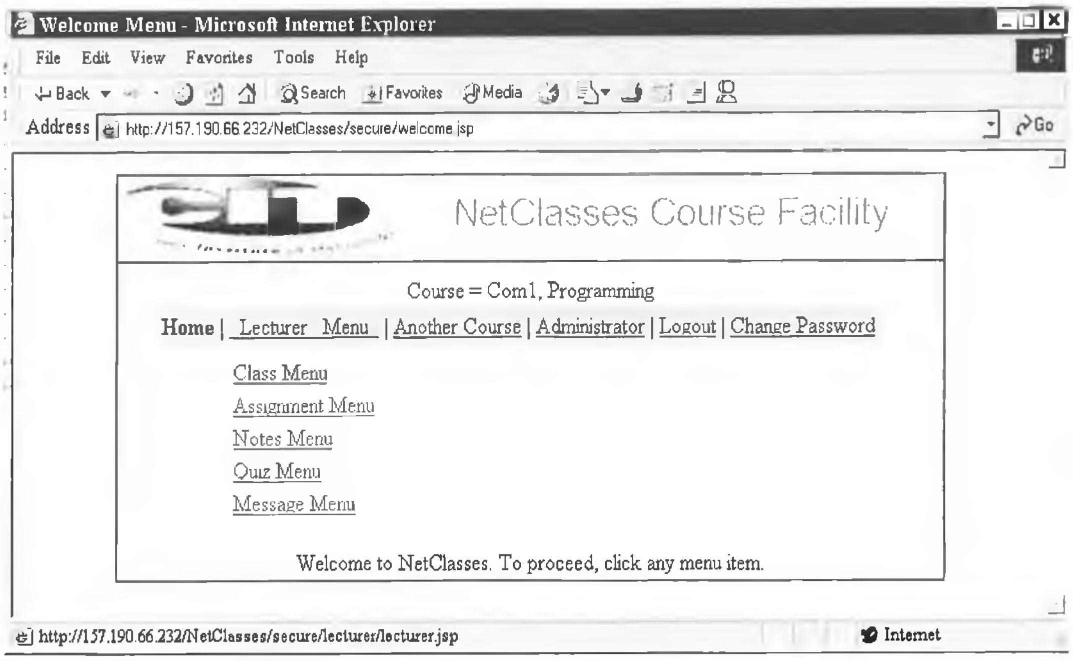
Evaluation Tools

Assessments
Create and edit quizzes, self-tests, and surveys.

Assignments
Create and edit assignments for individuals and groups.

Question Database
Build or import a repository of questions for use in *Assessments*.

Figure. 3.1a Some of the WebCT options.


Welcome Menu - Microsoft Internet Explorer

Address http://157.190.66.232/NetClasses/secure/welcome.jsp

NetClasses Course Facility

Course = Com1, Programming

Home | Lecturer Menu | Another Course | Administrator | Logout | Change Password

Class Menu
Assignment Menu
Notes Menu
Quiz Menu
Message Menu

Welcome to NetClasses. To proceed, click any menu item.

http://157.190.66.232/NetClasses/secure/lecturer/lecturer.jsp

File Edit View Favorites Tools Help
Back Search Favorites Media
Go
Internet

Figure. 3.1b Some of the NetClasses options.

Assignment uploading and lecturer notes uploading have much the same functionality in both systems.

In the matter of students joining a course there is a major difference between the NetClasses approach and that taken by WebCT. NetClasses allows lecturers to create class groups and populate their classes with students. The administrators for the site can only do student entry in WebCT courses.

WebCT instructions on using the application say:-

"**Warning:** Avoid using your browser's Back and Forward buttons to navigate through WebCT CE. Instead, use the *Course Menu, Control Panel, Menu Bar* and breadcrumbs". The forward and back buttons can be used in NetClasses without adverse effect (see *Section 4.5*).

3.4.2 Moodle (moodle.org)

Moodle is a free, open source course management system for online learning. It is written in PHP and requires the client to have session cookies enabled.

The Moodle notes page allows users to view single files or to click on folders to see a list of files. A Moodle page showing a link to a lecturer's notes folder is shown in *Fig 3.2* .

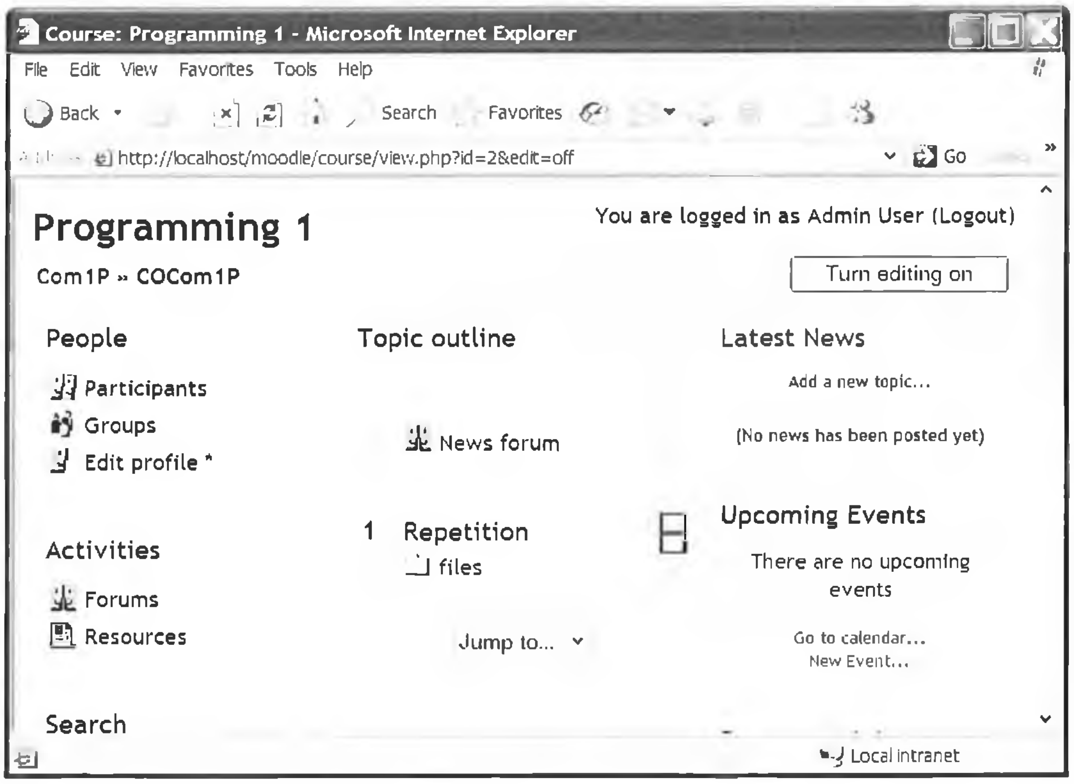


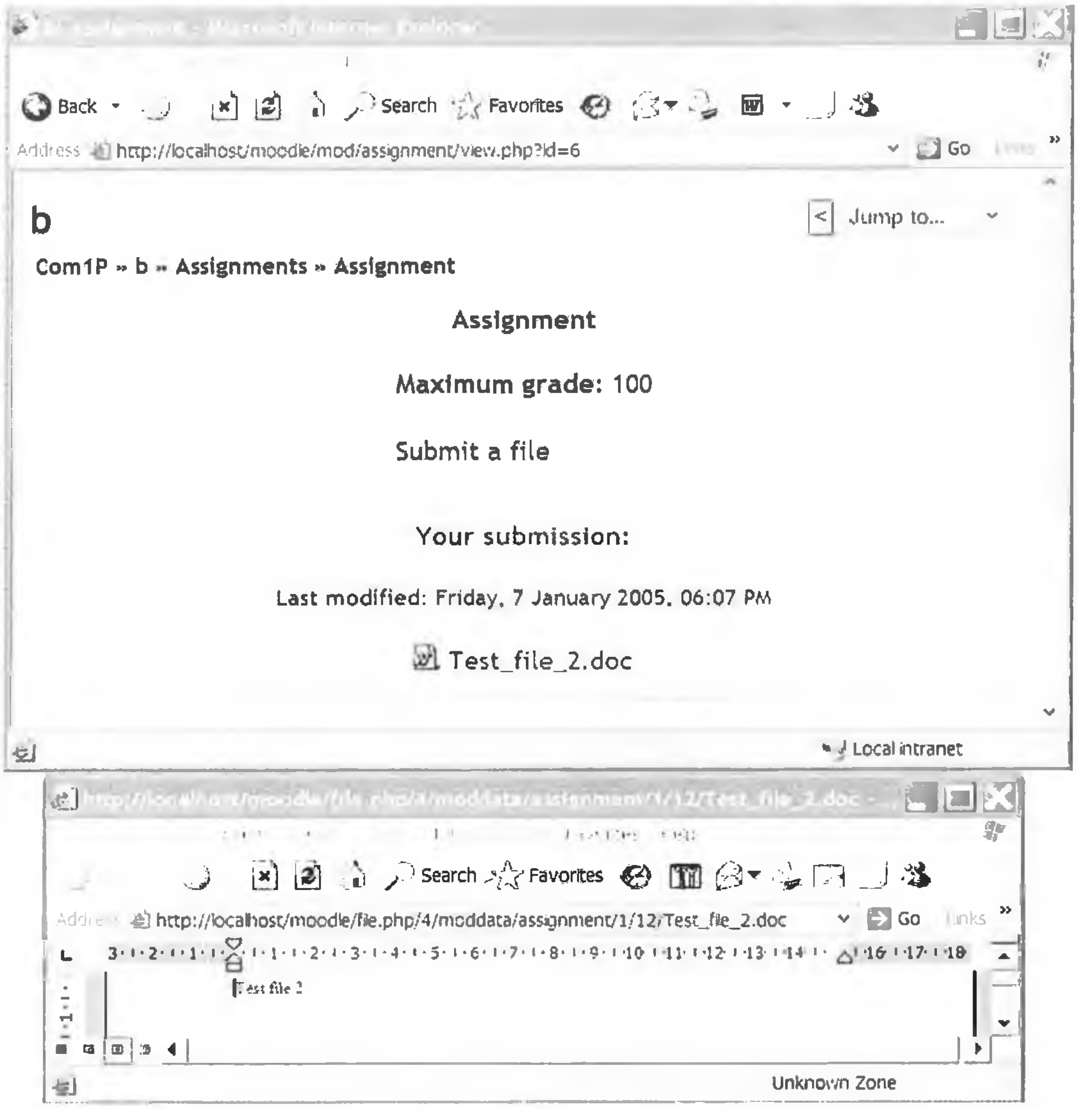
Figure 3.2 Moodle notes page.

Moodle can be set up to allow users to register themselves in the system. The user must supply an email address and a key is sent to that address so that the user can be registered. Administrators can also enter students into the system. Lecturers can enrol students in their course who are already in the database. NetClasses allows lecturers who are not administrators to enter students into the system and so is different to Moodle in that respect.

In Moodle, any user who is already in the system can enrol themselves on a course. If the lecturer does not want any user enrolling on a course then the course can be protected by a password so that only those who have the password can enrol themselves.

Submitting assignments is much the same in both NetClasses and Moodle except that Moodle only allows one file to be submitted whereas NetClasses users can submit multiple files.

Security is a bit slacker in Moodle than in Netclasses. The current version at time of writing, 1.4.3, allows a student who is enrolled on a course to view a file submitted by another student of the course simply by pointing the browser at that file. The student who does that would have to know the name of the target file and also the user number of the student whose file he is targeting. The user numbers are generated by Moodle at the time the users are entered into the system and are assigned sequentially. *Fig. 3.3* shows a student viewing his own submitted file and *Fig. 3.4* shows the same student viewing the file submitted by another student. The folder was changed from 12 to 11 and the file name changed from Test_file_2.doc to Test_file_1.doc in the address bar to achieve this.


Address http://localhost/moodle/mod/assignment/view.php?id=6

b

Com1P » b » Assignments » Assignment

Assignment

Maximum grade: 100

Submit a file

Your submission:

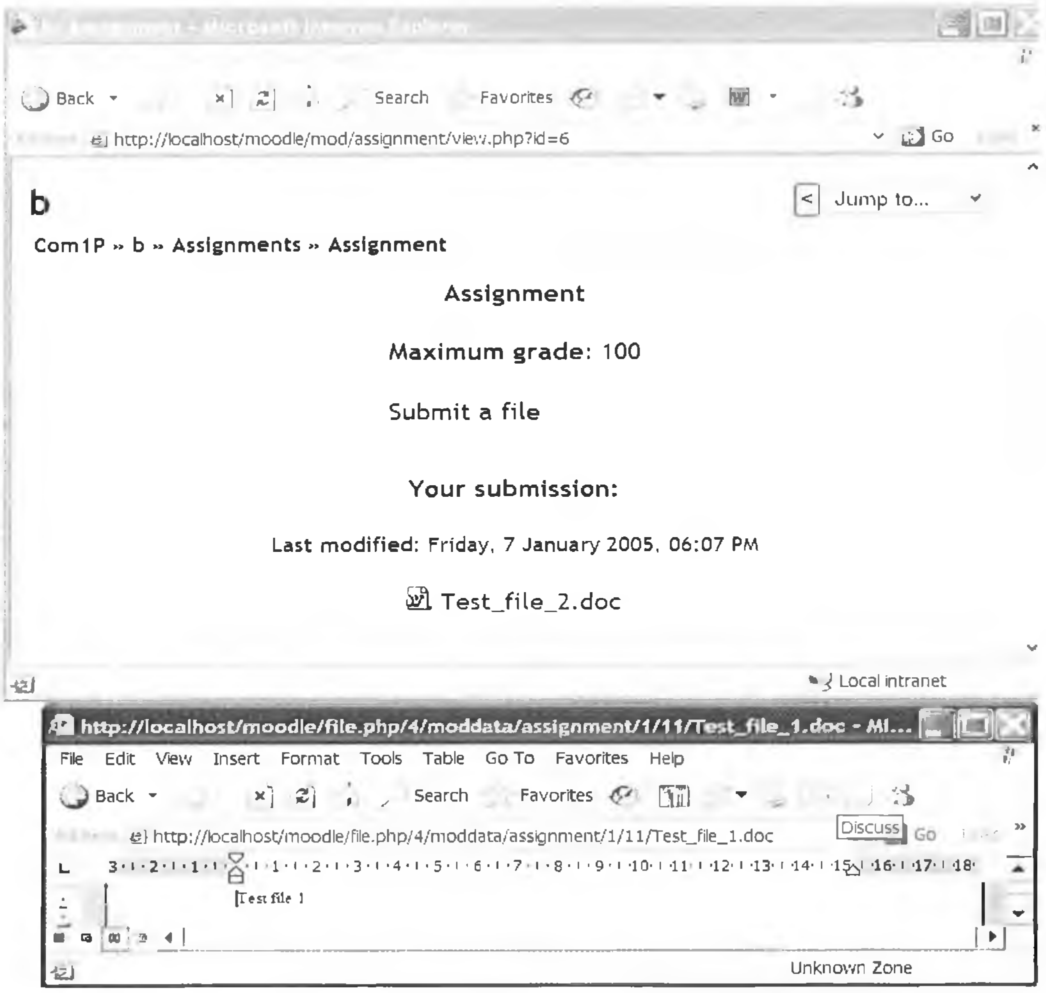
Last modified: Friday, 7 January 2005, 06:07 PM

Test_file_2.doc

Address http://localhost/moodle/file.php/4/moddata/assignment/1/12/Test_file_2.doc

Test file 2

Figure 3.3 A Moodle student viewing his own file.


http://localhost/moodle/mod/assignment/view.php?id=6

b

Com1P » b » Assignments » Assignment

Assignment

Maximum grade: 100

Submit a file

Your submission:

Last modified: Friday, 7 January 2005, 06:07 PM

Test_file_2.doc

http://localhost/moodle/file.php/4/moddata/assignment/1/11/Test_file_1.doc

Test file 1

Figure 3.4 A Moodle student viewing another student’s file.

3.4.3 TopClass LCMS (www.wbtsystems.com/products)

Another commercial application, TopClass Learning Content Management System does a lot more than NetClasses. For example it can dynamically deliver courses based on learners’ individual knowledge level. *Fig. 3.5* shows the wide range of capabilities of this product.

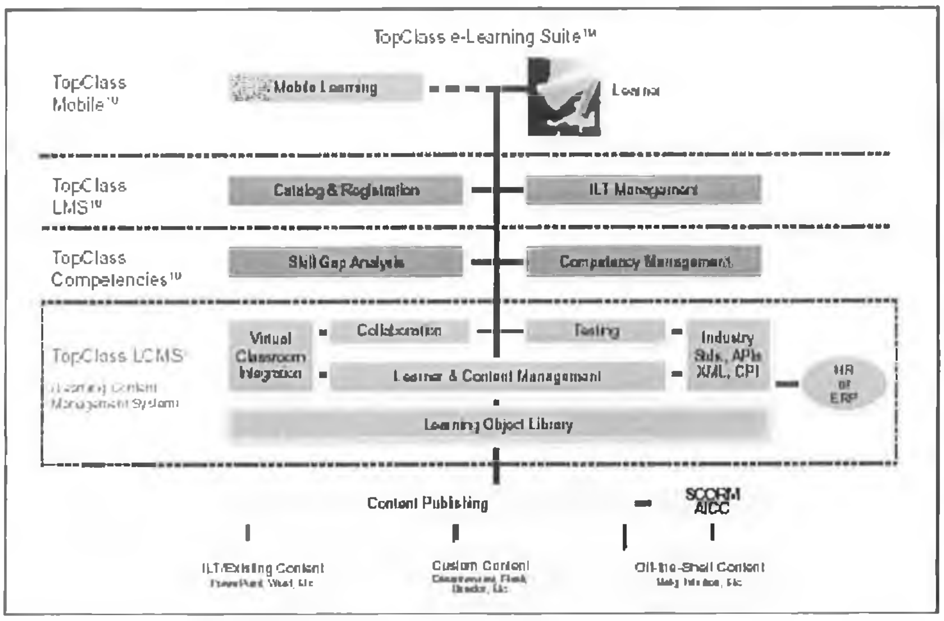


Figure 3.5 TopClass overview.

Different approaches are taken by TopClass and NetClasses with regards to the storage of course notes. NetClasses notes are stored in a folder for the individual subject, which is a subfolder of the class folder, which is itself a subfolder of the lecturer's folder. TopClass has a single Learning Objects library where lecturers can access learning objects to associate with their courses. Lecturers can add new learning object to the library. A learning object is a file e.g notes, picture or sound. TopClass has a more flexible system than NetClasses in that regard because TopClass learning objects are reusable whereas NetClasses notes files are only accessible to students of the course they are stored in.

