Master of Science Project

Final Audit

Montclair State University
Facility: Dr. J.Jenq

Development:
Student: Tatyana Sigalovich

Contents

Acknowledgment

I would like to express my gratitude to Dr. J. Jenq for his contribution to the success of this master project. His deliberations and considerable effort to collect and come up with the comprehensive set of requirements served an important role in shaping the application.

I also would like to thank many faculty members at Computer Science department who helped me learn many topics in Computer Science, which enabled me to achieve my academic goals.

Thanks to my family for their moral support in the effort of writing this project.

Abstract

This is to propose a new application to be used by the Registrar's Office and students that would facilitate daily operation by the department staff and organize and automate students' application processing. The application will have three types of users, namely, the department clerk, a student, and a system manager. The students will be able to initiate, view, and interfere with their application processing. This will increase transparency of the overall process, enable early error detection, and simplify, automate, and audit-trail the communication between students and the department personnel. It will also free the Registrar's Office clerks from the otherwise necessary volume of inquiries by the students about the application processing status. The clerk interface will automate the day-to-day department staff operations allowing them to process higher volume of application more efficiently, in shorter period of time, and with fewer people. The administrative component will allow the application to adjust to the changing environment in which the application will be used. It will be possible to modify the application processing process, address most of the new regulations, rules, or any other changes to the application processing procedure. The automation of the process will also address compliance of the overall process where all changes made in the application will be audit-trailed. Any compliance rules can be enforced in the application and violations tracked by the application itself. Potentially, such an application will allow the Registrar's Office personnel and the students to have automatic reminders alerting them about upcoming events and procedure deadlines. In addition, a reporting component can be added to the system allowing the management to view various statistics and information about the application processing state.

Introduction

The idea of this project belongs to Dr. J. Jenq. The automation of the final audit process can help the Registrar's Office to keep track of all student applications, verify eligibility for graduation against student records, get an exception report, and update the status. As a result the Registrar's Office can make a final decision quickly with fewer clerical errors and smaller staff. On the other hand, the students will be able to view the progress of their applications on-line. The idea of this application came about to solve a paperwork-intensive situations that often arise while students clear their status for graduation. For example, currently, if a student wants to substitute an elective class that s/he has taken for one of the required classes, the process requires the student to pick up an application from the Registrar's Office, get the application signed by the department offering the class, and finally getting the approval by the Registrar's Office allowing the submit the final audit application. This application is meant to automate this (as well as others) paperwork and time intensive process simplifying life for the students and the university's staff.

This document lists all the functional and non-functional requirements for the application, as well as a technical description of the application architecture and implementation. In the foundation of the application architecture lays the three-tier approach, which is essential to understanding of the rest of the document. Below is the theoretical discussion of multi-tier approach to the application architecture and advantages of three-tier architecture over other alternatives.

Any three-tier applications can be understood in terms of three different functional components: data management, application logic, and presentation. Different components are responsible for different purposes, namely, the data management component is responsible for storing and managing all the persistent data used in the application. The presentation layer is responsible to presenting the application data and functionality to the user and collecting the user's input to the application. The application logic component is the heaviest among all three. It implements the business logic of the whole application as well as addressing multiple functional and non-functional requirements by providing various services like caching, distributing computing, distributing transactions, data transformation, authentication, and many others.

Single-Tier Architecture

Before we start a detailed discussion of the three-tier application architecture, it makes sense to introduce single-tier and client-server architectures, which are the predecessors of the three-tier architecture. This approach will illustrate the advantages of three-tier architecture more distinctly.

Initially, data-intensive applications were combined into a single tier, including the data storage/retrieving application logic, and user interface, as illustrated in Figure 1. The application typically ran on a single powerful server (i.e. a mainframe), and users accessed it through thin clients (i.e. dumb terminals) that could perform only data input and display. This approach has

the benefit of being centralized on a single computer simplifying application deployment, maintenance, and upgrading.

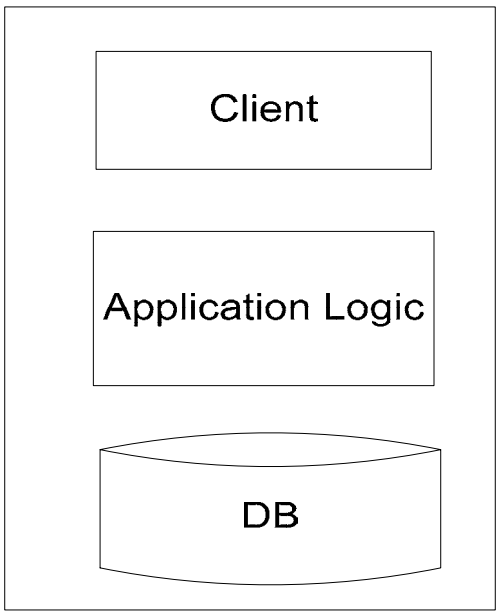

Figure1: Single-Tier Architecture

There are many disadvantages to using the single-tier application. This architecture limits the extent of complexity of the graphical user interface, because of the limited resources on the dummy terminals and limited computing power that can be allocated to the presentational layer on the application server. Therefore, the industry demand for more sophisticated graphical user interface and availability of relatively cheap computing powers on the clients by introduction of PC's, pushed for the change in general architectural approach by making the clients heavier and more application aware. On the server side, centralized application has a disadvantage of being a single point of failure compromising application reliability and robustness. This fact demanded for distribution of server functionality on multiple computers, therefore increasing the performance, reliability, and cost of server computing powers with the introduction of features like failing-over and load balancing.

Client-Server Architecture

Realizing the limitations of single-tier application architecture, the industry moved to more sophisticated scheme of two-tier application, which is commonly referred to as client-server architecture, as illustrated in Figure 2. This approach solves the graphical user interface complexity requirements by making the presentation layer in its own application-aware architectural component. However, this approach loosely defines how application logic should be distributed between the client and the server. In its traditional form, the client is wholly

responsible for the presentation layer (and, therefore is often referred to as a thin client), and the server is responsible for the rest of the application logic. However, this approach still inherits some of the problems of single-tier approach, namely, the problem of the server being a single-point of failure and the application server also being responsible for managing the application persistent data.

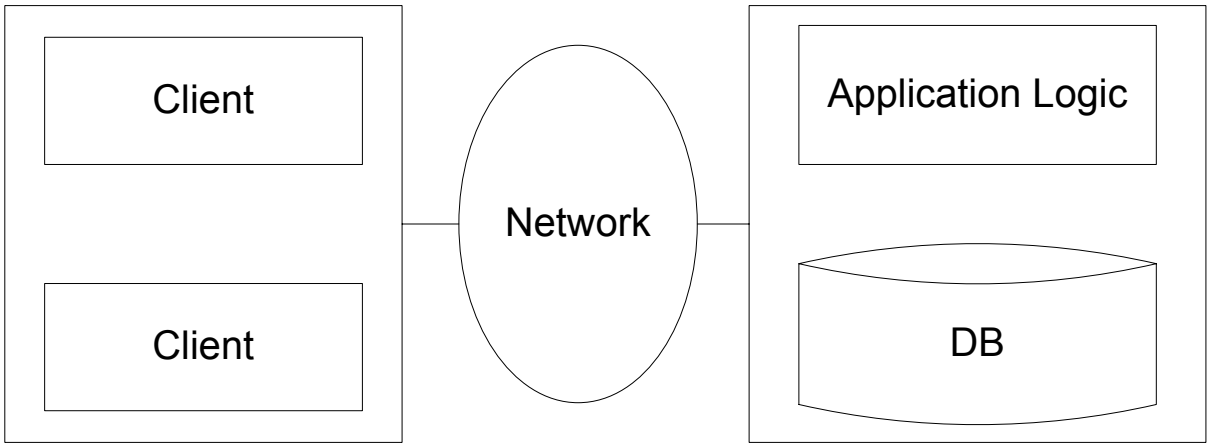

Figure2: Two-Server Architecture: Thin Clients

Other distribution schemes between client and server are possible, such as more powerful clients that implement both user interface and business logic, or clients that implement user interface and part of the business logic, with the remaining part being implemented on the server. The clients that also carry some of the application logic of the application are called thick clients**.** This architectural approach is illustrated in Figure 3.

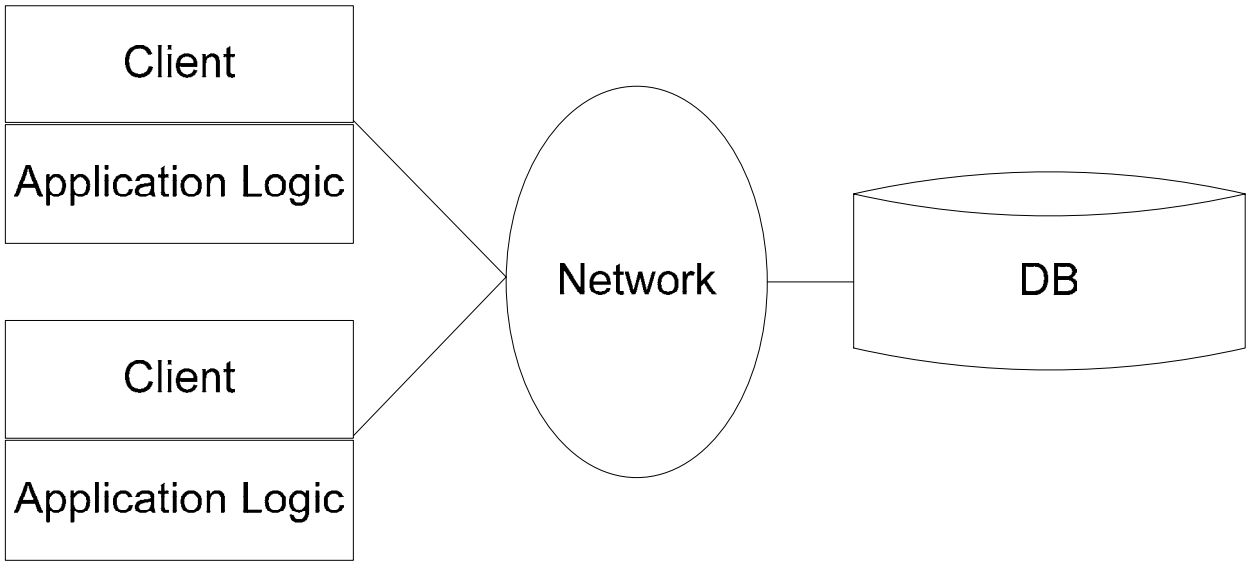

Figure3: Two-Server Architecture: Thick Clients

Compared to the single-tier architecture, two-tier architectures physically separate the user interface from the data management layer. To implement two-tier architectures, we can no longer have dumb terminals on the client side; we require computers that run sophisticated presentation code (and possibly, application logic).