TopClass provides a Graphical User Interface that is customizable by the lecturer and can be used by the student to access notes and other course material.

TopClass has a number of methods of registering students. One of these even allows learners to register themselves. It does not necessarily depend on a central administrator to enrol students. NetClasses allows only lecturers to register students and so is more restrictive than TopClass.

3.4.4 Blackboard (www.blackboard.com)

Blackboard is another commercial application and again is more powerful than NetClasses, for example there is a resource repository where instructors can log on and download teaching material to include in their course. *Fig. 3.6* shows the options available to a lecturer building a course.

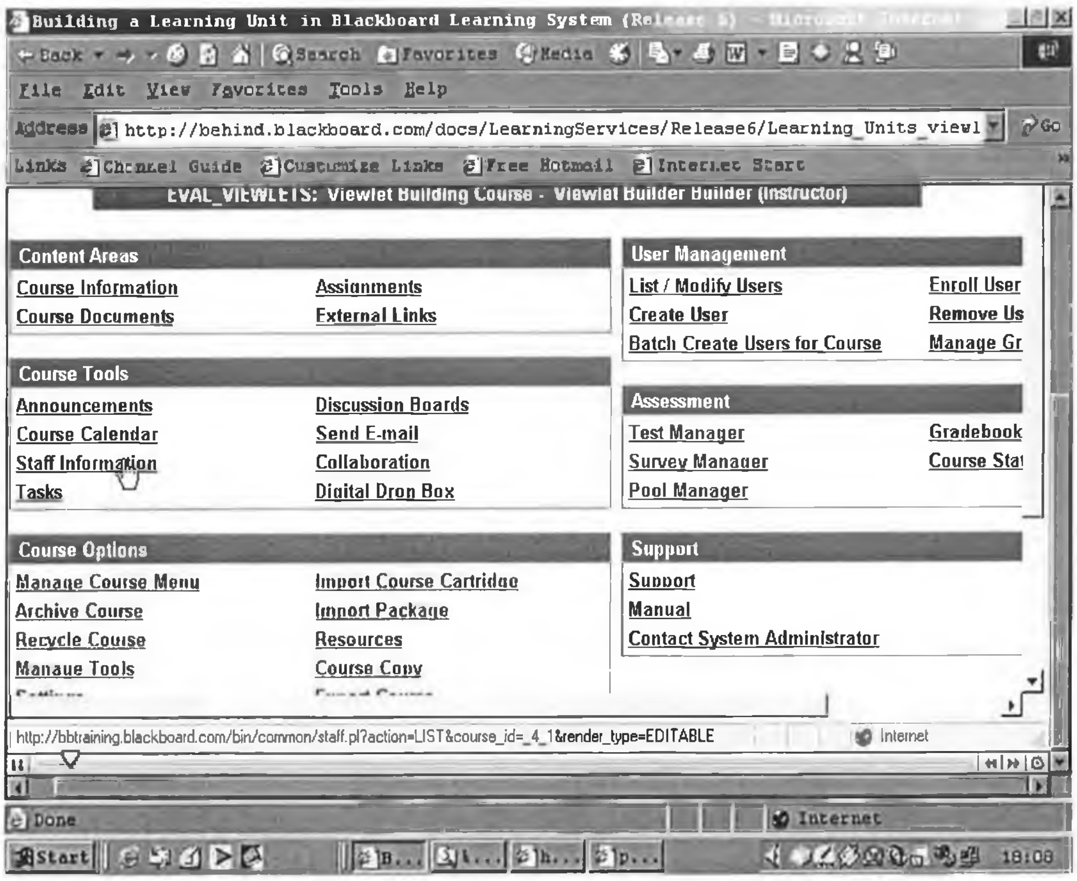


Figure 3.6 Blackboard course building options page.

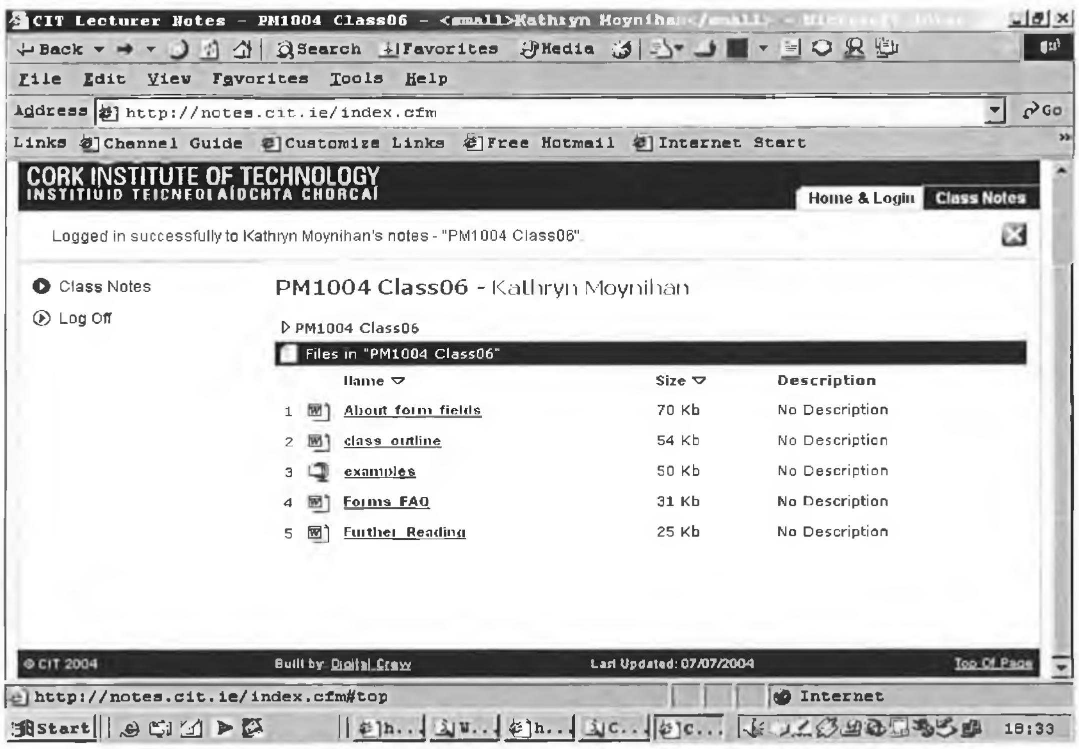
Blackboard provides a Graphical User Interface that is customizable by the lecturer and can be used by the student to access notes and other course material.

This application takes the same approach as NetClasses in adding students to a course. Lecturers can create courses and add students to them.

3.4.5 Notes (notes.cit.ie)

This notes posting facility is used in CIT to cater for lecturers who do not want the full functionality of WebCT but who simply want to make their notes available to students without having to learn to use a complicated interface. It was built by Digital Crew (www.digital-crew.com) at the request of CIT.

Notes has similar functionality to NetClasses for uploading course notes. The interface for viewing notes is very different. Notes has a flat list of the files whereas NetClasses can group files under a heading so as to keep files in logical bundles. *Fig. 3.7* shows the Notes file viewing interface.


CIT Lecturer Notes - PM1004 Class06 - <small>Kathryn Moynihan</small> - [illegible]
Back
Search
Favorites
Media
File Edit View Favorites Tools Help
Address http://notes.cit.ie/index.cfm
Go
Links Channel Guide Customize Links Free Hotmail Internet Start
CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ
Home & Login
Class Notes
Class Notes
Log Off
http://notes.cit.ie/index.cfm#top
Internet
Start
18:33

Logged in successfully to Kathryn Moynihan's notes - "PM1004 Class06"

PM1004 Class06 - Kathryn Moynihan

PM1004 Class06

Files in "PM1004 Class06"

	Name	Size	Description
1	About form fields	70 Kb	No Description
2	class outline	54 Kb	No Description
3	examples	50 Kb	No Description
4	Forms FAQ	31 Kb	No Description
5	Further Reading	25 Kb	No Description

© CIT 2004 Built by: Digital Crew Last Updated: 07/07/2004 Top Of Page

Figure. 3.7 Notes file viewing page.

Notes is not meant to be an assignment manager as well, so it does not have student uploading or individual passwords for students.

3.4.6 Extranet [RUSS00]

Extranet is the product of an MSc. project in UCC. The project was carried out by Gavin Russell and Jannejte van Leeuwen in 2000 as part of their thesis for an MSc. in Multimedia Technology.

The authors designed and produced a site that provides links to various freeware applications (in all cases modified to make them have the right look and feel). They wrote the file upload facility from scratch.

This product does not use a DBMS and so is not comparable to NetClasses. The idea was to research suitable freeware products and to gather them together in one site. The suite of applications can provide access to notes, assignment uploading and reporting and a bulletin board. It does not have online administrative functions like NetClasses has. Extranet does not use a login that allows access to the whole site but has separate login pages for different tasks.

The technology used in Extranet is CGI and the dynamic web pages are written in Perl.

Extranet is fundamentally different to NetClasses in that it is built for a single course i.e. one class with several lecturers whereas NetClasses is for multiple courses hosted by multiple lecturers.

3.5 Conclusions

This chapter reviewed various technologies used in the field of web content management and some of the available applications. Some aspects of the existing solutions are compared to the facilities offered by NetClasses.

NetClasses cannot compete with the three big commercial applications on every front. The notes uploading facility in NetClasses is similar in functionality to all the applications reviewed. The NetClasses interface to the course notes and submitted assignments is different from those reviewed. NetClasses has an interface for course material access that is intuitive to use and easy to learn. The NetClasses method of adding students is as useful as any for institutions such as CIT.

Chapter 4 presents theoretical issues behind NetClasses design and some of the decisions made about site architecture and deployment.

Chapter 4

Architecture And Design

4.1 Three Tier Architecture

Darrel Ince [INCE02], in his book, Developing Distributed and E-Commerce Applications, introduces the three-tier architecture for distributed systems with a database layer.

When web pages are not dynamic, two-tier architecture is often used. This consists of one layer with the presentation and logic elements of the system and the second layer with the data. The HTML code in the pages is both the presentation and logic layer where the only logic is deciding which page to show next and the data (web pages or other data such as animations or sound) is stored as files on the web server.

When using dynamic web pages there is more processing to be done on the data before it is presented. In that case it is no longer convenient to have the logic layer in the code that the client uses to display the pages. Either each client would have to maintain the data processing code or the code would have to be sent by the server along with the web page. Having data processing code on the client side can be problematic because the enterprise logic tends to change over time. The three-tier architecture, shown in *Fig. 4.1*, adds an extra processing layer between the presentation layer and the data layer. So before the presentation data is sent to the client, it is processed on the server side to reflect the dynamic situation.

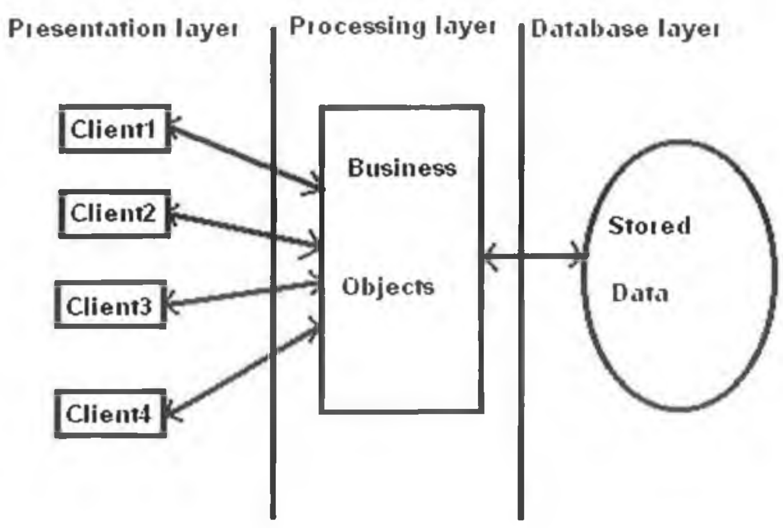


Figure 4.1 Three tier architecture for dynamic web pages.

The three-tier architecture as applied to dynamic web pages has the presentation layer in the HTML code sent to the browser, as well as the browser itself. The processing layer is encapsulated in the business objects. The data layer is the files that describe the web pages plus the relational database. The data in the database are used by the business objects either to create the pages or to decide which pages to create.

The processing layer needs more explanation. The business objects that implement the processing layer contain the logic of the application. User input, in the form of browser HTTP requests, signals one of the business objects. This happens by the action of intermediate software known as middleware. The business objects collect the data required and create the HTML code that will eventually create the web page on the client side. Again, because of middleware, the business objects are able to give an HTTP response with the dynamically created HTML coded web page to the client.

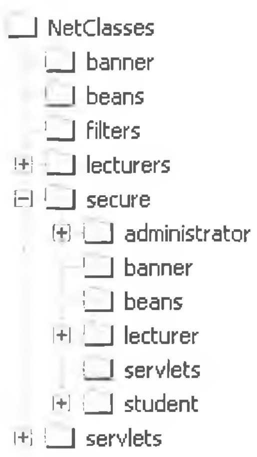
With Java Server Pages, the platform used to implement NetClasses, the servlet container (Tomcat in this case) provides the middleware that handles request and response scheduling. The servlet container allows the programmer to use HttpServletRequest and HttpServletResponse objects in the business object code. The application logic can be applied to any user input attached to the HttpServletRequest and the Java Server Pages can then be dynamically created using the data made available by the business objects.
HttpServlet is a class in the Java Server Page platform. Any object that is created to extend HttpServlet can use doGet( ), doPost( ) or service( ). Each of these methods have HttpServletRequest and HttpServletResponse objects as parameters and that is the interface between the HTTP protocol and the business logic of any application using Java Server Pages [FIEL02].

The NetClasses application uses the three-tier architecture because it fits well with this type of application. For the system to be flexible in its deployment, each user should not be required to have NetClasses application logic on their terminals. Users can then use the application from anywhere once they can get access to a web browser. For the same reason of flexibility, cookies are not used. Without cookies the behaviour of NetClasses is the same no matter which terminal the user is accessing from. The web browser presenting web pages, matches the presentation layer very well in that it is only used to facilitate the user interaction with the system. Some processing logic can be sent to the client computer in the form of Java Script embedded in the web pages. The modern browsers understand Java Script and so it is used in NetClasses to control and check user input.

The boundary between the business logic and data layer is not so clearly paralleled in the physical deployment of NetClasses files. The data held in the relational database is separated from the business objects but some of the Java Server Pages (which are data) contain references to business objects and even some Java code. However, the logical separation is there even when both logic and data exist on the same page. The logic is used on the server side before sending to the client. Only the presentation code gets sent.

4.2 File Structure

There are quite a number of files in NetClasses and some structure needs to be imposed on the files both to make maintenance of the system easier and to allow access to parts of the system to be controlled. The file structure for NetClasses is shown below in *Fig 4.2*.



```
NetClasses
   banner
   beans
   filters
[+] lecturers
[-] secure
   [+] administrator
       banner
       beans
   [+] lecturer
       servlets
   [+] student
[+] servlets
```

Figure 4.2 Netclasse file Structure.

Inside the secure section there is one folder for each of the three user types; administrator, lecturer and student. The lecturers folder outside the secure area is designed to hold all the lecturer notes and the assignment files. It is put outside the secure area because it could be left unsecured if the lecturer decided that there should be open access to the notes. Open access is not implemented in that way in NetClasses but the file structure is there.

4.3 Site Navigation.

The web is a navigational system, navigation is difficult and it becomes necessary to provide the user with navigational support beyond the simple hyperlinks [NIEL00]. With this in mind, NetClasses pages other than the logon page and file viewing page, are designed to be of two types. One with a menu bar so that the user can jump to different sections of the site without going back to a menu page. The other type is one where the user can only choose to go ahead with the task at hand or to cancel. The navigation of the site is mainly menu driven. The jump menu bar is meant to be for situations where the user wants to shortcut to the main menu or to carry out tasks that should logically be done from any position where the user is free to choose what to do next.

The type of page with menu bar is shown in *Fig. 4.3* and the type of page with a decision to go on or to cancel is shown in *Fig. 4.4*.