In the resent years, many new technologies have been developed that made distributed computing possible for the commercial applications. Such progress in the software technologies heavily contributed to the success of client-server architectural approach over all other architectural alternatives, also pushing for thinner clients in general.

The thick-client model has several disadvantages when compared to the thin-client model. First, there is no central place to update and maintain the business logic, since the application code runs at many client sites. This introduces a possibility of application version mismatch between a client and the server causing apparent or hidden run-time application errors (the latter ones being the hardest to identify). Second, a large amount of trust is required between the server and the clients. This violates business logic encapsulation paradigm making it hard to introduce a change to the application and increases difficulties of Quality Assurance, since the business logic spread over multiple sites require more sophisticated testing schemes.

A third disadvantage of the thick-client architecture is that it does not scale with the number of clients due to the lack of communication among clients and, therefore, absence of any reusability in data and functionality. As a result, each client ends up holding all the data required for the current state of the application and is wholly responsible for retrieving and updating the application data.

Finally, thick-client systems run the danger of quickly exhausting the server resources. The absence of communication among the clients makes it extremely difficult to use a scarce server resource in the application.

Three-Tier Architecture

With a time as the limitations of single-tier and client-server architectural approaches became apparent, the industry moved toward three-tier application architecture separating applications into presentational layer, data services layer, and a business logic layer (middle tier), as illustrated in Figure 4.

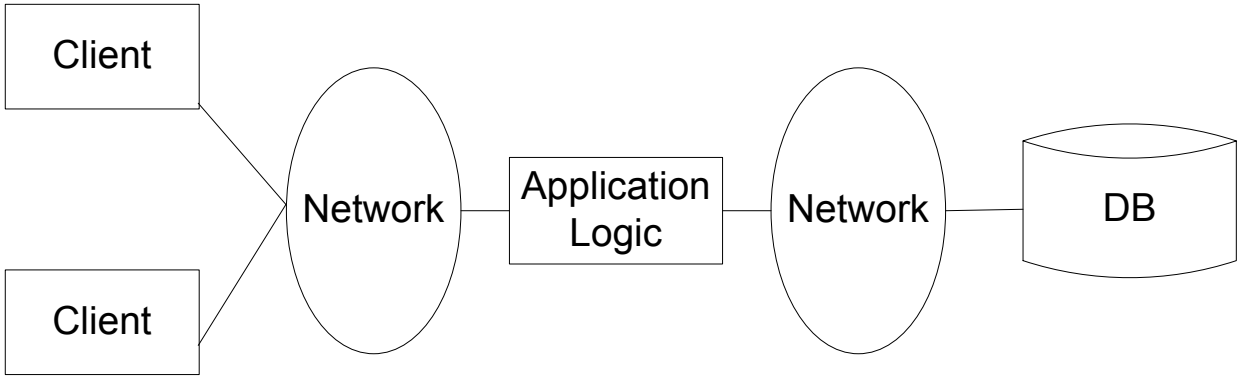

Figure 4: Standard Three-Tier Architecture

Presentation Tier

Presentation tier is responsible for all kinds of user interaction, namely, gathering information from the user, sending the user information to the business services for processing, receiving the results of the business services processing, and presenting those results to the user.

One of the most popular choices for the presentation tier nowadays is a Web-based client. It may be as simple as a collection of static HTML pages retrieved from the web server as is, or may be as complex as server-generated dynamic pages that utilize advanced web browser features like DHTML and browser scripting. Many technologies have been implemented to facilitate we-based application development, like Active Server Pages (ASP), Server-Side Includes (SSI), Java Server Pages (JSP), Java Beans (EJB), and many others. On the client side, VB Script or JavaScript are used to address presentation complexities of the graphical user interface (GUI).

There are many considerations to be made concerning the presentation tier. Since the presentation tier is least controlled from the portability point of view, many operating systems, devices, and display properties need to be supported. Assuming a standard web browser environment often simplifies this task making web-based presentation layer a very popular choice. Without a standard client environment, the adaptivity could be achieved only at the expense of complexity on the middle tier that would generate content based on the type of clients.

Middle Tier

The middle tier is the thickest application component that contains business logic of the application, provides various application services like caching, distributing computing, distributing transactions, data transformation, and authentication, and serves as a coordinator of other application components. In particular, it is responsible for receiving input from the presentation tier, interacting with the data services to retrieve or update application persistent data, and sending the processed results to the presentation tier potentially merging the dynamic content with the static pages.

The most important aspect of the middle tier is that it implements the business logic of the application: it controls (validates) the input data from the presentation tier before an action can be executed, determines the control flow between multi-action steps, controls access to the database layer, enforces many application business rules and overall application workflow, and often assembles dynamically generated pages.

Often the middle tier code is responsible for supporting all the multiple roles involved in the application making the application behave differently for different users. Many applications require at least two types of users, namely regular users of the application and super users or administrators. Each of these roles can require support for several complex actions. This requires middle tier to manage user's sessions and credentials throughout all the actions on the middle tier.

In addition, many standard services can be added to the middle tier. Caching service allows to store recently used persistent data in middle-tier memory, which potentially minimizes time spent on repeated data retrieves and reduces the load on the database services tier. Distributing computing allows faster computation by splitting a complex computational task into multiple subtasks and executing each of them in parallel on multiple machines. It also allows running duplicate middle tier services on multiple machines making failing-over and load-balancing possible, which increases the overall robustness of the middle tier. Distributed transaction service allows certain actions to be executed across multiple data stores guaranteeing a success or a failure of the transaction consistently across all transaction participants. These guarantees are essential when dealing with sensitive information. Data transformation service may also be used to transform the data between multiple formats like HTML, XML, Style Sheets, many binary formats, and others. Authentication service may be used to check and track users' credentials in the application and managing users' rights and privileges to view or modify certain application data as well as executing certain application services. The middle tier may also coordinate the use of scarce server resource managing access to such resources among users. It may also manage communication among various application components via messaging services. Finally, it manages itself in terms of memory and CPU utilization requirements.

Database Services Tier

The database services tier is responsible for storing, updating and retrieving the persistent data. A relational database is most often used as a database persistence tier; however, other types of databases are possible, namely, object-oriented databases, multi-dimensional databases, etc. A database server serves remote requests that come in a form of a query language (SQL in case of a relational database), and returns a record set or a cursor that contains a result of the submitted query. In case of a record set, the result of the query is returned to the requester's memory space reducing the use of database server's resources. In case of a cursor, on the other hand, the query result is held in the database server's memory space pointed by a remote reference returned to the requester. The latter case puts heavier load on the database server. In case of an object-oriented database, however, a collection of objects is returned to the requester.

In case of a relational database, the query may be submitted by a requester or stored with the database server as a stored procedure. The latter case is more efficient, since it allows the database server to pre-compile the execution plan for a query ahead of time. It is also beneficial from the architectural perspective, because, in this case, the stored procedures serve as a published API between a persistence tier and a middle tier.

Advantages of the Three-Tier Architecture

The three-tier architecture has many advantages over other architectural choices listed above (single-tier and client-server). In particular, different tiers can utilize the strengths of different platforms and different software components. This makes it easy to modify or replace the code at any tier without affecting the other tiers. Usually, three-tier architecture implies thin clients that were justified earlier. Three-tier application also benefits from centralized usage of scarce server resources, in particular database systems. Three-tier architecture is also much more scalable to many clients. By utilizing caching, distributed computing, and centralized resource management, many more clients can use middle tier services without running risk of resource starvation, a single resource failure, single machine limitations, or data redundancy per client. In addition, features like load-balancing only enhance the scalability of three-tier architecture.

There are also many software development advantages to three-tier architecture. By dividing the application cleanly into parts that address presentation, data access, and business logic, we gain greater application structure, components decoupling, and ease of testing, maintenance, upgrade, and enhancement. Interaction between tiers occurs through well-defined, standardized APIs. Therefore, each application tier can be built out of reusable components that can be individually developed, debugged, and tested.

Having said many theoretical considerations concerning three-tier application, we are ready to dive into the particular architectural description of Final Audit system. The project report consists of two parts. The first part describes gathered requirements and system design and architecture; the second part describes the implementation details of the project.

Purpose

The purpose of the project is to develop a web application that will allow students to submit applications for graduation clearance and check the status of the application online, as well as to facilitate the Registrar's Office staff processing of the applications.

Scope

The application will support three types of users, namely, an administrator, a staff member, and a student. From the student's perspective, application will give the users ability to submit final audit application, check the status of their applications, and check their eligibility for graduation. From the staff member's perspective, it will allow to view graduation status of the students as well as maintaining class equivalence acceptable for substitution by the graduation requirements. Finally, from the administrator's perspective, it will also allow to maintain system core parameters like a list of required courses or number of required credits for graduation.

Requirements, Design, and Architecture

Functional Requirements

General

1. All users of the system should have authorization credentials (username and password).
2. The system should support three types of users (administrator, staff member, and a student) and maintain a different set of available screens (privileges) for each type of users.
3. All system edits should be audit-trailed.

Students

1. Ability to review graduation requirements and general information about graduation procedures on-line.
2. Ability to view the graduation status.
    a. Completed number of credits per category of courses (i.e. core courses, elective courses, in-progress courses, etc.)
    b. Number of credits required for graduation per each category of courses
    c. Overall graduation status
    d. Remaining credits required for graduation per each category of courses.
3. Ability to fill out and submit an application for graduation.
4. Ability to view the status of submitted application.
5. Ability to submit a request for course substitution.

Staff Member

1. Ability to view the graduation status for any selected student.
    a. Completed number of credits per category of courses (i.e. core courses, elective courses, in-progress courses, etc.)
    b. Number of credits required for graduation per each category of courses
    c. Overall graduation status
    d. Remaining credits required for graduation per each category of courses.
2. Ability to view the status of submitted application for any selected student.
3. Ability to approve/deny a course substitution requested by a student.
4. Clear a student for graduation.

Administrator

1. Ability to edit graduation requirements (number of credits) per course category.
2. Ability to edit courses and their categories.
3. Ability to review audit-trails.

Non-Functional Requirements

1. The system should support about 5,000 registered users.
2. The system should support 50-100 concurrent users at any given time.
3. The system should be available 24/7 with some downtime for maintenance once a week.
4. The system should run correctly on the latest versions of MS Internet Explorer, Netscape, and other Mozilla-based browsers.