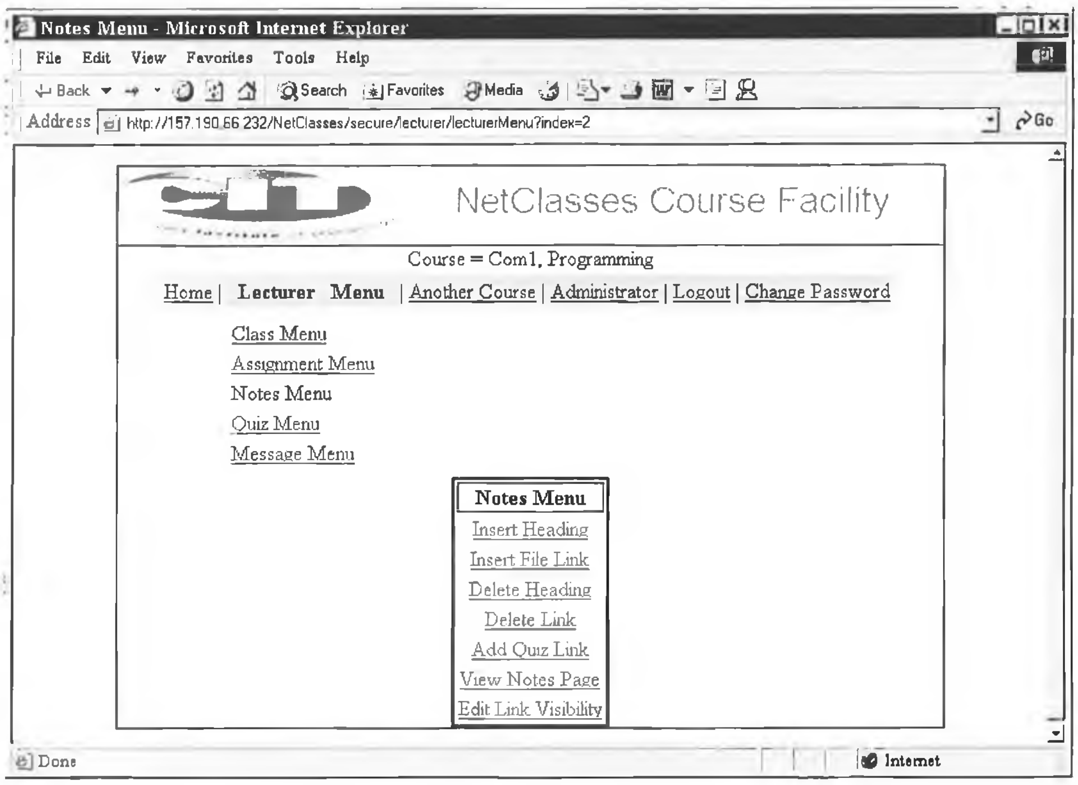
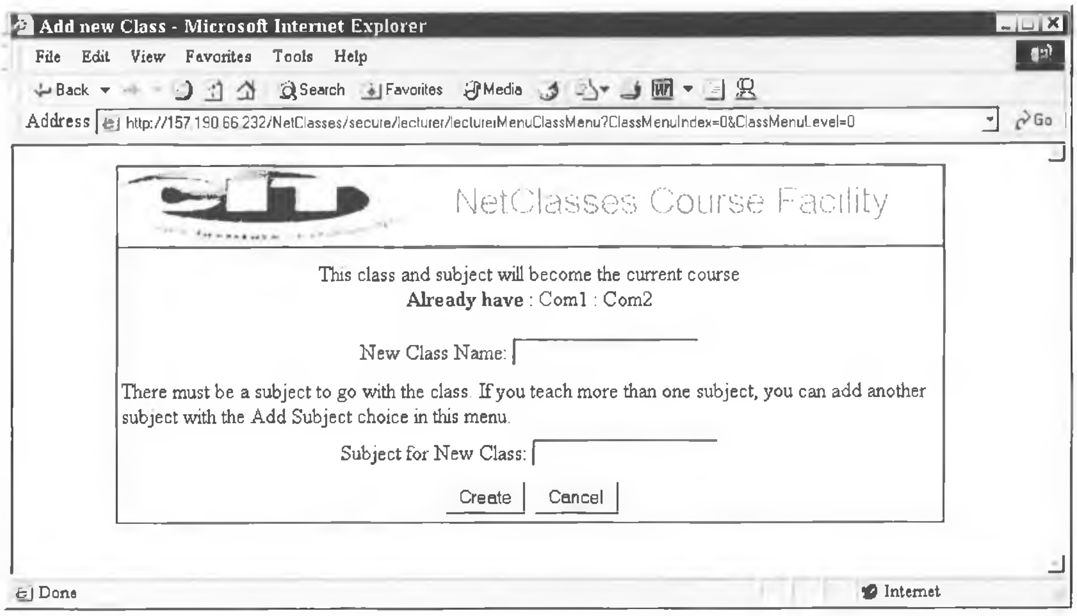

Notes Menu - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://157.190.66.232/NetClasses/secure/lecturer/lecturerMenu?index=2
Go
NetClasses Course Facility
Course = Com1, Programming
Home | Lecturer Menu | Another Course | Administrator | Logout | Change Password
Class Menu
Assignment Menu
Notes Menu
Quiz Menu
Message Menu
Notes Menu
Insert Heading
Insert File Link
Delete Heading
Delete Link
Add Quiz Link
View Notes Page
Edit Link Visibility
Done
Internet

Figure 4.3 The type of page with menu bar


Add new Class - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address
Go
Create
Cancel
Done
Internet

http://157.190.66.232/NetClasses/secure/lecturer/lecturerMenuClassMenu?ClassMenuIndex=0&ClassMenuLevel=0

NetClasses Course Facility

This class and subject will become the current course
Already have : Com1 : Com2

New Class Name:

There must be a subject to go with the class. If you teach more than one subject, you can add another subject with the Add Subject choice in this menu.

Subject for New Class:

Figure 4.4 The type of page with decision buttons

Each user has a main menu that is designed to cater for that user's requirements. Depending on the role of the user, different menus present themselves. An administrator can access two menus, the administrator menu and the lecturer menu. Lecturers and students can access the lecturer and student menus respectively.

In [NIEL00] Jakob Nielsen identifies the three most important questions to be able to answer when browsing a web site as:

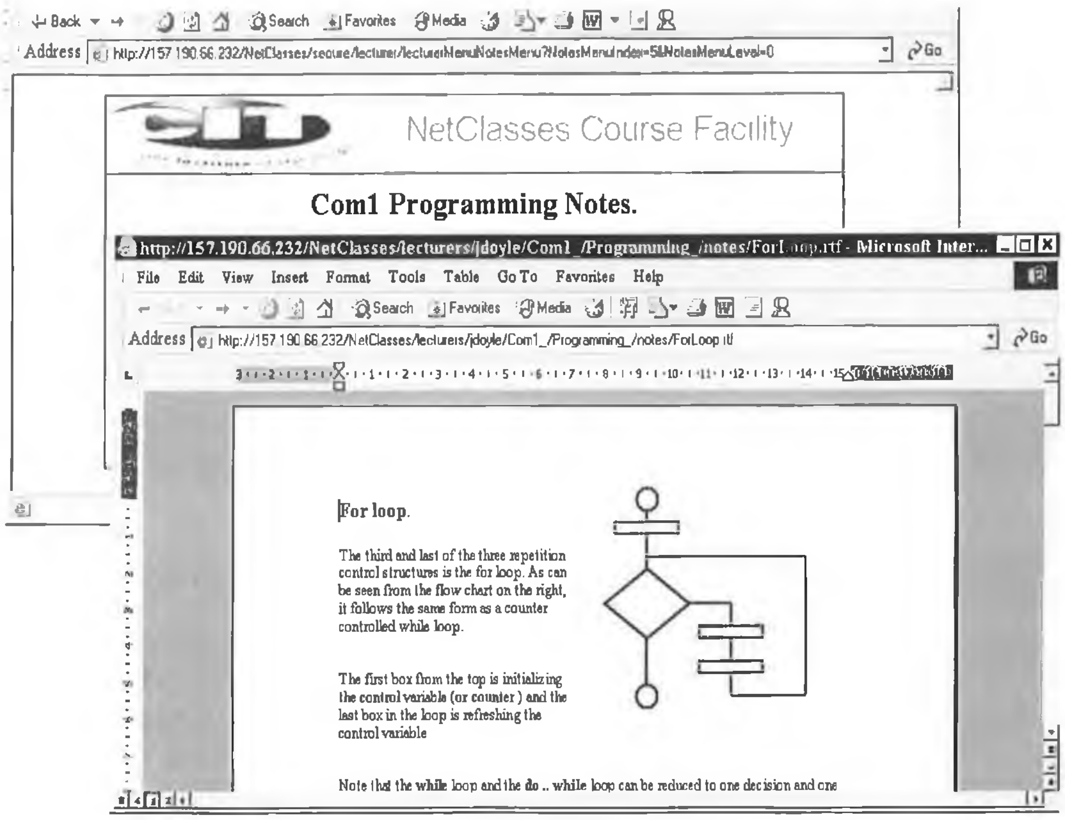
1. Where am I?
2. Where have I been?
3. Where can I go?

He goes on to say that the first question should refer both to the Internet as a whole and to the particular site being browsed. NetClasses provides global information by including the site banner on the top of each page. The question of local whereabouts in the site is answered when a user is in a page like *Fig. 4.3* because the name of the menu is given on the page. The whole of the NetClasses site, except the pages for logging in, viewing course notes and reading assignment files are administrative. Administrative pages perform some task that changes information in the database. The architecture of most of the site is meant to be task oriented and is menu driven. The menu pages have clickable menu options and when any task is finished, the next page presented is the menu page again. In some intermediate pages, when a task requires more than one page to complete, the question of "where am I locally?" is not always answered directly on the page.

The second question can be answered by using the back button in the case of the task-oriented pages. This is acceptable and normal according to Nielson [NIEL00]. Special care is taken in NetClasses to ensure that the back button does not cause any trouble by resubmitting information that has already been dealt with (see *Section 4.5*). When viewing the notes files the question "which notes have already been read?" is taken care of by the browser colouring those links differently to the unvisited ones.

In the type of page shown in *Fig. 4.3*, the information necessary to answer the third question is available in the jump menu in blue at the top of the page below the banner and in the menu items themselves. NetClasses provides a short description of what each menu item does in a little text box that pops up when the mouse rests over the menu item. When the user is on an intermediate task page, it is desirable that they finish the task rather than jump to anther part of the site. Even though NetClasses discourages jumping out of such pages the design ensures that it is safe to do so.

When a user is reading the notes page or an assignment file there is no jump menu provided. Files are viewed in a new window, so when the user is finished reading the notes file, for example, they can close the window and resume the NetClasses session. That design feature is there because the notes file could contain links to sites outside of NetClasses and it is safer if such visiting is done in a separate window. Another reason for using a separate window is that some browsers open files like Word document files in such a way that they take up the whole window and the temptation is to close the window when finished reading the file. That would end the NetClasses session by mistake. The sites visited from links on a notes file would probably not have the same look and feel as NetClasses pages and sites are sometimes configured so that you cannot get out of them with the back button. *Fig. 4.5* shows a notes file page being opened in a new window.


Back
Search
Favorites
Media
Address
http://157.190.66.232/NetClasses/secure/lecturer/lecturerMenuNotesMenu?NotesMenuIndex=5&NotesMenuLevel=0
Go
NetClasses Course Facility
Com1 Programming Notes.
http://157.190.66.232/NetClasses/lecturers/jdoyle/Com1_/Programming_/notes/ForLoop.rtf - Microsoft Inter...
File
Edit
View
Insert
Format
Tools
Table
Go To
Favorites
Help
http://157.190.66.232/NetClasses/lecturers/jdoyle/Com1_/Programming_/notes/ForLoop.rtf

For loop.

The third and last of the three repetition control structures is the for loop. As can be seen from the flow chart on the right, it follows the same form as a counter controlled while loop.

The first box from the top is initializing the control variable (or counter) and the last box in the loop is refreshing the control variable

Note that the while loop and the do .. while loop can be reduced to one decision and one

Figure 4.5 A notes viewing page in a new window.

4.4 Restricting Access

The login process is a big part of NetClasses. Once the user is logged in, it is important to control their access rights to various pages.

Depending on what role a user has they should be allowed or denied access to various section of the application. A naïve approach to this would be to code the access rules into the pages and servlets that are being requested. A distinct disadvantage to this is that the access code would have to be scattered around the application. The writer would have to be continually checking to see which user is allowed to do what.

The pseudocode for such a denial segment in a page meant for lecturers might look like: -

```
If NOT user.hasrole(Lecturer )
        Forward to access_denied.jsp
```

A more informed approach, that allows separation of this type of barring from the functional logic of the job in hand, is to apply filters [FIEL02]. Filters allow developers to layer new functionality on top of web-based resources. For NetClasses, this means that requests and responses can be intercepted and processed in filters. The filters can decide whether a request is going to be passed on to the intended target or whether some other action (logging in for instance) is more appropriate. The filters, then, can encapsulate the barring activity so that it does not have to be coded into each page or servlet. The filters can be mapped to apply to requests where the web address or Universal Resource Locator (URL) matches a certain pattern.

For Java Server Pages and servlets, mapping filters can be done in the "web.xml" file. This type of mapping means that the entire URL pattern after the context is required. Contexts are defined in "server.xml" in the "conf" folder of the Tomcat

Servlet Container. The context for NetClasses is the part of the web address up to and including NetClasses e.g. http;//…/NetClasses". Trying to control the behaviour of the application with this type of mapping proves to be unworkable because the entire structure of the URL is needed at the time of writing the filter map. If a change in directory structure is applied to the project at some stage, then these mappings become invalid.

In practice, it is easier to have filters mapped to trap all requests and responses rather than to try and list in advance the exact URL patterns that should get caught. In that way, NetClasses can have the rule, for instance, in the student filter that any user who does not have the role "student" will be directed away from any page with "/student/" as part of the request URL.

Besides the student filter, NetClasses needs a lecturer filter, a login filter and a domain filter. The lecturer filter is similar to the student filter. In fact it uses the same code with different parameters.

The login filter is required because there needs to be an area of the site that a user who is not logged in should be allowed to go. One such area is the login page. The filter needs to intercept requests from users that are not logged in and are attempting to access secure areas. In NetClasses there are two areas where the user needs to be logged in for access. These are the secure folder and the lecturers folder.

4.4.1 Domain filter

The domain filter is needed for security reasons. It comes into play when a user who is correctly logged in to one lecturer's area tries to migrate to another lecturer's area. They might attempt this by changing the request URL in their browser. This would only happen if a user were deliberately trying to shortcut the login process that is required if changing from one lecturers page to another. The lecturer's login ID is called the domain in NetClasses. The user is associated with a domain when the course is decided during the login process.

The original approach in NetClasses to this was to have a directory structure that had the lecturer's ID at the head. The users would then enter that directory and access pages from that branch only. This type of layout is shown in *Fig. 4.6.*

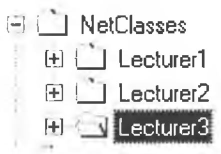


Figure 4.6 Lecturers as branches of tree.

This is workable and easy to code but means that there is duplication of pages. With this method there are pages that are the same for each lecturer where only data depending on the lecturer ID differ. Now the pages that are common to all lecturers are in the secure/lecturer directory. Some servlets and pages are outside the secure area because they are needed before login (see *Fig. 4.2*).

Effecting this change makes it obvious that the domain filter cannot be mapped in the usual way e.g.

```
<filter-mapping>
    <filter-name>domainlecture1</filter-name>
    <url-pattern>/lecturer1/*</url-pattern>
 </filter-mapping>
```

It means rewriting the web.xml file anytime a new lecturer is added and also creating a new filter instance (the same filter – different parameters). The servlet container (Tomcat) needs to be restarted after this modification of the web.xml file so that the new filter gets registered. Although adding a lecturer is a rare event and is only done by an administrator, NetClasses should be started once and left running for ever. Therefore, this method of filtering access to a lecturer's section of the NetClasses site is unsuitable.

In order to make the domain filter work even for domains that don't exist when the web.xml file is written, a different kind of mapping is needed. The mapping needs to be general. The code in the filter must be able to know both the domain the user is attempting to access and the domain the user is logged in under. The login domain is easy to determine at login time and this becomes an attribute of the user object. To get the name of the domain the user is trying to access, the domain filter needs to parse the request URL to extract the domain folder name. If the two are the same the filter passes on the request and response objects to the next filter. If the user attribute, lecturer, is not the same as the domain that the user is attempting to access, then the request and response are passed on to the login page.

The use of filters for processing HTTP requests and responses in NetClasses is versatile. It allows for greater separation of logic and page design than does the coding of the logic governing access to a page in the page itself. The file permissions of the host machine operating system cannot be employed to restrict access. This is because the only user that the operating system is aware of is the NetClasses application no matter who is logged in to that application.

4.5 Flow Control

A three-tier architecture is presented in *Section 4.1*. There are the three layers; Presentation, Processing and Database. The logic of the application is encapsulated in the processing layer. The logic can be further broken down to business logic and control logic.

The business logic looks after processing the data. For a web application, the control logic decides which page to show next. Duane K. Fields et al highlight this separation in their book: - Web Development with Java Server Pages [FIEL02]. They introduce an extra layer (see *Fig. 4.7*) between the presentation layer and the business logic.

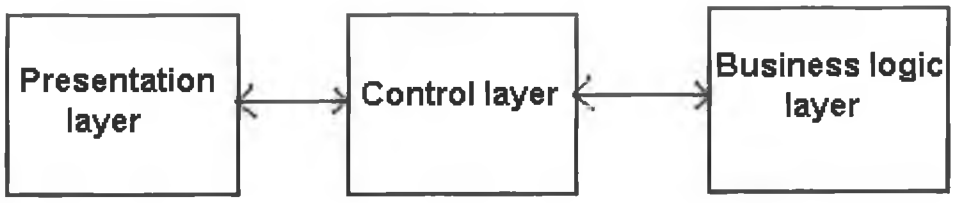


Figure 4.7 Control layer.

The requirement for a control layer becomes apparent when the inherent difference between web-accessed applications and conventional off-line applications is considered.

The Model-View-Controller design pattern [BUSC96] is applicable here. The Presentation layer is the View. The Control layer is the Controller and the Business Logic layer along with the database is the Model. The rest of this chapter is about implementing the Control layer.

4.5.1 Conventional Applications

Conventional applications either have no user choice, in which case there is no flow control problem, or the user effectively chooses from a menu. If the user selects one item from a menu, the flow control manages a return to the menu after that item has been dealt with. Even if the user choice is to go to a sub menu, flow control can manage to return execution to the original menu after exiting the sub menu. The pseudocode for such a standard flow control problem is shown below.

```
Repeat
    DisplayMenu( )
    Read user choice

    Case user choice of
        1:    job1( )
        2:    job2( )
        3:    submenu( )

Until user choice = 4
```

Execution follows a path that is bound by the constraints of the repeat loop. Users select a choice and the process associated with the choice is executed. When the job is finished the user will be presented with the menu again. The cycle of choosing, executing and choosing again continues until exit (4 in the example) is chosen. It works because of the built-in flow control in programming languages [DEIT02]. The user's choice is not read before the menu is displayed in the example. After the user's choice is read, it is examined and a selection is made depending on its value.

4.5.2 Web Accessed Applications

Like conventional off-line applications, web accessed applications can use sequence, selection and repetition to manage flow control. The essential difference is that user input is in the form of HTTP requests and these are handled as soon as they are received. The web server is never waiting for a specific user input as a response to a specific prompt. Instead, the web server is always listening for any HTTP request. Indeed that request can be from any user and the web application has the job of recognising which user is sending the signal.

In the previous example, flow control is facilitated by the fact that execution waits for user input and then continues where it left off in the same thread. With web-accessed applications, the user input is serviced immediately on the server side in a new thread. The thread that sent the web page to the user, asking for input, is not aware of when that input arrives. Instead, a new thread is created to handle the new request independently of the requesting thread. It is clear that the solution expressed in the pseudocode example, cannot work for web–accessed applications. Each HTTP request is unconnected to what went on previously.

There are two strategies to implement flow control in JSP applications; Page-Centric Design and Servlet-Centric Design.

4.5.3 Page-Centric Design

All control decisions about which page to visit next are written into the JSP pages themselves, or accessed in the JSPs through referenced objects.

An example from NetClasses illustrates the point. A menu is presented that gives the choice of creating a new class of students, deleting a class, populating a class, removing a student, adding a subject or removing a subject. This menu is shown in *Fig. 4.8*.