Systems Requirements

The system is written assuming the following hardware and software environment (requirements):

Server(s)

1. One Windows 2000 Server (or later) machine with at least 2 GB of RAM and 100 MB of available free disk space for the database and code installation.
2. IIS 5.0 (or later) service.
3. Microsoft .NET Framework 1.1.
4. Microsoft SQL Server 2000 Standard/Enterprise Edition (on the same or different server machine).

Client(s)

1. A latest version of any of the following browsers:
    a. Microsoft Internet Explorer
    b. Netscape
    c. FireFox
    d. Mozilla-based browsers
2. The browser must support HTML frames, JavaScript 1.0, and cookies.

Deployment Requirements

Initial installation

The initial installation will be handled by means of Microsoft Installation Package which will contain a software installation as well as database schema creation script. The installation will expect that the database is created prior to the instillation.

On-going changes

The ongoing changes will be pushed to the server side of the system during its once a week downtimes by replacing affected files with newer versions. The system upgrades are limited to the changes to the server side – no client changes will be necessary.

Support Requirements

Application Administration

The system will require one part-time application administrator to maintain accuracy of the system data, namely, the list or courses and the graduation requirements in terms of required number of credits for graduation per category of courses. It is expected that the administrator will be required to spend some time populating the system initially (one time effort). The time spent on the on-going updates will depend on the frequency and the amount of changes to the program and cannot be assessed at this time.

System Design

Interface Design

The application will only have web-based interface for all the users. All interface pages will have Montclair State University logo in the upper-left corner. All the pages will be classified into two sections, namely, *Home* and *Final Audit*. The *Home* section will host Montclair State University Home page allowing students to browse existing Montclair State University without leaving the application. The *Final Audit* section will contain all the functionality of the Final Audit application.

The *Final Audit* section will be further subdivided into submenus. Initially, only two choices will be displayed:

1. **Graduation Information** – a menu that will display the current graduation requirement information. This information will not require a login in order to be viewed.
2. **Login** – a menu that will allow users to log in to the application.

Upon logging in, the users will be presented with a set of menus based on the type of the logged in user. All the users will still see the *Graduation Information* menus as well as the following additional menus:

Administrator

1. **Edit Graduation Requirements** – a menu that will allow editing credit requirements for graduation per course category.
2. **Edit Courses** – a menu that will allow editing offered courses.
3. **View Audit Trails** – a menu that will allow viewing audit trails in the application.

Staff Member

1. **Current Submissions** – a menu that will display student's fulfilled requirements and student's status for graduation, as well as the student's application for graduation submitted earlier. This menu will also allow clearing or declining the student's application for graduation.
2. **Archived Submissions** – a menu that will display student's fulfilled requirements and student's status for graduation for the students that were cleared or declined.

Student

1. **Summary** -- a menu that will display student's fulfilled requirements and student's status for graduation.
2. **Fill out Application** – a menu that will allow filling out and submitting an application for graduation.
3. **Application Status** -- a menu that will display status of the student's application for graduation submitted earlier.

The *Summary* page will list all the categories of course as clickable links. When clicked, the list of taken courses under the given category taken by the current student will be displayed. For the courses that are not counted toward graduation requirements and do not have substitute courses, the third column (named *Substitute*) will have an entry field and a submit button allowing user to request a course substitution.

Once logged in, all the users will also be given with *Log Out* submenu allowing the users to terminate their current session.

Database Design

Table Listing:

MajorDescription: Description of the Major
Major: Category of the course for the major
MajorRequirement: Number of core and elective credits for the Major
Course_Credit: Number of credits for the course
Student: Student General Information Table
StudentProgress: Description what Courses Student passed or failed
Application: Table where Final Audit Application is stored
CourseSubstitute: Known information about substituted courses
UserData: User Information Table