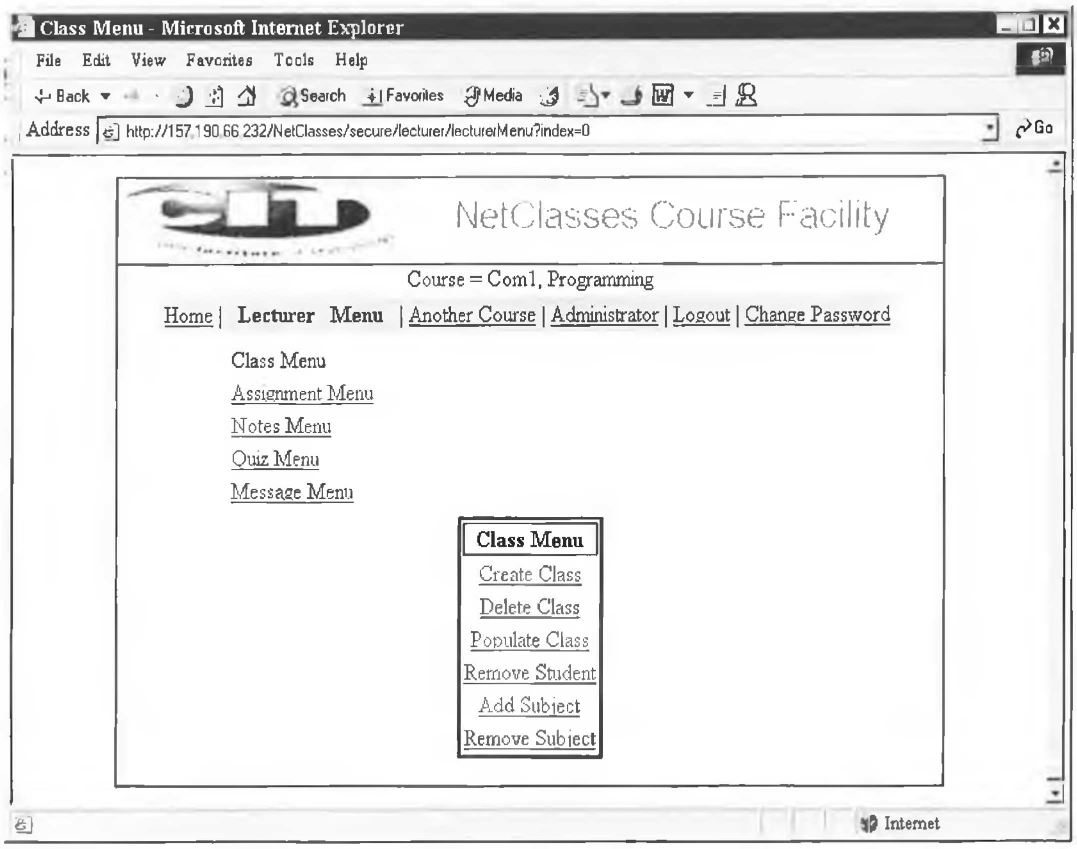

Class Menu - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://157.190.66.232/NetClasses/secure/lecturer/lecturerMenu?index=0
Go
NetClasses Course Facility
Course = Com1, Programming
Home | Lecturer Menu | Another Course | Administrator | Logout | Change Password
Class Menu
Assignment Menu
Notes Menu
Quiz Menu
Message Menu
Class Menu
Create Class
Delete Class
Populate Class
Remove Student
Add Subject
Remove Subject
Internet

Figure 4.8 Class Menu page.

The menu should be continually presented until the user jumps to another part of the site. If the user clicks on Create Class, then a new class should be created and the menu presented again. Similarly with Delete Class and Populate Class and the others, the menu should be presented again when these jobs are done.

With page-centric design [FIEL02], a menu item would typically be a link to the JSP page that looks after that job. The individual pages would then direct the user back to see the menu page again. This is quite easy to control with each page directing where flow should go next.

Things get a little more complicated when jobs require more than one page. An example is the Create Class menu item. First of all the user should be asked for a name of the class to create. This means another intermediate web page is needed to ask the question and read the answer. The answer has to be compared to class names that are already there. If the class name exists already, then the user must get an error page and then the menu should be presented again. If the new class name is valid, the user could be informed with an intermediate page. If the user wants to proceed, the class should be created, and the user informed with another intermediate page, that everything went according to plan. Finally the menu is presented again. The user should also be given a chance to cancel instead of providing input if that is what the user wants.

Still this is no more difficult than, and is indeed analogous to, providing structured programming without repeat structures. It is like using combinations of if...goto to provide the functionality of repeat…until. The goto is like the part of the JSP page that redirects to the next page.

The real difficulty comes from the fact that users can use the back and forward button to access any of the intermediate pages. Without careful programming, this can lead to a breakdown in more complicated applications.

“Since each segment of a page-centric JSP application is its own page represented by its own URL, there is really nothing to stop a user from executing the pages out of order. Each page of your application must check for valid request parameters, verify open connections, watch for changing conditions, and generally take an assume-nothing approach with regard to the order of operations of your pages. As you can imagine, this quickly becomes unmanageable for all but the simplest applications.” [FIEL02].

4.5.4 Servlet-Centric Design

With Servlet-Centric design [FIEL02], the flow control logic is encapsulated in servlets (the application objects, that extend HttpServlet, and so have access to HttpServletRequest and HttpServletResponse).

The servlets then redirect the requests to the required pages. Server side redirecting with JSP does not create a new HTTP request but keeps the old one along with all its parameters. Information can also be added to the request in the form of Parameters or Attributes (Parameters for strings and Attributes for Java objects). This information can be used in the JSP page, for instance the information could contain the address of the next page or servlet to activate.

Where there are intermediate pages that the user should not be able to activate by refreshing the web browser, then some system of validating the HTTP request is needed. One such system of validating http requests centres on a use-once token.

4.5.4.1 Use-Once token

Use-once tokens are a servlet-centric solution to the problem of users refreshing pages [FIEL02]. Each page that raises a request by user action will be serviced only once. Any attempt to refresh the page or to resend the request will fail because the token will not be current.

The lifecycle of the HttpSession object and the HttpRequest object are different. A session object is created when a user first access the application and dies when the browser is closed or the connection lost. The request object is created any time the user requests a page or submits a form and dies when the request is serviced. Typically any request is finished being serviced when a new page is presented to the user.

The use-once token is attached as an attribute to both the request and session objects. Commands are only executed if the token is the same in both the session and request. Each time a new request is raised, the current token is changed. When a user resubmits a form that has already been serviced, the token attached to that request will be stale. The token attached to the request object for a page is sent as part of the page after being put there by server-side JSP scripting. So the token becomes a hidden part of the page the first time it is requested and if it is refreshed, the same token is sent back with the new request.

The use-once token does not put a flow control regime in place, it is merely a devise to prevent users bypassing whatever flow control is in effect.

4.5.4.2 Program Flow By Numbers

The NetClasses solution to the flow control problem is presented in this section. Each menu item is assigned an index number. If a command has many parts, as with the Create Class example in *Section 4.5.3*, each part is assigned a level number.

The HTTP request object is made to carry three special parameters with this system. Continuing with the example they are:

1. ClassMenuIndex.
2. ClassMenuLevel.
3. status.

All requests to do with servicing the Class Menu are directed to the lecturerMenuClassMenu servlet, *Appendix 2*. When writing the JSP pages, there is no need to be aware of any flow control issues as all forms and pages request the same servlet. Using a servlet in this way is normal for a servlet-centric approach [FIEL02].

The servlet directs flow to a command and passes the HttpRequest and HttpResponse to that command. Each level of command has its own Java code. The command forwards the request to the JSP page to create the user view. This is also usual for servlet-centric solutions.

The NetClasses control servlets read the request parameters index and level and decides whether the request comes from a menu command or an intermediate page. Menu levels have the value 0. If the request comes from a menu command, then the session index attribute is set accordingly. The level is raised by one in either case because the next part of a multi-part command will have a level number, one greater that the previous part. The session attribute, level, is then set. The session attribute, index, remains the same until a new choice is selected from the menu. Both request attributes (index and level) are set as well. The commands are stored in arrays and the index and level determine which command is called next.

The command knows which level and index it is at in this method because it is set at initialization. It then checks its own level and index against the session level and index. If they are the same, execution will go ahead and the appropriate JSP page is called, otherwise control is handed back to the menu page.

With the NetClasses method, the request attributes, index and level, are used to control flow and the JSP writer does not need to know at what level and index the page fits in the system. In fact, the same JSP page can be used in two different places. The JSP page has the request level and index embedded into it as part of the dynamic content and merely passes those same values back to the server when the user submits the request.

The third parameter, status, is set to either “ok” or “cancel” in the JSP pages and depending on which value it has, the command code decides whether to go ahead or not.

The code for LecturerMenuClassMenuServlet.java is given in *Appendix 2* as is the code for LecturerCreateClassCommand_2.java.

This system is easy to operate. It solves the flow control problem by always flowing back to the handling servlet after every command is executed, except in the case of cancel or error. In these special cases, it is the command execute( ) method that decides which page to show next. When writing the JSP pages, no attention is given to flow control. When writing the controlling servlet, the flow control is determined by the index and the level. Of course the servlet has to be initialized so that arrays with the correct commands at the correct indices exist. When writing the command execution methods, only the exceptional cases need to be considered. These will return control to an error page or back to the menu page.

The refresh problem is also solved by this method of flow control. A second advantage is that the menu level values can be manipulated to break the regular flow if needed. One example of this is the case of a user trying to create an assignment that is already there. The expected error page comes up, but then, instead of going back to the menu page, flow is directed back to the page where the error was made so that it can be corrected.

4.5.5 Java Struts And Flow Control

Struts is from the Jakarta Project (freeware suppliers and developers of Tomcat Servlet container). It is a development framework for Java servlet applications based upon the Model-View-Controller (MVC) design paradigm [SPIE03].

The flow control in a struts project is managed by a controller servlet and configuration file combination. The configuration file can be changed, so it is a reusable solution.

As the NetClasses flow control regime solves the problem and it could be made configurable by the use of initialization parameters in the controller servlet, the application does not to use the Struts framework. NetClasses implements the MVC design patern.

4.6 Conclusion

This chapter addresses the problems of site access, flow control and accessing pages out of order.

Site access is controlled in a sophisticated manner by using filters. Both the flow control and refresh problems are solved by an original solution developed for this thesis. NetClasses is a secure and robust application as a result of these solutions being implemented.

Chapter 5 presents the NetClasses components.

Chapter 5

Components

A calendar that is used in the assignment component is taken from freeware code. The basic login procedure is taken from [FIEL02] but is heavily modified to cater for requirements of this particular project. All other code is written from scratch.

5.1 Login

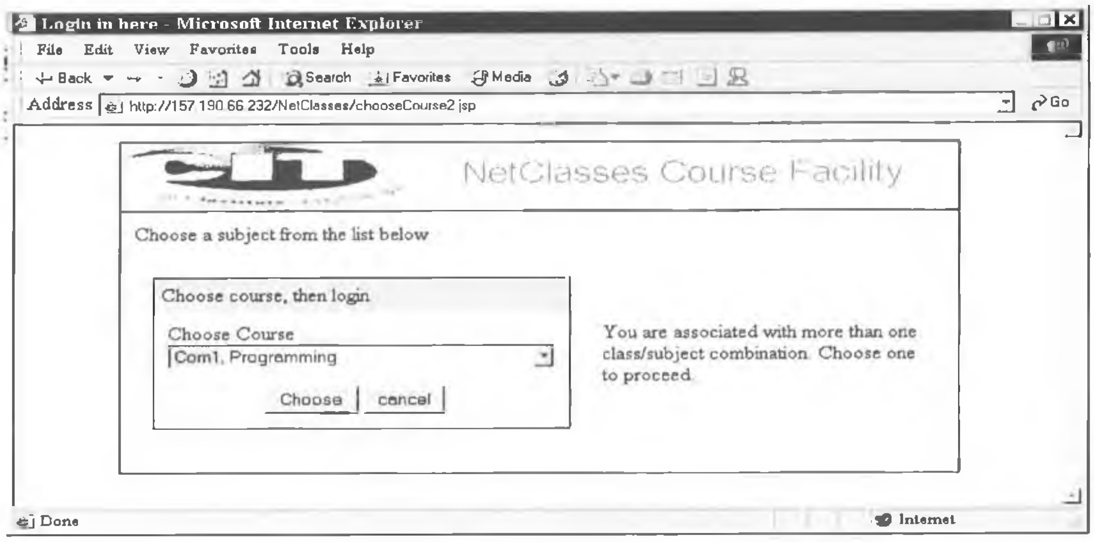
As outlined in *Section 2.2.2*, all users have the same logon page. Users do not have to indicate which role they have in the system.

The minimum amount of information that NetClasses needs to logon a student is the student's username, password and course. A course is identifiable by the lecturer, class and subject. The user must choose from the list of courses if they are associated with more than one course.

If a student tries to login when they are not in a class then they are refused. Lecturers and administrators are allowed to login when they have no course to select.

The code for the login page and servlet were developed from a standard login page in [FIEL02].

Fig. 5.1 shows a page asking the lecturer to choose between two courses taught by that lecturer.


Login in here - Microsoft Internet Explorer

Address http://157.190.66.232/NetClasses/chooseCourse2.jsp

NetClasses Course Facility

Choose a subject from the list below

Choose course, then login

Choose Course: Com1, Programming

Choose | cancel

You are associated with more than one class/subject combination. Choose one to proceed.

Figure 5.1 Choose course page.

5.2 Lecturers

5.2.1 Notes

One of the most interesting parts of the project is the implementation of the file uploading module for lecturers to post their notes on the site and for students to upload their assignment work.

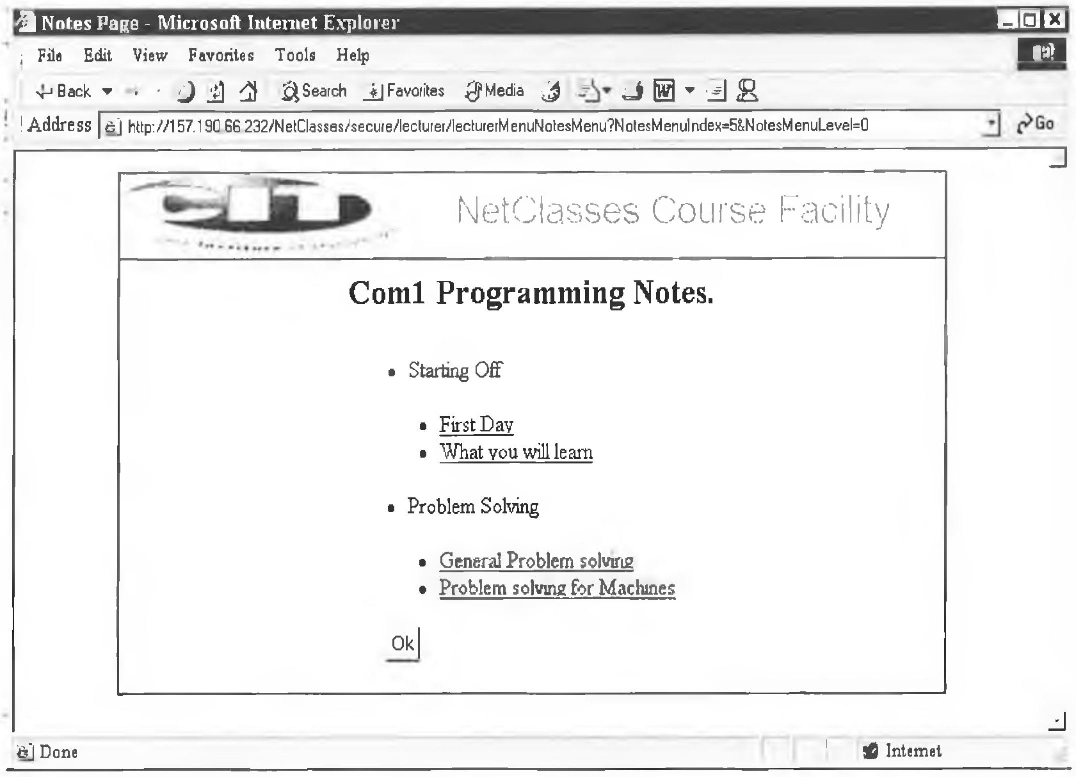
Java's Remote Method Interface (RMI) was at first considered to implement this part of the project. In the end a commonly used method of uploading files from forms on web pages using

```
enctype="MULTIPART/FORM-DATA"
```

as part of the form descriptor was employed. A Java RMI solution has the disadvantage that some of the application code has to be on the client machine. An advantage of an RMI solution is that whole folders can be uploaded instead of just single files. Although little enough is written about MULTIPART/FORM-DATA (RFC2388) in Java Server Pages textbooks, it is the normal way to upload files using HTTP. [FIEL02] has no reference to this at all. [YANK03] has a PHP example. There are a number of commercial JSP upload servlets on the web and a small number of free offerings as well. A few of the free ones were examined but it was found to be easier to develop one from scratch. The NetClasses uploader is a java class that is not a servlet (does not have access to HttpRequest and Http Response) and that class is used in servlets when required. The solution is general and can be used when the file uploading button is anywhere in the web page form. It is called UploadServerBean.java and is given in *Appendix 2.3*.

NetClasses has a web page that gives the students access to the uploaded files. Many lecturers have their own web pages with access to their notes. Indeed this project was conceived partly so that it would be easier for a lecturer without good IT skills to create web pages for notes. NetClasses produces a notes page that looks like one that might be produced by a course lecturer to make their notes available to students.

The NetClasses interface is a web page where all the files of notes are grouped under various headings. The page (*Fig. 5.2*) looks like a context page of a book and the headings are like chapter names. The various clickable file links below the headings are like sections in a chapter. When the student clicks on any of the sections the relevant file opens so that that section can be read.


Notes Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://157.190.66.232/NetClasses/secure/lecturer/lecturerMenuNotesMenu?NotesMenuIndex=5&NotesMenuLevel=0
Go
NetClasses Course Facility
Done
Internet

Com1 Programming Notes.

- Starting Off
  - First Day
  - What you will learn
- Problem Solving
  - General Problem solving
  - Problem solving for Machines

Ok

Figure 5.2 The notes viewing page

The notes page works by a system of unique numbers that belong to the headings and links and that identify where they should all appear in the list. The numbers are called parent and serial. Each link belongs to a heading and each heading has a serial number. The parent of a link is the heading that the link belongs to. The headings are made to have parent of 0. When the names of all the headings and links are retrieved from the database, the parent and serial numbers can be used to decide where they go on the page. For example, the link ‘General Problem

Solving' in *Fig. 5.2* will have a parent of 2 and a serial of 1. No other item on the page will have the same parent and serial numbers.

In order for lecturers to be able to put the links in the correct place on the page, there are similar interfaces provided for them to click on the page item (header or link) below which their new link will appear.

Deleting links and heading is also provided for and the lecturer is not allowed to delete a heading unless there are no links still attached. This rule is there because the links need to have a heading and would attach themselves to the previous heading if their own heading was deleted.

A disadvantage of the NetClasses system of ordering the links on the page is that every link has to be under a heading and so no link can stand alone.

Links to quizzes can be put on the notes page in the same way as links to files are. This is because it makes sense to take a quiz after reading notes or to take a quiz about one section before moving on to the next section.