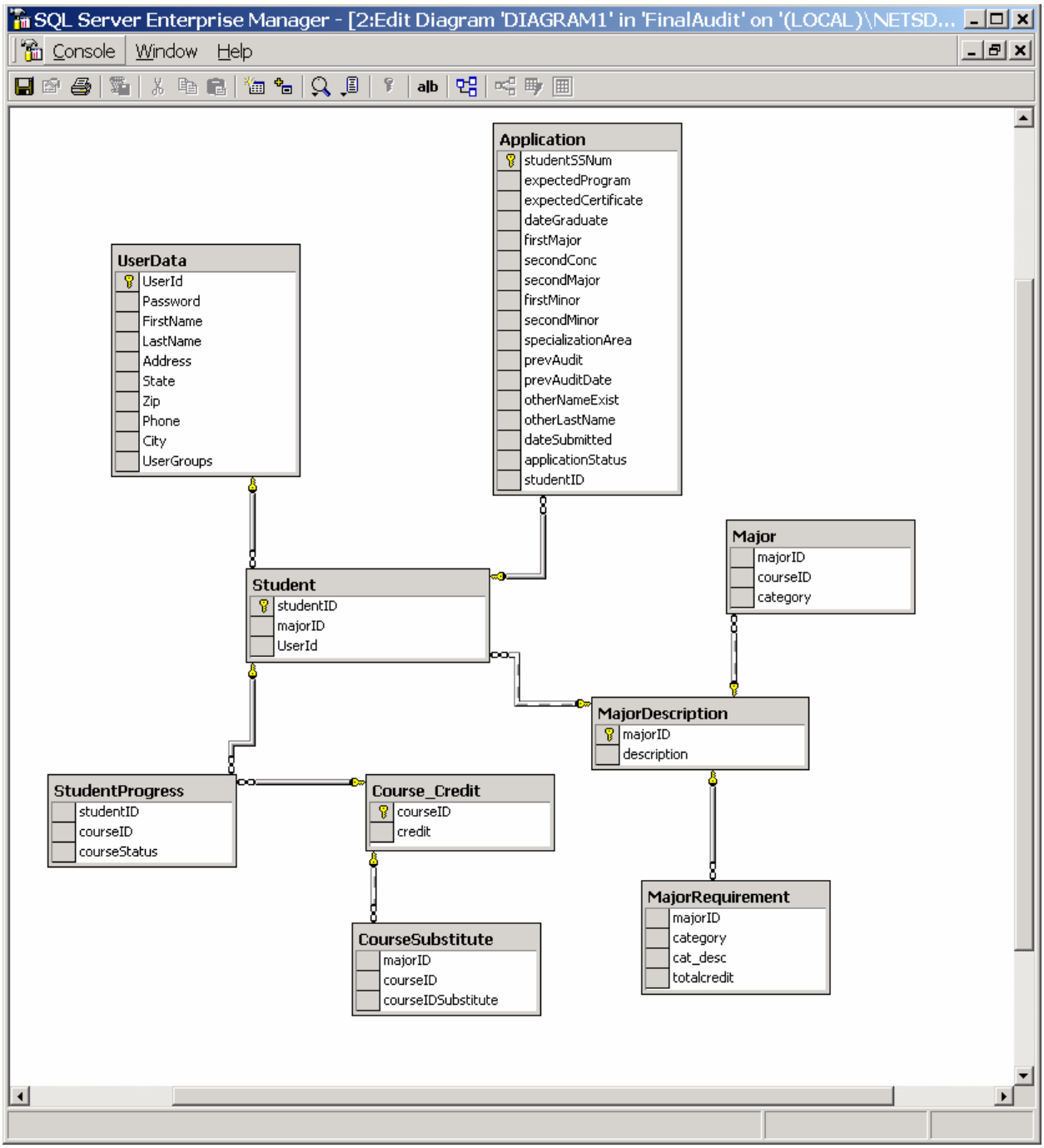
SQL Server Enterprise Manager - [2:Edit Diagram 'DIAGRAM1' in 'FinalAudit' on '(LOCAL)\NETSD...
Console
Window
Help
Application
studentSSNum
expectedProgram
expectedCertificate
dateGraduate
firstMajor
secondConc
secondMajor
firstMinor
secondMinor
specializationArea
prevAudit
prevAuditDate
otherNameExist
otherLastName
dateSubmitted
applicationStatus
studentID
UserData
UserId
Password
FirstName
LastName
Address
State
Zip
Phone
City
UserGroups
Major
majorID
courseID
category
Student
studentID
majorID
UserId
MajorDescription
majorID
description
StudentProgress
studentID
courseID
courseStatus
Course_Credit
courseID
credit
MajorRequirement
majorID
category
cat_desc
totalcredit
CourseSubstitute
majorID
courseID
courseIDSubstitute

Table Name: MajorDescription
Description: Description of the Major

No	**Field Name**	**Field Type**	**Primary Key Sequence**	**Length**	**Null**	**Description**
1	majorID	varchar	1	10	N	Major code
2	Description	varchar		50		Major description

Example:

majorID	Description
MATH	Mathematics
CPSC	Computer Science
STAT	Statisctics

Table Name: Major
Description: Category of the course for the major

No	Field Name	Field Type	Primary Key Sequence	Length	Null	Description
1	majorID	varchar	1	10	N	Major code
2	courseID	varchar	2	10	N	Course code
3	category	char		2		Course Category: Core or elective

Example:

majorID	courseID	category
CPSC	CMPT 699	E1
CPSC	CMPT 581	R1
MATH	MATH 404	R1
STAT	STAT 440	R1

Table Name: MajorRequirement
Description: Number of core and elective credits for the Major

No	Field Name	Field Type	Primary Key Sequence	Length	Null	Description
1	majorID	varchar	1	10	N	Major code
2	category	char	2	2	N	Course Category: Core or elective
3	cat_desc	varchar		100		Description of the category
4	totalcredit	int				Number of credits required by major for each category

Example:

CPSC	R1	Required Core Courses	27
CPSC	E1	Comp Science, Math and/or Stat Electives	12

Table Name: Course_Credit
Description: Number of credits for the course

No	Field Name	Field Type	Primary Key Sequence	Length	Null	Description
1	courseID	varchar	1	10	N	course ID
2	credit	int				Number of credits for the course

Example:

CMPT 699	3
CMPT 585	3

Table Name: Student
Description: Student General Information Table

No	Field Name	Field Type	Primary Key Sequence	Length	Null	Description
1	studentID	varchar	1	10	N	student ID
2	majorID	varchar	2	10	N	Major code
3	UserId	varchar		20		Student UserID to login into Application

Example:

111111111	CPSC	levinm1
111111112	CPSC	paninj1

Table Name: StudentProgress
Description: Description what Courses Student passed or failed