All link items on a notes page can be made visible or hidden. Among the attributes of a page item stored in the database is 'visible' and it takes the value 1 or 0. Items with a visible of 0 are not shown to the student.

5.2.2 Assignments

Submission of assignments has always been a thorny problem in CIT and in other colleges as well [JONE97]. Traditionally students used to print out their assignments and hand the scripts to the lecturer or put them in an assignment post box before a certain date. A number of lecturers still work this way. It seems hardly appropriate for computing courses where the assignment work is an application or presentation that is meant to run on computer anyway. Where the paper version will not do, there are a number of alternatives.

The student can hand disks up. This is better than printouts for material that need to be checked on a computer but there is the problem of corrupted disks and it is not unknown for disks to get lost. Floppy disks seem to be particularly prone to corruption. Often the material on the disk needs to be copied to another place to be used. There is no automatic recording of the receipt of the assignment material.

A common method in CIT is for the student to email the assignments. This has the drawback that the students use a variety of different email providers and the mail, or its attachment, does not always get to the lecturer. There is also the slight inconvenience of the lecturer having to save the attachment in a suitable folder and to acknowledge receipt of the assignment.

Another method is for the college to provide a folder with the appropriate access rights where the student can copy their files over from another college machine and the lecturer can collect them. The biggest drawback with this system is that the student must be in college when transferring the files. It also requires the cooperation of a third party to set up the repository folder correctly.

A fourth method is the use of File Transfer Protocol (FTP) to upload the files from any machine. This allows the student to transfer files from their home machines as well as from college. It presupposes a certain amount of computer expertise on the part of the students and not every college is happy about providing FTP sites for this purpose.

Uploading files to a web site like NetClasses does not give rise to any problems like those listed above. Permission to upload does not require any programming by a third party. The lecturer does not have to acknowledge or save files. There is very little computing skill needed to use the facility and it can be done from anywhere.

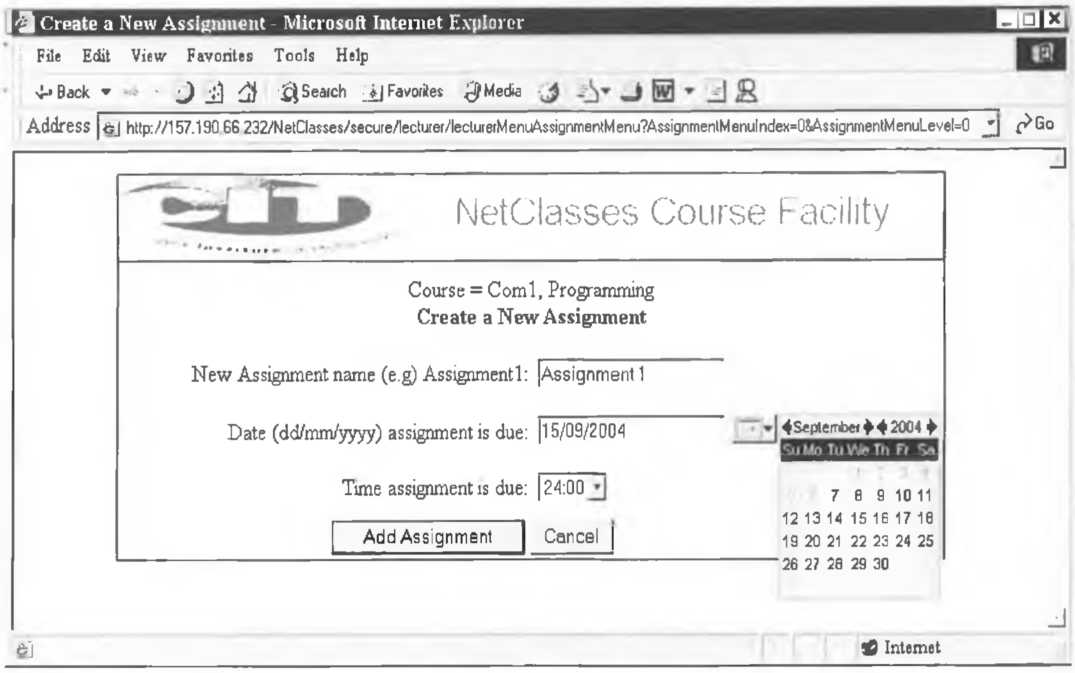
The files are all stored in appropriate folders on the web server machine. In order to separate the files of one student from those of another, NetClasses creates suitable folders. There is one folder for each assignment created by the lecturer. In that folder will be a sub-folder for each student who submits work for that assignment. The code of NetClasses is easily available and the file structure is deducible from the code. A clever student who wanted to cheat could point their browser at the submitted file of another student after working out where that file is stored. To prevent such behaviour NetClasses has a domain filter that deflects any such requests, see *Section 4.4.1* Each student who is logged-in has access to any notes files or assignment description files that are in the folder associated with the course they are logged-in to. The subfolders where the submitted assignment files are stored are only accessible by the lecturer and by the student for whom the particular subfolder is created.

When a lecturer is creating an assignment, one of the boxes to fill in is for the date the assignment is due. The lecturer clicks on a calendar logo and a clickable calendar appears. The code for this calendar was found on the web. It is freeware and credit is given to the writer in the code. It seems only fair to include a credit to the author here as well:-

Author : Lea Smart

Source : www.totallysmartit.com.

Fig. 5.3 shows the page for creating a new assignment.


Create a New Assignment - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://157.190.66.232/NetClasses/secure/lecturer/lecturerMenuAssignmentMenu?AssignmentMenuIndex=0&AssignmentMenuLevel=0
Go
NetClasses Course Facility
Add Assignment
Cancel
September 2004
Internet

Course = Com1, Programming
Create a New Assignment

New Assignment name (e.g) Assignment1:	Assignment 1
Date (dd/mm/yyyy) assignment is due:	15/09/2004
Time assignment is due:	24:00

Figure 5.3 Creating assignment page.

5.2.3 Class

The concept of a group of students forming a class is central to NetClasses. It is how NetClasses is structured. Classes have subjects and lecturers teach subjects to classes. Students log in to a lecturer, class and subject. The NetClasses method of managing a class is designed around the idea that the classes should be entirely managed by the lecturer, see *Section 2.2.4*. The lecturer using this application does not need to have permission from anybody to create, populate or delete classes.

To enable the class information to be managed, there are some menu choices available to the lecturer in Class Menu. These are discussed in the sections below.

5.2.3.1 Create And Delete Class

When a class is added to the system, a sub folder in the lecturer's folder is also created. This folder is intended to hold all the notes and assignment material. The lecturer can choose any class name except a name of a class that particular lecturer already has in the system. Windows does not allow any folder to be named Com1, Com2 or Com3. It would seem that this operating system cannot understand the difference between Com1 folder and Com1 communications port. To get around this bug NetClasses makes the folder name associated with the class to have the same name as the class plus '_'. For example Com1 has a folder Com1_. This came to light because there is a class is called Com1 within CIT. For safety sake the same thing is done for any folder created by NetClasses except the folder with the same name as the lecturer. That folder is created when a lecturer is added to the system. Two lecturers can have the same class name describing two different classes in the system. The two classes may be the same in real life but NetClasses treats them as two different groupings.

Deleting a class means that the class name will disappear from the database but also that all folders, quizzes, notes, assignments and messages associated with that class will go. Students remain in the system in case they belong to another course.

5.2.3.2 Populate Class

The web page that is presented to the lecturer when choosing to populate a class contains a lot of JavaScript. The script describes client side processing that is done on the list of names and passwords that the user enters.

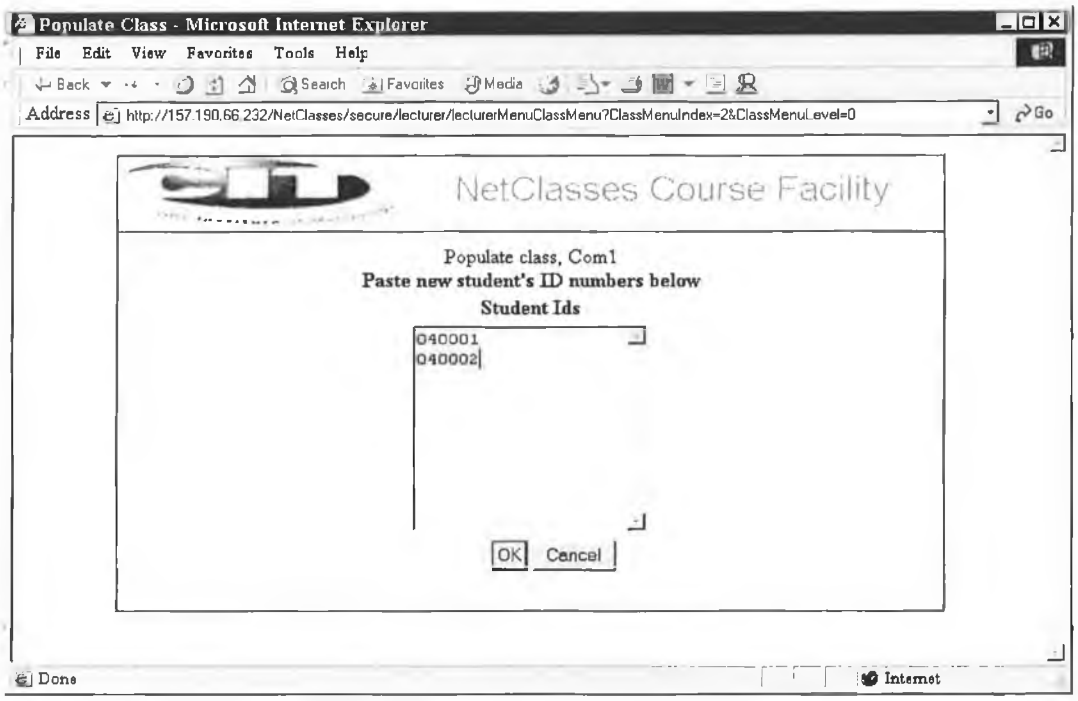
Because the input of a long list of names is precisely the kind of work that is supposed to be made easier by computers, this interface in NetClasses is made so that the names can be pasted from another file. In CIT the class lists become available before the students register. This list is usually heavily edited to reflect the actual number registered after registration day. Editing can take a few days and the updated class may not be published until well after the students are in class. It may be peculiar to CIT, but some change in the student population always occurs during the first few weeks of term. If the class lists are available online the lecturer can copy and paste columns from there to the Populate Class page in NetClasses.

The code that implements the populating of the class is interesting in that a lot of the input checking is done on the client side to make the job of the server easier. The main piece of checking that is needed is to make sure that all student usernames are unique within the class. This means checking the usernames against the rest of the students that are being added and against the students that are already in the class. The last piece of processing is an example of distributed processing. NetClasses is essentially an application where the processing is done on the web server and the client computer is used to show the information or accept user responses. In this case there is a bit more than that done on the client side.

Section 4.1 refers to the fact that no NetClasses logic code is kept on the client machines. In cases like this the code needed is incorporated in the web page as JavaScript. All DOM browsers understand JavaScript in the same way so we can have this kind of simple processing that does not write to the database, on the client side. This saves the number of HTML requests and hence the number of pages to download from the server.

The student details are later checked against all users already in the system, but that is done on the server side. No student in a class can have the same username as the lecturer either. It is also necessary to check that each student has a first name, last name, username and password. Checking that all students have each field filled is done by counting the number of entries in each column, see *Fig. 5.5*.

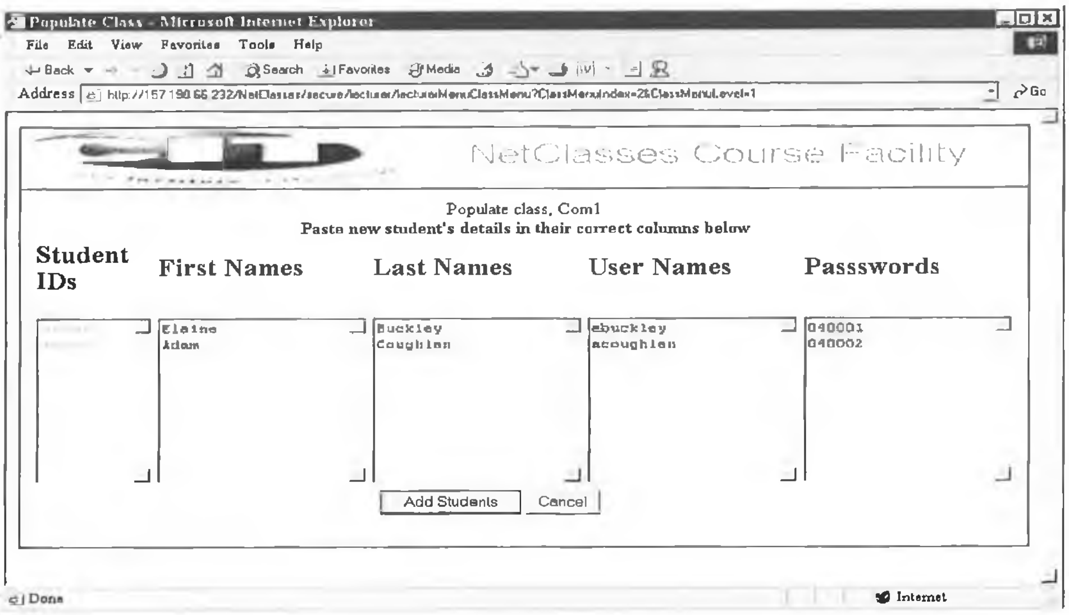
When adding students to a class, NetClasses should be aware if the students are already in the system. This is managed by having a unique student ID stored in the database for each student. The lecturer is asked to enter the student IDs of the students to be entered in the class, see *Fig. 5.4*. NetClasses then looks in the list of existing students to see if they already exist. If they do, they are added to the class without further input from the lecturer. If they are not in the system then the lecturer is asked to enter the details. The job of checking whether students exist already is done on the server side because of the potentially large number of students involved. Even though the client side processing checks that usernames are unique among students being entered, further checking is required on the server side to check that usernames are unique in the system. If the usernames are not unique in the system NetClasses automatically adds a number on to the username to make it unique. The lecturer is made aware of the change by a popup message.


Populate Class - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://157.190.66.232/NetClasses/secure/lecturer/lecturerMenuClassMenu?ClassMenuIndex=2&ClassMenuLevel=0
Go
NetClasses Course Facility
OK
Cancel
Done
Internet

Populate class, Com1
Paste new student's ID numbers below
Student Ids

040001
040002

Figure 5.4 Entering Student ID page.


Populate Class - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://157.190.66.232/NetClasses/secure/lecturer/lecturerMenuClassMenu?ClassMenuIndex=2&ClassMenuLevel=1
Go
NetClasses Course Facility

Populate class, Com1
Paste new student's details in their correct columns below

Student IDs	First Names	Last Names	User Names	Passswords
	Elaine	Buckley	ebuckley	040001
	Adam	Coughlan	acoughlan	040002

Add Students
Cancel
Done
Internet

Figure 5.5 Entering Student Details page.

As well as populating a class NetClasses provides the facility to modify a student's details. Administrators only are allowed do this. Only the first name, last name and password can be changed. NetClasses does not allow alteration of a username because the student might already have folders stored within the system under the old username. Another reason is that the uniqueness of the new username would have to be checked again.

Because students may leave the course for one reason or another, there is the facility in NetClasses to remove a student. The student reference is taken from the class and the student's folders are removed but the details remain in the system.

5.2.3.3 Add And Delete Subject

Lecturers can teach more than one subject to a class and lecturers sometimes change courses. A method of removing a subject from the system is needed and similarly a method to add an extra subject to a class is required. Because of the design of the user interaction with NetClasses, the user is always considered to be logged in to a lecturer, class and subject. If a subject is added then that subject becomes the current one i.e. the user is considered to be logged in to that new subject.

When the lecturer chooses to delete a subject, NetClasses assumes that it is the current subject that is being deleted. That is to avoid the often unnecessary step of choosing from a list of subjects. If it is not the current subject that is to be deleted then the user must go to Another Course on the top menu to make a different subject current. A slight difficulty arises when the subject is deleted: which subject will now be current? The lecturer is not considered to be accessing any course until another one is chosen.

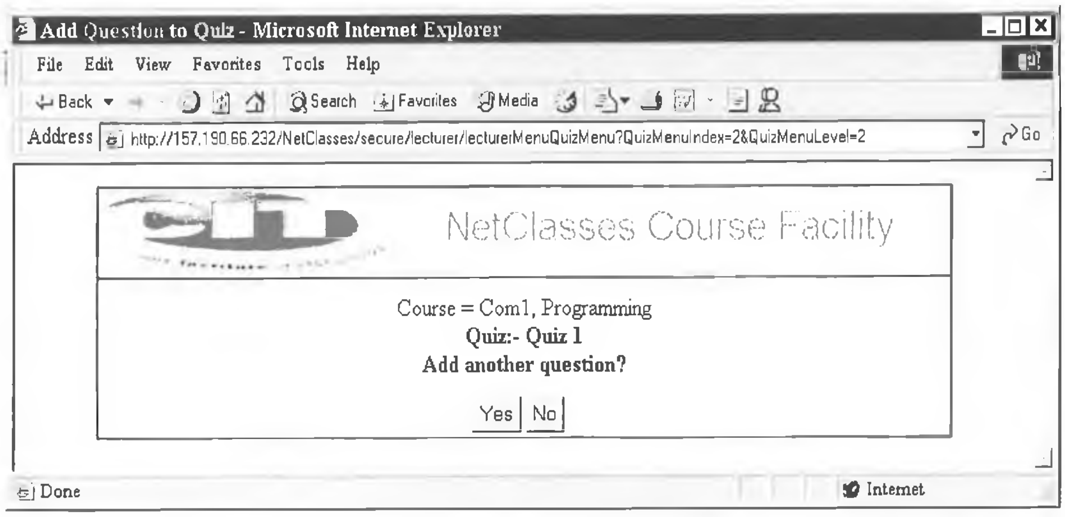
Every Class must have a subject in NetClasses, otherwise there is no reason for the class to be in the system. For this reason it is not possible to delete the only subject in a class (unless the class is being removed from the system). If the lecturer attempts to remove the only subject for a class then an error screen appears and the deletion is cancelled.