No	Field Name	Field Type	Primary Key Sequence	Length	Null	Description
1	studentID	varchar	1	10	N	student ID
2	courseID	varchar	2	10	N	Course ID
3	courseStatus	varchar		1	N	Status of the course: Completed-Pass, In Progress, Fail, NC

Example:

111111111	CMPT 585	P
111111112	CMPT 585	F

Table Name: Application
Description: Table where Final Audit Application is stored

No	Field Name	Field Type	Primary Key Sequence	Length	Null	Description
1	studentSSNum	varchar	1	10	N	student SS number
2	expectedProgram	varchar		50	N	Expected Program Completition:MS,BS,ect.
3	expectedCertificate	varchar		50		certificate you expect to receive
4	dateGraduate	datetime			N	Date of expected graduation
5	firstMajor	varchar		50	N	Major you completed
6	secondConc	varchar		50		Second Concentration you completed
7	secondMajor	varchar		50		Second Major you completed
8	specializationArea	varchar		50		Specialization Area
9	prevAudit	varchar		2		Previous Audit: Yes or No
10	prevAuditDate	varchar		50		Previous Audit Date
11	otherNameExist	varchar		50		Other names used:Yes or No
12	otherLastName	varchar		50		Other names used
13	dateSubmitted	datetime	2		N	Date when application was submitted
14	applicationStatus	varchar		50	N	Status of the application: Submitted, In Progress, ect.
15	studentID	varchar		10	N	Student ID

Table Name: CourseSubstitute
Description: Known information about substituted courses

No	Field Name	Field Type	Primary Key Sequence	Length	Null	Description
1	majorID	varchar	1	10	N	Major ID
2	courseID	varchar	2	10	N	Course ID
3	courseIDSubstitute	varchar	3	10	N	Course ID substitute

Example:

CPSC	STAT 570	MATH 560

Table Name: UserData
Description: User Information Table

No	Field Name	Field Type	Primary Key Sequence	Length	Null	Description
1	UserId	varchar	1	20	N	User ID
2	Password	varchar	2	15	N	password
3	FirstName	varchar	3	50	N	First Name of the User
4	LastName	varchar	4	50		Last Name of the User
5	Address	varchar		50		Address
6	State	varchar		2		State
7	Zip	varchar		5		Zip
8	Phone	varchar		10		Phone
9	City	varchar		50		City
10	UserGroups	varchar		50		Name of the User Group: Student, Staff

Example:

levinm1	1111	Mike	Levin	First st.	NJ	07661	201-973-1111	Montclair	STUDENT

Server Component Architecture

The system will consist of three tiers, namely, the front end consisting of web-based thin client implemented in HTML and generated JavaScript, the MS SQL database serving as a back end, and the C# middle tier. The diagrams below depict the system architecture. The system diagram (below) shows all three tiers and the way they interact among themselves. The data flow diagram describes the data exchange among the tiers, and the class diagram depicts the class architecture on the middle tier.