5.2.4 Quiz

Lecturers need to create quizzes and add questions to them. In the quiz section NetClasses' flow control system is exercised and it implements the following algorithm;

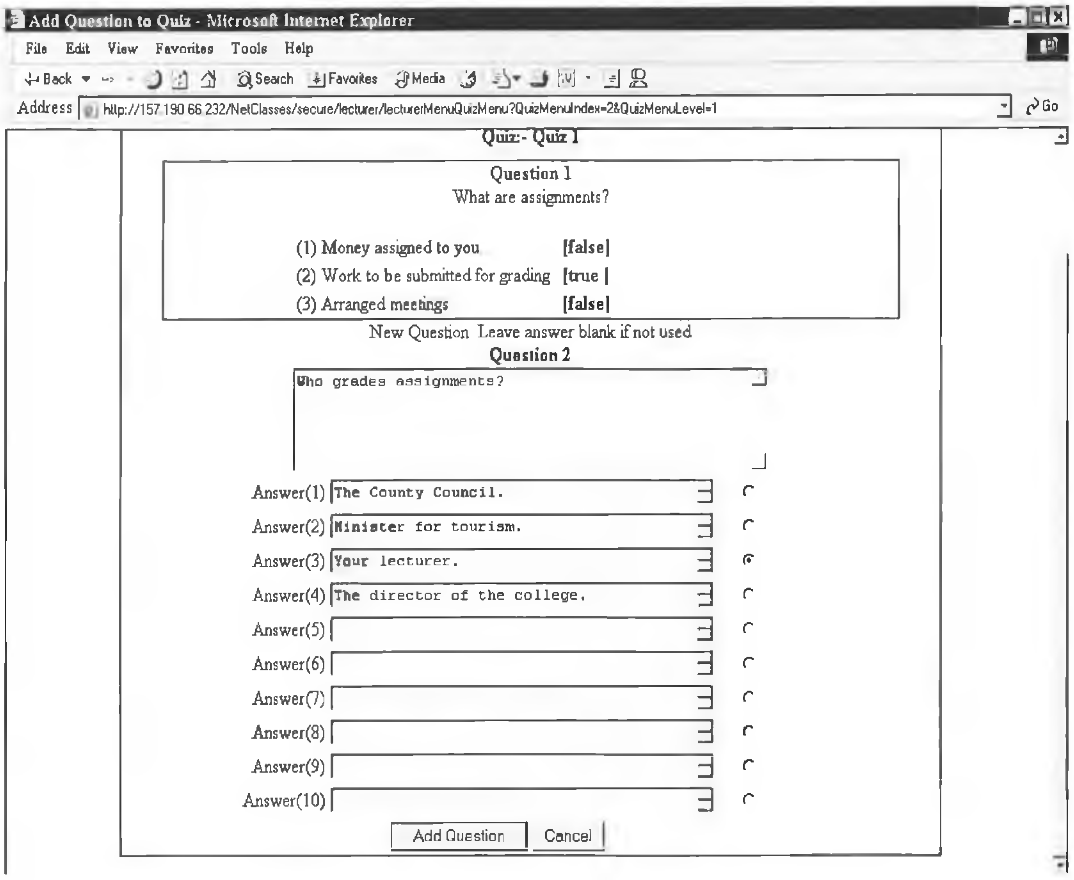
```
Create Quiz
Repeat
        Add Question and Answers for that Question
Until No More Questions is selected.
```

The system of index, level and status attributes, *Section 4.5.4.2*, works well to put the design into effect. Implementing the repeat…until section required two pages. One page is to enter the question and multiple-choice answers along with the correct answer, see *Fig. 5.7*. The other page is to ask the user whether there are more question or not see *Fig. 5.6*. If there are more questions then the level attribute is decreased by one so that flow goes back to the previous task (adding a question) instead of finishing. If the user migrates back to the 'add question' page and submits a question that has already been submitted or cancelled, NetClasses catches it and diverts the request to the quiz menu.


Add Question to Quiz - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://157.190.66.232/NetClasses/secure/lecturer/lecturerMenuQuizMenu?QuizMenuIndex=2&QuizMenuLevel=2
Go
NetClasses Course Facility
Yes
No
Done
Internet

Course = Com1, Programming
Quiz:- Quiz 1
Add another question?

Figure 5.6 Page asking if another question is to be added.


Add Question to Quiz - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://157.190.66.232/NetClasses/secure/lecturer/lecturerMenuQuizMenu?QuizMenuIndex=2&QuizMenuLevel=1
Go
Add Question
Cancel

Quiz:- Quiz 1

Question 1
What are assignments?

(1) Money assigned to you	[false]
(2) Work to be submitted for grading	[true]
(3) Arranged meetings	[false]

New Question Leave answer blank if not used
Question 2

Who grades assignments?

Answer(1)	The County Council.	○
Answer(2)	Minister for tourism.	○
Answer(3)	Your lecturer.	●
Answer(4)	The director of the college.	○
Answer(5)		○
Answer(6)		○
Answer(7)		○
Answer(8)		○
Answer(9)		○
Answer(10)		○

Figure 5.7 Page for inputting questions and answers.

Questions can be added later to a quiz by clicking Add Question on the Quiz Menu. It is also possible to delete questions and edit them.

A quiz can be taken either from the student menu or from a quiz link on the notes page. See *Section 5.3.3*.

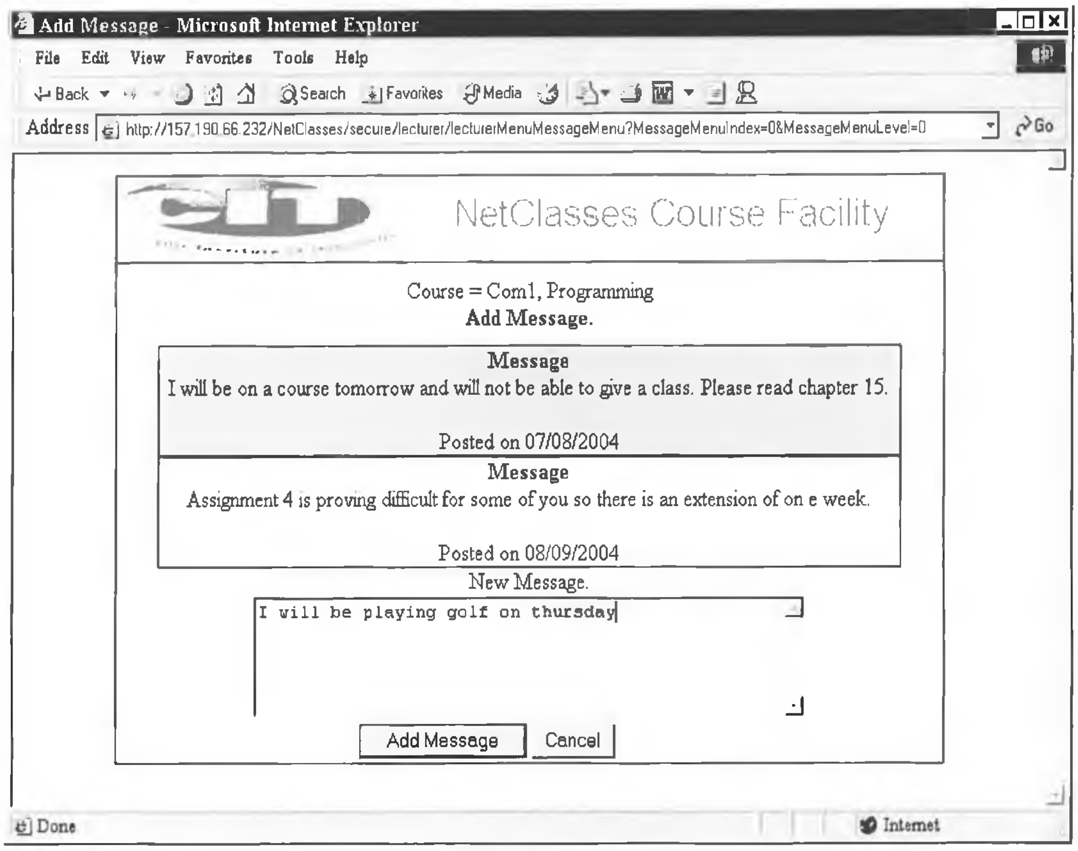
Quizzes are created in either activated or hidden mode. A hidden quiz cannot be accessed by the student from the student menu. Quiz links can also be visible or hidden. A hidden quiz link will not be seen and so the quiz cannot be taken by the student. When a quiz link is made visible then the quiz itself becomes active and can be taken either from the notes page or from the student menu. Similarly when a quiz link is hidden the quiz becomes hidden and is not available from the student menu either.

Quizzes can only be changed from hidden to active by changing the visibility of the link.

5.2.5 Messages

This is the simplest module in NetClasses. When a lecturer chooses to post a message it is collected in a text box and stored with a time stamp in the database. When a lecturer chooses to delete a message it is removed from the database. There are no consequences for any other module.

When adding a message, the lecturer sees a list of the previous messages. *Fig. 5.8* shows a message being added.


Add Message - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://157.190.66.232/NetClasses/secure/lecturer/lecturerMenuMessageMenu?MessageMenuIndex=0&MessageMenuLevel=0
Go
NetClasses Course Facility
Add Message
Cancel
Done
Internet

Course = Com1, Programming
Add Message.

Message
I will be on a course tomorrow and will not be able to give a class. Please read chapter 15. Posted on 07/08/2004
Message
Assignment 4 is proving difficult for some of you so there is an extension of on e week. Posted on 08/09/2004

New Message.

```
I will be playing golf on thursday
```

Figure 5.8 Adding a message page.

5.3 Students

The student menu is less complex than the lecturers menu because the student only accesses the system to read notes messages and assignment, to take quizzes and to submit assignments.

5.3.1 Notes

In *Fig. 5.2*, the notes viewing page. If the student clicks on a link on the notes page a new browser window will open with that file in it. This interface is provided for the student to make navigating through the notes a simple task where the architecture of the NetClasses site need not be remembered. If the user chooses to click on a link on the notes page and then, after reading the resulting page, closes the window they will arrive back at the notes page. Pressing "ok" will return the user to the Student Menu.

5.3.2 Assignments

Students can read the assignment instructions in much the same way that they can read the notes. Any link from the Assignment page is on a new window so the session will not be inadvertently closed.

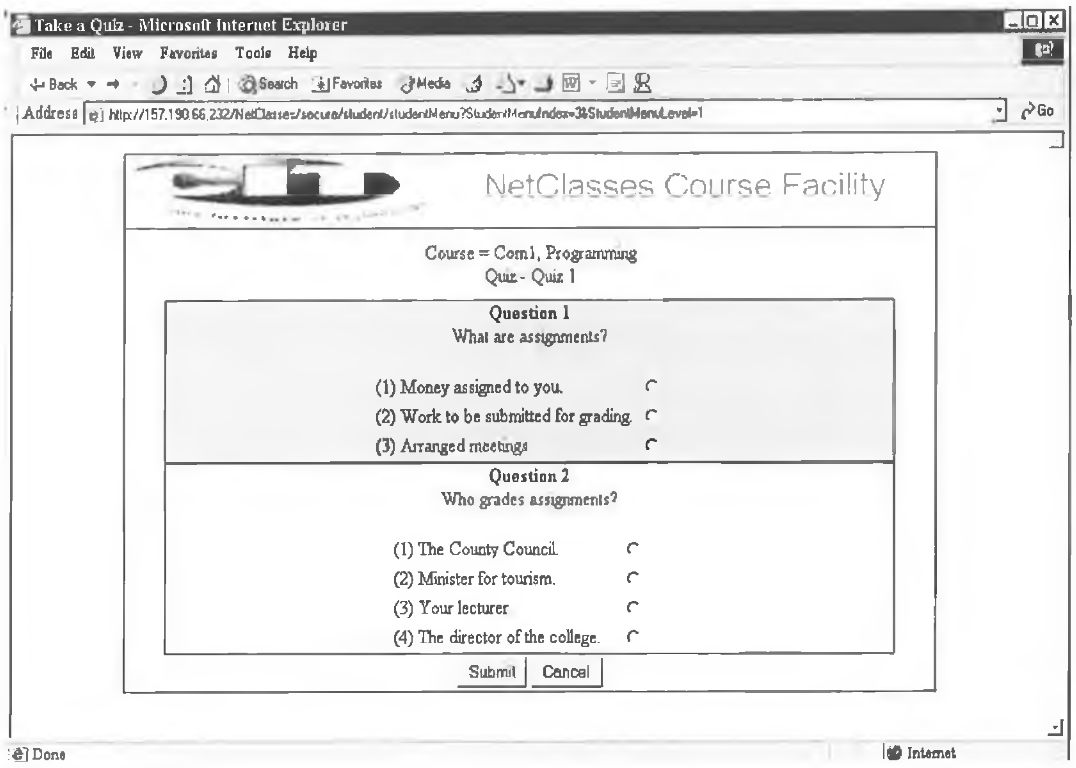
Uploading of assignment files is described in the lecturers *Section 5.2.2*. When a student is uploading, the time of attempted upload is compared to the deadline stored in the database. If the assignment is still on time then uploading is allowed. Students can overwrite their files before the deadline and students can upload more than one file to the server.

5.3.3 Quiz

Students can take quizzes from two different places in the site. One place is from an option on their menu and the other is from a link on the notes page.

Quizzes can be accessed from both places, partly to prove the flexibility of the NetClasses design and partly because the lecturer may prefer not to have the quiz linked from the notes page. There is a slight difference between the two methods of taking the quiz. Accessing the quiz from the menu requires the student to first choose from the list of quizzes and then take the chosen quiz. Accessing the quiz from a link on the notes page means that the quiz is already chosen so control passes to the part of the system that presents and grades the quiz.

Either way, the student will be presented with the quiz as in *Fig. 5.9*. Control must pass back to the student menu in one case and the notes page in the other case. This is implemented by passing a parameter called 'source' to the takeQuiz servlet. Control reverts back to the source when the quiz is finished.


Take a Quiz - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://157.190.66.232/NetClasses/secure/student/studentMenu?StudentMenuIndex=3&StudentMenuLevel=1
Go
Done
Internet

NetClasses Course Facility

Course = Com1, Programming
Quiz - Quiz 1

Question 1
What are assignments?

(1) Money assigned to you.
(2) Work to be submitted for grading.
(3) Arranged meetings

Question 2
Who grades assignments?

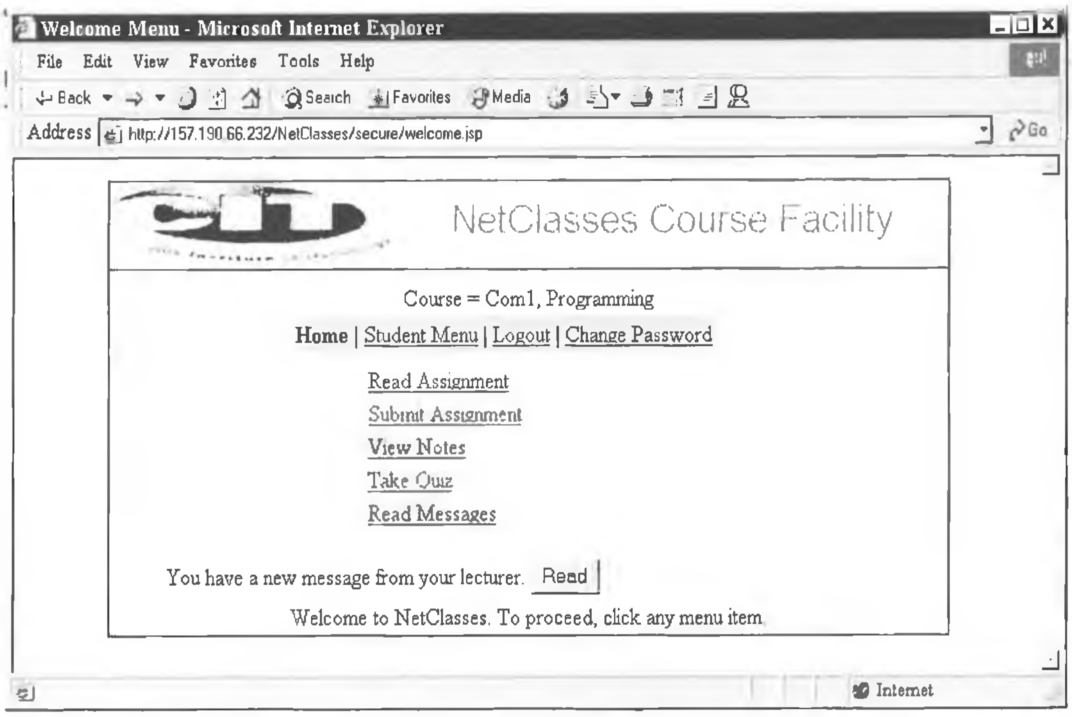
(1) The County Council.
(2) Minister for tourism.
(3) Your lecturer
(4) The director of the college.

Submit Cancel

Figure 5.9 Student quiz page.

5.3.4 Messages

When a student reads a message, that fact is recorded in the database. When a student logs in, NetClasses checks to see if there is a message in the database that has not been read by that student. If there is, an alert message is put on the welcome screen. *Fig. 5.10* shows the welcome screen where there are unread messages.


Welcome Menu - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://157.190.66.232/NetClasses/secure/welcome.jsp
Go
Internet

NetClasses Course Facility

Course = Com1, Programming

Home | Student Menu | Logout | Change Password

Read Assignment
Submit Assignment
View Notes
Take Quiz
Read Messages

You have a new message from your lecturer. Read

Welcome to NetClasses. To proceed, click any menu item.

Fig 5.10 Welcome page with new message.

5.4 Administrators

Administrators have the dual role of administrator and lecturer. In NetClasses it is the job of the administrator to create, delete and edit lecturers. Editing a lecturer can involve changing the first name, last name, password and role. Administrators can add lecturers who have the role of administrator. NetClasses differentiates between the main administrator who is programmed into NetClasses and the administrators who are created by the main administrator. If an administrator is not the main administrator then he may only add, delete and edit lecturers who are not administrators. Administrators may only edit and delete lecturers that they created themselves. When the main administrator is editing a lecturer the choice of role that can be entered is limited to two (lecturer and administrator) by using a selection box.

5.5 Conclusion

This chapter examines various components of NetClasses. The workings of some of the components are explained. An account of how some implementation is facilitated by the flexibility of the flow control mechanism built into NetClasses is given.

Chapter 6 examines the success of NetClasses in meeting the requirements.

Chapter 6

Summary And Conclusions

It is possible to produce a usable web based course material facility as described in *Section 1.1* with good site access control using JSP and free middleware. There were no ready-made solutions available that were free of charge.

We have produced one and learned a lot about client side and server side web based technologies along the way.

A thorough test of NetClasses is outside the scope of this thesis. It is a multi user client server application and a meaningful test would require a team of testers checking for volume of traffic capabilities. The functionality of each menu has been tested while the server side was running on both Windows 2000 and Red Hat 7 operating systems.

Here the requirements from *Section 1.1* are reviewed to see if they are adequately satisfied in NetClasses:

- **Lecturer to post notes.** This requirement is met with the user able to upload files of any type in an orderly fashion. Additionally any link on the page can be made visible or hidden both at the time of creation and later from a menu option

  There is a security risk, though, in allowing files of any type to be uploaded. A devious user could upload a JSP file as part of the notes. If that JSP file contained code, it would be executed when the notes are viewed. The student reading the notes would not see the JSP file as written, but instead, the file would be executed and the student would see any output.

- **Lecturer to assign passwords to students.** This works. When lecturers populate a class they can, and must, assign a password to every student. When developing this module it was thought that the passwords would be the student numbers that are normally available from the same college file as the student names.

Students can change their own passwords. Every user has access to the jump menu that has a Change Password option. As well as that, NetClasses caters for students who forget their password. Administrators can change a student's password. This facility might also be useful if the administrator wanted to withdraw a particular student's right to use the system.

No method is included for having a visitor to the system. If a lecturer wants to make the notes available to the public, it should be possible to do so. This could be done by having a student with username 'visitor' programmed into every class and that student would be able to log on with no password or with any password. When a lecturer did not want the notes to be public that lecturer could simply remove visitor student from the class. The visitor student would not be able to upload files.

- **Lecturer to post assignment instructions.** The requirement here is met. A lecturer can post as many files as he wants to describe an assignment. The same problem applies here to JSP files as in the post notes section. The JSP file cannot be viewed and if it contains code, the code will be executed.

- **Lecturer to collect assignments.** Lecturers can only view files submitted by students. If the lecturer needs to collect the files on their own machine (assuming the server is a different machine) then they must choose to save the file again. This means that the lecturer can collect the assignments, but collect them file-by-file instead of all together. This slightly different outcome still fulfils the requirement. The files are collected together on the server. Links to all the files submitted by students can be seen by the lecturer on one page. In that sense the lecturer can collect the assignments.

- **Lecturer to present multiple-choice quizzes.** The requirement here is met. The lecturer can create quizzes. The quizzes can be made available from the student menu or from links on the notes page, or from both. In addition quizzes can be modified by adding, deleting and editing questions.

  Quizzes can be hidden from the student until such time as the lecturer activates them.

- **Lecturer to operate a notice board.** Messages can be added to the system for the students of any class. The messages can be deleted as well. This fulfils the message board requirement. In addition, individual students only see a warning that new messages exist if there are messages that they have not read.

- **Student to read notes.** This works well. There is a web page where the student can see the links to the lecturer's notes. The web page is laid out logically with the links grouped together under chapter headings. When the student is finished reading a notes file, the page can be closed and the NetClasses session resumes.

- **Student to read assignment instructions.** Like the notes, the student has a web page where there are links to the assignment instructions. The requirement is met.

- **Student to submit assignments.** This requirement is met in NetClasses. A student can submit as many files as required. The system is open to attack here in that an upper limit on the total size of files submitted by a student is not included. This is not too difficult to implement because the size is easily obtained from the uploading code. The size needs to be recorded in the database so that the total size of previous files plus the file being uploaded can be calculated. If the size is over a predefined limit, then the uploading can be aborted.

The problem noted above about JSP files being uploaded applies here too. JSP files can be uploaded by the student as part of the assignment files and that file could contain JSP code that would be executed on the server.

- **Student to take multiple-choice quizzes.** This works. Students can take a quiz from either the student menu or from links on the notes page. The quiz is graded immediately it is submitted and the student gets back a mark out of 100 and a list of questions incorrectly answered. The mark is not recorded in the database so it works only as a self test for the student.

- **Student to read notices on the notice board.** This requirement is met. In addition students are alerted about new messages when they log on. As soon as a student reads a message, that fact is stored in the database. The next time the user logs on previously read messages are not treated as new.

- **A requirement in the original project proposal is that all pages within the system have the same look and feel.** This is achieved, not by using cascading style sheets as might be supposed, but by including at the top of each JSP file another file that contains most of the layout descriptors for the page. Each page has a border around it and a logo and a message on the top. It was intended that the logo and message could be changed by the administrator but that has not yet been implemented. It was also proposed that pages belonging to one lecturer would be especially homogonous with regard to look and feel. In the present implementation there is no differentiation between the pages of one lecturer and another i.e. they all look the same and the lecturer cannot alter the appearance.

6.1 Further Work

The biggest thing lacking in NetClasses is the ability to empty a class of students. That would need to be done at the end of an academic year when the names of the students would change. At the moment a lecturer can delete a class but this gets rid of the notes as well as the students. This development of the project would take a small amount of work as the code for deleting a single student from a class is already there

Another related improvement would be to provide a facility for the main administrators to be able to delete students from the system if they are not members of a class group. At the moment students are added to the system every time they are added to a class and are not in the system already, but they are never deleted.

The next most glaring omission is that the administrator cannot change the appearance of the pages. This is a must-do because at the moment the logo is peculiar to CIT. It would be a matter of uploading the logo file to the correct folder. At the moment there are two copies of the two images that are used to form the top of the web pages. There was a problem in Internet Explorer that resulted in the pictures not being shown on a web page under some conditions when the page is revisited. The problem does not occur in NetScape. A quick fix is to have two copies of the files in the system. Relative addressing to a sub folder with the files in it is used. If absolute addressing within the NetClasses context is used, the pictures sometimes do not appear. Even with that less than perfect architecture, it is still possible to code for the administrator to change the pictures. It would take some further work to solve the multiple storage problem.

Without the facility to change the logo and message through the web interface, it has to be done manually i.e. logo.jpg and message.jpg must be copied in to the correct places in the file structure.

In the notes module there is room for improvement by providing a facility to put links to other web pages on the notes page. At the moment such links can be put in any file uploaded to the notes folder but not directly on the notes page. The only links on the notes page now are links to files and links to quizzes.

Appendix 1

Bibliography

BAUE99	Bauer, Marian et al. Transforming Universities. Jessica Kingsley Publishers, 1999. Page 238.
BAXL02	Baxley, Bob. Making the Web Work: Designing Effective Web Applications. New Rider, 2002. Page 223.
BELL03	Bell, John T. and Lambros, James and Ng, Stan. J2EE Open Source Toolkit: Building an Enterprise Platform with Open Source Toolkit. Wiley, 2003.
BERG00	Bergsten, Hans. Java Server Pages. O'Reilly, 2000.
BUSC96	Buschmann, Frank et al. Pattern-Oriented Software Architecture, Volume 1: A System of Patterns. Wiley 1996. Page 125.
BROW02	Brown, Jeffrey and Thomas, Susan L. and Bruzzese, Peter J. Site and E-Commerce Design. Sybex, 2002. Pages 50 and 501.

CARE02	Carey, Patrick. New Perspectives on Creating Web Pages with HTML and XML. Course Technology, 2002.
CENT03	Central Applications Office. CAO Handbook, Irish Government Stationary Office, 2003.
COLE00	Coleman, Pat et al, eds. Perl, CGI and Java Script Complete. Sybex, 2000. Page 209.
COOK03	Cook, Todd. Mastering JSP. Sybex, 2003. Page 7.
DEIT02	Deitel, H.M. and Deitel, P.J. and Neito,T.R. Internet & World Wide, Web How to Program. Prentice Hall, 2002. Chapter 8.
DOHE94	Doherty, Geoffrey D. Developing Quality Systems in Education. Routledge, 1994. Page 181.
FIEL02	Fields, Duane K. and Kolb, Mark A. and Bayern, Shawn. Web Development with Java Server Pages. Manning Publications Co., 2002.

FORO03	Forouzan, Behrouz A. TCP/IP Protocol Suite. McGraw-Hill, 2003. Chapter 8.
GALI02	Galitz, Wilbert O. The Essential Guide to User Interface Design, 2nd Edition. Wiley, 2002. Page 259.
GANG02	Ganguli, Madhushree. Making Use of JSP. Wiley, 2002. Page 24.
GOOD02	Goodwill, James. Pure JSP: Java Server Pages. Sams, 2002.
HALL03	Hall, Marty and Brown, Larry. Core Servlets and JavaServer Pages, Vol. 1, Second Edition. Prentice Hall PTR, 2003.
HOLM02	Holmes, Merlyn. Web Usability & Navigation. McGraw Hill/Osborne, 2002. Page 11.
HOLZ01	Holzshalag, Molly E. XML, HTML, XHTML Magic. New Riders, 2001.

INCE02	Ince, Darrel. Developing Distributed and E-Commerce Applications. Pearson Education, 2002. Pages 54 and 134.
ISAA01	Isaacs, Ellen and Walendowski, Alan. Designing from Both Sides of The Screen. New Riders, 2001.
JOHN03	Johnson, Jeff. Web Bloopers: 60 Common Web Design Mistakes. Morgan Kaufmann, 2003. Page 53.
JONE97	Jones, David and Jamieson, Bruce. Three Generations of Online Assignment Management. Proceedings of ASCILITE'97, Perth, Australia. Kevil, Rod and Oliver, Ron and Phillips, Rob, eds. 1997. Page 317.
LIUM03	Liu M.L. Distributed Computing: Principles and Applications. Pearson Education, 2003.
MARA00(a)	Maran, Ruth. Active Server Pages 3.0. IDG Books Worldwide, 2000. Page 40.

MARA00(b)	Maran, Ruth. HTML, Your visual blueprint for designing effective Web pages. IDG Books Worldwide, 2000.
MARI02	Marini, Joe. Document Object Model: Processing Structured Documents. Osborne/McGraw-Hill, 2002.
MELO03	Meloni, Julie. PHP Essentials, 2nd Edition. Premier Press, 2003. Page 22.
MISR01	Misra, Jayadev. A Discipline of Multi Programming: Programming Theory for Distributed Applications. Springer-Verlag, 2001.
MORR01	Morrison, Michael. HTML & XML for Beginners. Microsoft Press, 2001.
NIEL00	Nielson, Jakob. Designing Web Usability: The Practice of Simplicity. New Riders, 2000. Pages 48 and 188.

ORFA99	Orfali, Robert and Harkey, Dan and Edwards, Jeri. Client/Server Survival Guide, 3rd Edition. Wiley, 1999. Page 105.
PFAF00	Pfaffenberger, Bryan and Karrow, Bill. HTML 4 Bible. Wiley, 2000.
POWE01	Powell, Thomas A. HTML: The Complete Reference, Third Edition. Osborne/McGraw-Hill, 2001.
PREE02	Preece, Jenny and Rogers, Yvonne and Sharp, Helen. Interaction Design. Wiley, 2002.
RAJA02	Rajagopolan, Suresh et al. Java Servlet Programming Bible. Wiley, 2002. Page 6.
ROBP01	Rob, Peter and Coronel, Carlos. Database Systems: Design, Implementation and Management, Fifth Edition. Course Technology, 2001.
RUDI96	Rudisill, Marianne et al. Human-Computer Interface. Morgan Kaufmann, 1996.

RUSS00	Russell, Gavin and van Leeuwen, Jannetje. Extranet and Courseware for the MSc in Multimedia Technology. UCC, 2000.
SACH96	Sachs, David and Stair, Henry. The 7 Keys to Effective Web Sites. Prentice Hall, 1996.
SCHR98	Schreiber, Deborah A. and Berge, Zane L., eds. Distance Training. Jossey-Bass Inc., 1998. Page 145.
SPEI03	Speilman, Sue. The Struts Framework: Practical Guide for Java Programmers. Morgan Kaufmann, 2003.
SPOL01	Spolsky, Joel. User Interface Design for Programmers. Apress, 2001.
STER00	Sterling, Scott M. Java Server Pages Application Development. Sams, 2000.
TANE03	Tanenbaum, Andrew S. Computer Networks, Fourth Edition. Prentice Hall PTR, 2003.

TAYL98	Taylor, Christopher and Kimmett, Timothy. Core Java Web Server. Prenice Hall, 1998.
TOGN92	Tognazzini, Bruce. Tog on Interface. Addison-Wesley Longman, 1992.
VALE99	Valesky, Thomas. Enterprise Java Beans (TM): Developing Component-Based Distributed Applications. Addison-Wesley, 1999.
WATS97	Watson, Mark. Creating Java Beans: Components for Distributed Applications. Morgan Kaufmann, 1997.
WILL99	Williamson, Alan R. Java Servlets by Example. Manning Publications, 1999.
YANK03	Yank, Kevin. Build Your Own Database Driven Website Using PHP & MySQL, Second Edition. Sitepoint, 2003. Page 173.

Appendix 2

Code

Login Page Code

```
<jsp:useBean id="lecturerbean" scope="request"
class="NetClasses.beans.UserBean">
</jsp:useBean>

<%
String lecturer_firstName = new String("");
String lecturer_lastName = new String("");
String lecturer_username = new String("");

String oldUsername = request.getParameter("username");

if(oldUsername == null)
{
   oldUsername = "";
}


//This bit gives an error message if the previous
//login atttempt was faulty

boolean retry = false;

if (request.getParameter("retry") != null)
{
   retry = true;

}
%>

<html>
<head><title>Login in here</title>

</head>
<script>

function prepare()
{
   if(<%= retry %>)
   {
       document.getElementById("headline").innerHTML="Incorrect
user/password combination . Try again.";
   }
}
</script>
<body onload="prepare();">
<center>

<jsp:include page="banner.jsp" flush="false"/>

<form name="loginForm"
      action='<%= response.encodeURL(request.getContextPath() +
"/servlets/login") %>'
      method="POST"
>
```

```
<tr>
   <td width="90%" valign=top style="padding:10px">
   <table cellspacing=0 cellpadding=0 width="100%">
   <tr>
      <td><span id="headline"> Welcome to NetClasses online notes and
course material facility</span></td>
   </tr>
   </table>



<input type="hidden" name="loginTarget" value="welcomePage">
<input type="hidden" name="status" value="ok">


<table cellpadding=10 cellspacing=0 border=0 width="100%">
<tr><td width="320" valign=top>

   <table cellpadding=5 cellspacing=0 border=0 bgcolor="#FFFFFF"
          style="border:1px solid black;" align=center>

   <tr>
      <td style="color:black;background-color:#FFCCCC;border-bottom:ipx
solid black;">
      Enter username and password, then login
      </td>
   </tr>
   <tr>
      <td>
      <table cellpadding=5 cellspacing=0 border=0 width="300"
align=center>

   <tr><td><span style="width:40%">Your Username:</span>
       <input type="text" name="username" align="left"
                  style="width:50%;background-color:#FEFFBB;"
                  size="20" value="<%= oldUsername %>">
   </td></tr>

   <tr><td><span style="width:40%">Password:</span>
       <input type="password" name="password" align="left"
                  style="width:50%;background-color:#FEFFBB;"
                  size="20">
   </td></tr>

   <tr><td width="100%" align="center"><table>
   <tr><td width="50%" align="right">
      <input type="submit" value="Log In">
      </td>
      <td width="50%" align="left">
      <input type="submit" value="Cancel"
onClick="document.loginForm.status.value='cancel'">
      </td>
   </td></tr>
  </table></td></tr>
```

```
</table>
</td></tr>
</table>
</td>

</td>
<td><p><b>Students:-</b>You can access notes that your lecturer posts on
this site. You can also download assignment instructions, submit your
assignment work, take quizzes and read messages from your lecturer.</p>
    <p><b>Lecturers:-</b>You can post your notes, give students
individual usernames and passwords, post and collect assignments, set
quizzes and broadcast messages to classes.</p>

</td>

</tr>
</table>

</form>
</tr>
</table>

</center>
</body>
</html>
```

Class Menu Servlet Code

```
package NetClasses.secure.servlets;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import java.lang.Integer.*;
import NetClasses.beans.*;

public class LecturerMenuClassMenuServlet extends HttpServlet
{
   private Vector commands = new Vector();
   private Vector secondCommands = new Vector();
   private Vector thirdCommands = new Vector();
private UserBean user;
private DatabaseListBean list_maker;
private String next;

private Command default_command;
private String errorPage = "lecturer/classMenu/error.jsp";

public void init(ServletConfig config) throws ServletException
{
   super.init(config);
   default_command = new ForwardCommand(
      "lecturer/classMenu/classMenu.jsp");

    try
   {
         list_maker = new DatabaseListBean();
   }
   catch(Exception e)
   {
         list_maker = null;
   }

   /////////////////////////First Commands/////////////////////
   commands.add(0,
   new LecturerCreateClassCommand_1(
   "lecturer/classMenu/lecturerCreateClass1.jsp", 1, 0));

   commands.add(1,
   new LecturerDeleteClassCommand_1(
   "lecturer/classMenu/lecturerDeleteClass1.jsp", 1, 1));



   commands.add(2,
   new LecturerPopulateClassCommand_1(
   "lecturer/classMenu/lecturerPopulateClass1.jsp", 1, 2));

   commands.add(3,
   new LecturerPopulateClassCommand_1(
   "lecturer/classMenu/lecturerRemoveStudent1.jsp", 1, 3));

   commands.add(4,
   new LecturerPopulateClassCommand_1(
   "lecturer/classMenu/lecturerEditStudent1.jsp", 1, 4));

   commands.add(5,
   new LecturerAddSubjectCommand_1(
   "lecturer/classMenu/lecturerAddSubject1.jsp",1, 5));

   commands.add(6,
   new LecturerDeleteSubjectCommand_1(
   "lecturer/classMenu/lecturerDeleteSubject1.jsp",1, 6));

   /////////////////////////Second Commands ///////////////////////
   secondCommands.add(0,
   new LecturerCreateClassCommand_2(
   "lecturer/classMenu/lecturerCreateClass2.jsp", 2, 0));
```

```
    secondCommands.add(1,
    new LecturerDeleteClassCommand_2(
    "lecturer/classMenu/lecturerDeleteClass2.jsp", 2, 1));

    secondCommands.add(2,
    new LecturerPopulateClassCommand_2(
    "lecturer/classMenu/lecturerPopulateClass2.jsp", 2, 2));

    secondCommands.add(3,
    new LecturerRemoveStudentCommand_2(
    "lecturer/classMenu/lecturerRemoveStudent2.jsp", 2, 3));

    secondCommands.add(4,
    new LecturerEditStudentCommand_2(
    "lecturer/classMenu/lecturerEditStudent2.jsp", 2, 4));


    secondCommands.add(5,
    new LecturerAddSubjectCommand_2(
    "lecturer/classMenu/lecturerAddSubject2.jsp",2, 5));

    secondCommands.add(6,
    new LecturerDeleteSubjectCommand_2(
    "lecturer/classMenu/lecturerDeleteSubject2.jsp",2, 6));

    ////////////////////// Third Commands//////////////////////////
    thirdCommands.add(0, default_command);

    thirdCommands.add(1, default_command);



    thirdCommands.add(2, new LecturerPopulateClassCommand_3(
    "lecturer/classMenu/lecturerPopulateClass3.jsp", 3, 2));

    thirdCommands.add(3, default_command);

    thirdCommands.add(4, default_command);

    thirdCommands.add(5, default_command);

    thirdCommands.add(6, default_command);


}

public void service(HttpServletRequest req, HttpServletResponse res)
{
    if(list_maker == null)
    {
           next = errorPage + "?message=";

       String error = "There is no Database Connection";
       next = next + error;

       gotoNextPage(req, res);
```

```
      return;
   }

   String temp_string = req.getParameter("ClassMenuIndex");
   int index;

   try
   {
     index = Integer.parseInt(temp_string,10);
   }
   catch(NumberFormatException e)
   {
         //There was no good index in the request Parameter
         index = -1;
   }

   Integer next_index = new Integer(index);

   temp_string = (String)req.getParameter("ClassMenuLevel");
   int level;
   Integer next_level;


   try
   {
     level = Integer.parseInt(temp_string,10);
   }
   catch(NumberFormatException e)
   {
         //There was not a good level in the request Parameter
         level = -1;
   }

   level++;

   //To avoid getting stuck in a rut we should get out now if the
   //level is still error (0)
   if(level == 0)
   {
      try
      {
         next = default_command.execute(req,res);
      }
      catch(CommandException e)
      {
         //do nothing because we will be able to get out later
         //This code is here to stop the error of having no classes
         //selected resulting in a flow control infinite loop.
      }
      gotoNextPage(req, res);
      return;
   }


   //We should not service a request to do anything with Class Menu
   //unless the lecturer is focused on a course.
```

```
//If there are no courses stored for this lecturer then they must
//be told that there are no courses and they should create one
//if index = 1 (Create a class) then execution is allowed
HttpSession session = req.getSession(true);
String className = (String)session.getAttribute("className");
String subject = (String)session.getAttribute("subject");

if((className == null || className.equals("")) && index != 0)
{
   user = (UserBean)session.getAttribute("user");
   String user_name = user.getUsername();
   Vector list = list_maker.createClassesList(user_name);

   if(list.size() == 0)
   {
      next = errorPage + "?message=";

      String error = "There is no course (class + subject) " +
                     "in the\n database for " + user_name +
                     ".\n You can create one by going to "
                     "Create Class\n in the current menu.";
      next = next + error;
   }
   else
   {
      next = errorPage + "?message=";

      String error = "There is no course (class + subject) " +
                     "selected\n " +
                     "You can select one by going to Another " +
                     "Course on the Main Menu.";

      next = next + error;
   }

   gotoNextPage(req, res);

   return;
}


next_level = new Integer(level);

if(next_level.intValue() == 1) //then the request came from a
                                //menu click
{   session.setAttribute("ClassMenuLevel",next_level);
    session.setAttribute("ClassMenuIndex", next_index);
}
else //the request came from an intermediate command
{
    //We just increment the session level and leave
    //the index alone
    Integer session_level = (Integer)session.getAttribute(
                                         "ClassMenuLevel");
    int temp = session_level.intValue();
    temp++;
    Integer session_level_plus_1 = new Integer(temp);
```

```
            session.setAttribute("ClassMenuLevel",session_level_plus_1);
        }

        //either way, the request Attributes will show this index and
        //the next level
        req.setAttribute("ClassMenuLevel", Integer.toString(level));
        req.setAttribute("ClassMenuIndex", Integer.toString(index));

        //System.out.println("ClassMenuLevel = " + level);
        try
        {
           if(level == 1)
              next = ((Command)commands.get(index)).execute(req, res);
           else if(level == 2)
              next = ((Command)secondCommands.get(index)).execute(
                                                        req, res);
           else if(level == 3)
              next = ((Command)thirdCommands.get(index)).execute(
                                                        req, res);
           else
              next = default_command.execute(req,res);
        }
        catch( CommandException e)
        {
           req.setAttribute( "exception", e );
           next = errorPage;
        }


        gotoNextPage(req, res);

        return;

    }

    private void gotoNextPage(HttpServletRequest req,
                                          HttpServletResponse res)
    {

        RequestDispatcher rd;
        rd = getServletContext().getRequestDispatcher("/secure/" + next);
        try
        {
           if(rd == null)System.out.println("Sister Mary");
           //System.out.println("About to forward to " + next);
           rd.forward(req,res);
           return;
        }
        catch(ServletException e)
        {
           System.out.println("Servlet Error forwarding page "
           + next);//What to do?
        }
        catch(java.io.IOException e)
        {
           System.out.println("IO Error forwarding page " + next);
          //What to do?
        }
    }
}
```

Code For Create Class Command 2

```
package NetClasses.secure.servlets;

import NetClasses.servlets.*;
import NetClasses.beans.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

public class LecturerCreateClassCommand_2 implements Command
{
   private String next;
   private UserDataBaseBean userDB;
   private int level;
   private int index;
   private DatabaseListBean list_maker;
   private Vector classNames;
   private String className;
   private String subject;
   private String lecturer;
   private UserBean user;


   public LecturerCreateClassCommand_2(
                                  String next, int level, int index)
   {
      this.next = next;
      this.next = next;
      this.level = level;
      this.index = index;

      try
      {
            list_maker = new DatabaseListBean();
      }
      catch(Exception e)
      {
            list_maker = null;
      }

      userDB = UserDataBaseBean.getSingleton();
   }

   public String execute(
                     HttpServletRequest req, HttpServletResponse res)
```

```
throws CommandException
{
   if(list_maker == null || userDB == null)
   {
      String message = " No database connection";
      String error_next = "lecturer/classMenu/error." +
                          "jsp?message=";
      error_next = error_next + message;
      return error_next;
   }


   HttpSession session = req.getSession(true);

   //Check to see if the user pressed cancel
   String status = req.getParameter("status");
   if(status != null && status.equals("cancel"))
      return "lecturer/classMenu/classMenu.jsp";

   //Check to see if this command is the one currently
   //being executed in the session. If not then it did not
   //get requested from a menu but by resubmitting some
   //page. It will only be serviced if current.
   Integer session_level =
         (Integer)session.getAttribute("ClassMenuLevel");
   Integer session_index =
         (Integer)session.getAttribute("ClassMenuIndex");

   if(   level != session_level.intValue()
      || index != session_index.intValue())
   {
         //System.out.println("level = " + level);
         //System.out.println("session_level = " + session_level);
         return "lecturer/classMenu/classMenu.jsp";
   }

   classNames = (Vector)session.getAttribute("ClassMenuClassList" );

   if(classNames == null)
   {
      //System.out.println("The class list is lost");
      return "lecturer/classMenu/classMenu.jsp";
   }

   subject = req.getParameter("subject").trim();

   if(subject.equals(""))
   {
      String message = " Subject must not be null";
      String error_next = "lecturer/classMenu/error." +
                          "jsp?message=";
      error_next = error_next + message;
      return error_next;
   }

   className = req.getParameter("class_name").trim();
```

```
if(   className.equals("")
   || !isClassNameValid(className, classNames))
{
   String message = className + " is not valid";
   String error_next = "lecturer/classMenu/error." +
                       "jsp?message=";
   error_next = error_next + message;
   return error_next;
}

user = (UserBean)session.getAttribute("user");
if(user == null)
{
   //System.out.println("The user bean is lost");
   return "lecturer/classMenu/classMenu.jsp";
}

lecturer = user.getUsername();

if(lecturer == null)
   lecturer = new String("");

//Add the table to the database
if(!userDB.createClassTable(className, lecturer, subject))
{
   String message = "The " + className +
                  " table cannot be created";
   userDB.deleteClassTable(lecturer, className);

   String error_next = "lecturer/classMenu/error." +
                       "jsp?message=";
   error_next = error_next + message;
   return error_next;

}

//Add the corresponding folders
try
{
  //Create the directory for the new lecturer
  File f = new File("../webapps/NetClasses/lecturers/"
            + lecturer + "/" + className + '_');

  if(f == null || !f.mkdir())
  {
     String message = "Unable to create folder "
        + lecturer + "/" + className;
        String error_next =
           "lecturer/classMenu/lecturerCreateClassError2.jsp" +
           "?message=" + message;
     return error_next;
  }

}
catch(SecurityException e)
{
   String message = "Unable to create folder  " + className;
```

```
            String error_next =
              "lecturer/classMenu/error.jsp" +
              "?message=" + message;
        return error_next;
    }

    if(!addSubjectFolder() || !addNotesFolder())
    {
        String message = " Folders for " + subject +
                                   " cannot be added.";
        String error_next = "lecturer/classMenu/error." +
                             "jsp?message=";
        error_next = error_next + message;
        return error_next;
    }

    classNames = list_maker.createClassesList(lecturer);

    session.setAttribute("ClassMenuClassList", classNames);
    req.setAttribute("class_list", classNames);

    //this course now becomes the one the lecturer is currently
    //focussed on
    session.setAttribute("className", className);
    session.setAttribute("subject", subject);

    return next;

}



private boolean isClassNameValid(
                     String className, Vector classNames)
{
    for(int i = 0; i < classNames.size(); i++)
    {
        if(className.equalsIgnoreCase(
                                  (String)classNames.elementAt(i)))
            return false;
    }

    if(     className.equalsIgnoreCase("lecturer")
        || className.equalsIgnoreCase("administrator"))
    {
          return false;
    }
    return true;

}

private boolean addSubjectFolder()
{
     //Add the corresponding folders
```

```
        try
        {
          //Create the directory for the new class
          File f = new File("../webapps/NetClasses/lecturers/" +
                   lecturer + "/" + className + "_/" + subject + '_');

          if(f == null || !f.mkdir())
          {
             return false;
          }

        }
        catch(SecurityException e)
        {
           return false;
        }

        return true;
    }



    private boolean addNotesFolder()
    {
          //Add the corresponding folders
        try
        {
          //Create the directory for the new class
          File f = new File("../webapps/NetClasses/lecturers/" +
               lecturer + "/" + className + "_/" + subject + "_/notes" );

          if(f == null || !f.mkdir())
          {
             return false;
          }

        }
        catch(SecurityException e)
        {
           return false;
        }

        return true;
    }
}
```

Upload Server Bean Code

```
package NetClasses.secure.beans;
import javax.servlet.ServletInputStream;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.*;
import java.util.*;


public class UploadServerBean
{
   private String savePath;
   private String originalFileName;
   private Dictionary fields;
   private ServletInputStream receivedFile;
   private String lineString;
   private byte[] boundary_array;
   private int boundary_length;
   private FileOutputStream target_file;
   private byte prev_c;
   private byte[] buffer;
   private byte[] mid_buffer;
   private byte[] oldest_buffer;
   private byte[] temp;


   public String getOriginalFileName()
   {
            return originalFileName;
   }

   public String getSavePath()
   {
         return savePath;
   }

   public void setSavePath(String path)
   {
         savePath = path;
   }

   public UploadServerBean()
   {
        boundary_array = new byte[128];
        receivedFile = null;
        buffer = new byte[1024];
        temp = new byte[1024];
        mid_buffer = new byte[1024];
        oldest_buffer = new byte[1024];
        originalFileName = "";
        savePath = "";
        fields = new Hashtable();
   }
```

```
public boolean doUpload(HttpServletRequest request ,String path)
{
     setSavePath(path);
     return doUpload(request);
}

public boolean doUpload(HttpServletRequest request)
{
   try
   {
     //now the file
     int i = readFile();

     if(i == -1)
        return false;

     //Now there could be more field-value pairs

     //Another name value pair
     i = receivedFile.readLine(buffer, 0, 1024);









     //Continue with the field value pairs until EOF
     while(!(isByteArrayEOFBoundary(buffer, i) || i == -1))
     {
        readFieldValuePairs(i);

        //A boundary
        i = receivedFile.readLine(buffer, 0, 1024);
        if(i == -1)
        return true;

         //Another name value pair
        i = receivedFile.readLine(buffer, 0, 1024);
        if(i == -1)
        return true;
     }
   }
   catch(java.io.IOException e)
   {
         System.out.println("Error reading file");
         return false;
         //e.printStackTrace();
   }

   return true;
}

public Dictionary getParameters()
{
        return fields;
    }


    //This section is for parameters that come before the file
    public Dictionary readParameters(HttpServletRequest request)
    {
        try
        {
          receivedFile = request.getInputStream();

          //read first line. It is a boundary marker
          int i = receivedFile.readLine(buffer, 0, 1024);

          //First line should be a boundary
          if(i < 3)
             return null;

          boundary_length = i;

          //Put boundary in boundary_array
          for(int j = 0; j < i; j++)
          {
              boundary_array[j] = buffer[j];
          }




          int count = 0;

          //First of the Field-Value pairs in three lines
          // ( name - value - boundary )
          i = receivedFile.readLine(buffer, 0, 1024);
          lineString = new String(buffer, 0, i);

          while(!(
                 lineString.startsWith(
                              "Content-Disposition: form-data; name=")
                 &&
                 (lineString.indexOf("filename=\"") != -1)
                 )
                 && i != -1)
          {

             //The Field-Value pairs must be extracted from the
             //the three lines. Two more lines will be read in
             // this function.
             readFieldValuePairs(i);

             //Boundary after one pair
             i = receivedFile.readLine(buffer, 0, 1024);

             //the first of another three lines( name - value - boundary)
             i = receivedFile.readLine(buffer, 0, 1024);
             lineString = new String(buffer, 0, i);
```

```
      count++;
   }

   if(i == -1)//There is no file to upload because we did not
              // find "filename=\"
   {
     return fields;
   }
   //The last of the Field-Value pairs
   //before the file content (this is the filename).
   //The filename Field-Value pair are not in the three line
   //format

   String fieldname = "";
   StringTokenizer st = new StringTokenizer(lineString, "=");
   if(st.hasMoreTokens())
   {
       //Get fieldname
       st.nextToken("=");
       if(st.hasMoreTokens())
       {
         st.nextToken("\"");

         if(st.hasMoreTokens())
            fieldname = st.nextToken("\"");

         else return fields;
       }

       else
         return fields;
    }
    else return fields;

    //get filename
    st.nextToken("=");
    if(st.hasMoreTokens())
    {
         st.nextToken("\"");

         if(st.hasMoreTokens())
           originalFileName = st.nextToken("\"");

         else return fields;
     }
     else
        return fields;

   fields.put(fieldname, originalFileName);

   //read the next line with type info
   i = receivedFile.readLine(buffer, 0, 1024);

   //There is a blank line then
   i = receivedFile.readLine(buffer, 0, 1024);

 }
```

```
    catch(java.io.IOException e)
    {
       System.out.println("Error reading file");
       //e.printStackTrace();
    }

    return fields;
  }



private int readFile()
{
   int count = -1;
   int temp_count;
   try
   {
      //start reading file

      //System.out.println("Save path = " + getSavePath());

     target_file =
           new FileOutputStream(new File(getSavePath()));


     //We keep two buffers going. We write the oldest buffer
     //to file while the current buffer is not a boundary. There is
     //third buffer to make it possible to swap the other two.
     int count_oldest = receivedFile.readLine(oldest_buffer, 0, 1024);
     if(count_oldest == -1) return count_oldest;

     count = receivedFile.readLine(buffer, 0, 1024);
     if(count == -1)
       return count;

     while(count != -1 && !isByteArrayBoundary(buffer, count))
     {
        target_file.write(oldest_buffer, 0, count_oldest);

        //promote the other buffer

        //Shallow copy but thats ok because they are not meant to
        //be copies but swaps
        temp = oldest_buffer;
        temp_count = count_oldest;

        oldest_buffer = buffer;
        count_oldest = count;

        buffer = temp;
        count = temp_count;

        count = receivedFile.readLine(buffer, 0, 1024);
     }

     //the last buffer needs to be written
```

```
      target_file.write(oldest_buffer, 0, count_oldest);

      target_file.close();
    }
    catch(java.io.IOException e)
    {
       System.out.println("Error reading or writing file in\n" +
                          "UploadServerBean.java");
       //e.printStackTrace();
       return -1;//Interpreted as an error in calling function
    }

    return count;
  }

private boolean isByteArrayBoundary(byte[] buff, int length_buff)
{
       //First check is it the end of file boundary
       //The end of file boundary has two more characters than
       //a normal boundary
       if(length_buff == boundary_length + 2)
       {
              return isByteArrayEOFBoundary(buff, length_buff);
       }

       //Then check is the length right
       if(length_buff != boundary_length)
       {
              return false;
       }

       //Finally the most CPU expensive check. Each element is compared
       for(int i = 0; i < boundary_length; i++)
       {
              if(boundary_array[i] != buff[i])
              {
                     return false;
              }
       }

       //if we get this far then it is a boundary
       return true;
}

//This function should only be called if the boundary_array has been
//read.
private boolean isByteArrayEOFBoundary(byte[] buff, int length_buff)
{
     if(length_buff != boundary_length + 2)
     {
              return false;
     }

     for(int i = 0; i < boundary_length; i++)
     {
       if(boundary_array[i] != buff[i])
       {
            //If we get as far as the endline character then it is an EOF
            //because we already established that it is the right length
            if((boundary_array[i] == '\n') || (boundary_array[i] == '\r'))
            {
               return true;
            }

            return false;
         }
       }


    //We should never get this far because that would mean the two
    //arrays both ending with with a newline character and one two
    //characters longer than the other were equal
    return false;
}

private void readFieldValuePairs(int i)
{
    String name = null;
    String value = null;
    try
    {
       int start = lineString.indexOf("=");

       if(start != -1)
       {
          StringTokenizer st = new StringTokenizer(lineString, "=");
          if(st.hasMoreTokens())
          {
                 //Get name of field
                 st.nextToken("=");
                 if(st.hasMoreTokens())
                 {
                    st.nextToken("\"");

                    if(st.hasMoreTokens())
                       name = st.nextToken("\"");
                    else name = "";
                 }
                 else
                 name = "";
          }
          else name = "";

          //Blank line
          i = receivedFile.readLine(buffer, 0, 1024);
          lineString = new String(buffer, 0, i);

          //value
          i = receivedFile.readLine(buffer, 0, 1024);
          lineString =  new String(buffer, 0, i);


          //The value is in a line of it's own
          //We need to get the value without the end of line characters
          value = new String("");

          for(int j = 0; j < lineString.length(); j++)
          {
            char c = lineString.charAt(j);

            if(c != '\n' && c != '\r')
            {
                  value = value + c;
            }
          }


          fields.put(name, value);

          //System.out.println("name = " + name);
          //System.out.println("value = " + value);

       }
     }
     catch(java.io.IOException e)
     {
      System.out.println("Error in file headers or footers\n" +
                        "in UploadServerBean.java");
      //e.printStackTrace();
     }
   }
}
```