System diagram

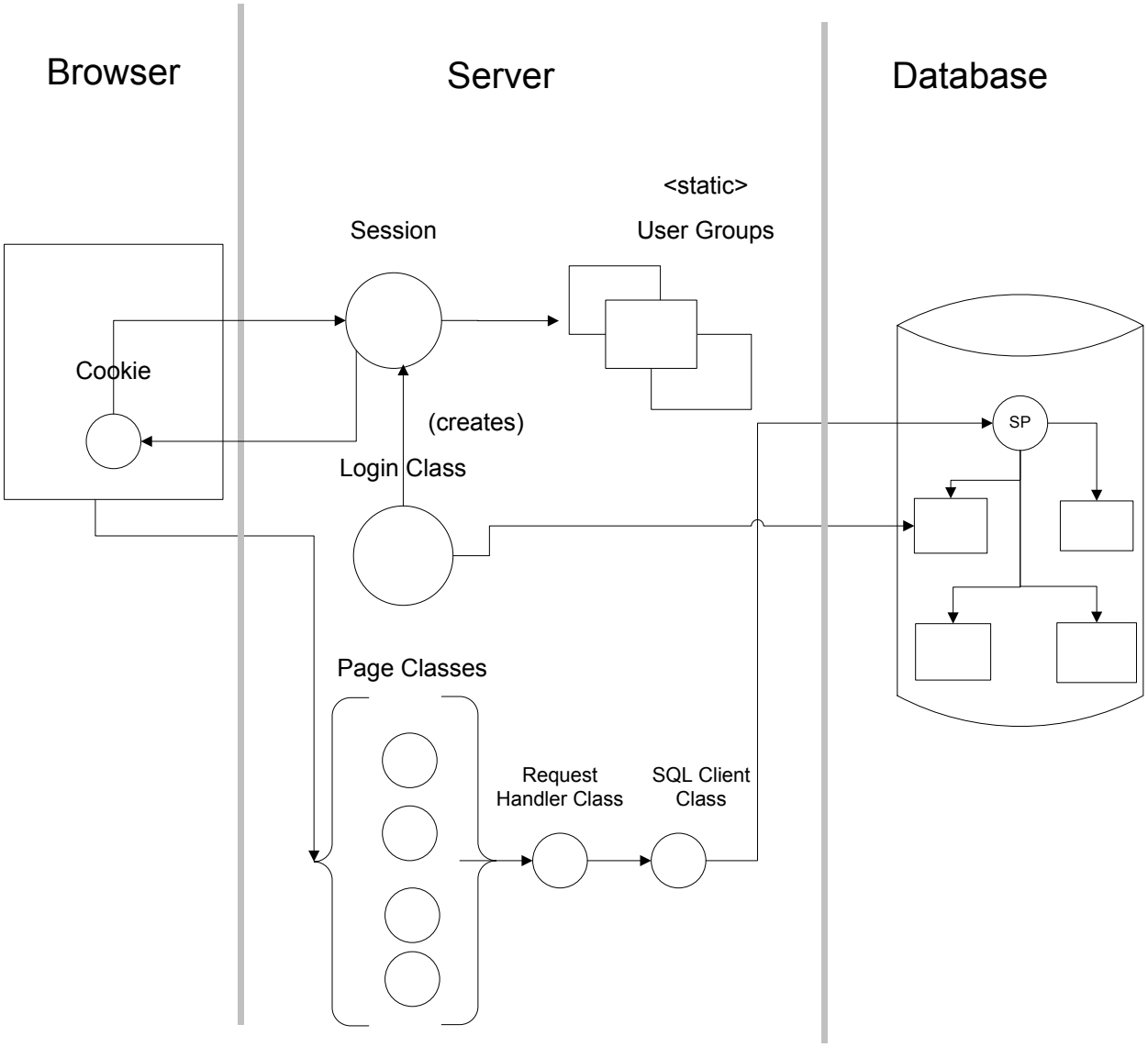

Figure 5: System Diagram

The system diagram above shows the browser client, server middle tier, and database back-end. The only state being kept on the client is cookies needed for session implementation. The session cookies are automatically generated by the web server and transparently handled by the browser. This mechanism allows to have a transparent authentication support for the users of the system. Each session gets associated with one of the static user groups, which determines the look and functionality of the application for that particular user. Once a user is authenticated by the web server, his/her request gets handed into the appropriate page class written in C#. All page classes use the same Request Handler class, which contains common components across all the pages. In particular, the Request Handler uses SQL Client class for all the database communication. On the database level, all the requests from the middle tier are coming through appropriate stored procedure, which controls (and optimizes) the uses of underlying tables.

Data flow diagram

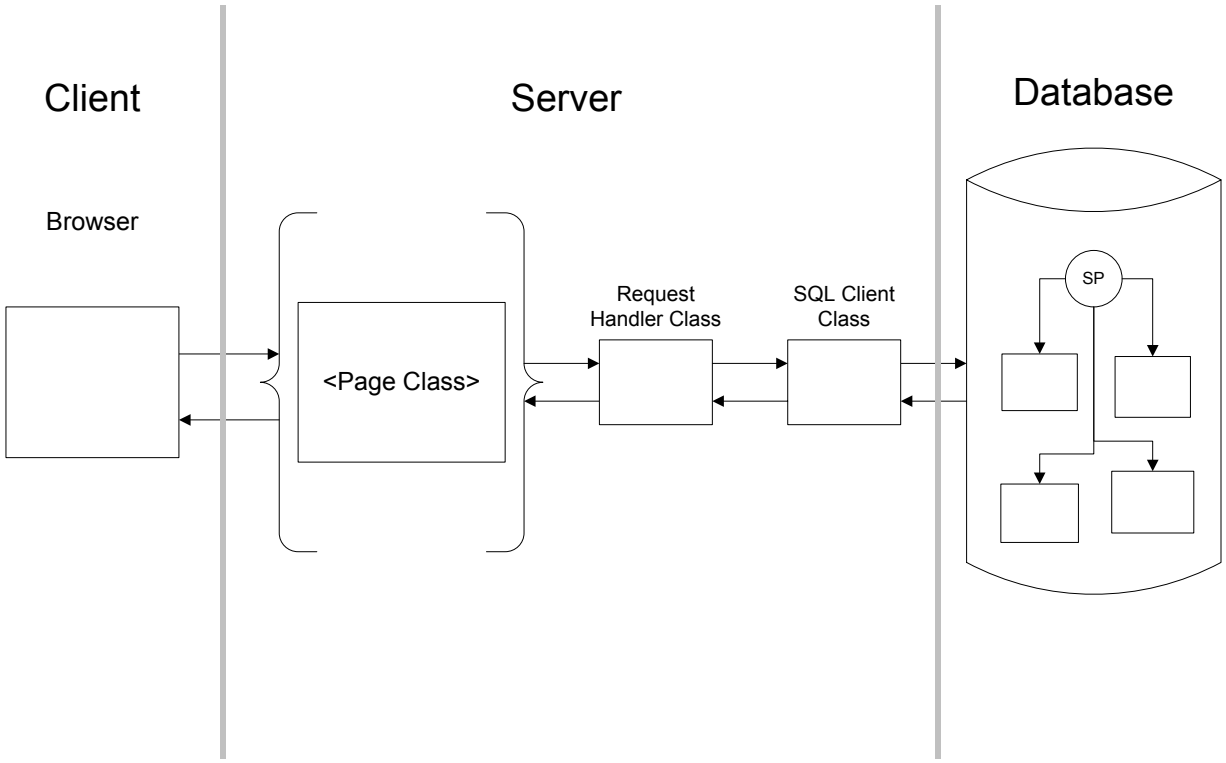

Figure 6: Data Flow Diagram

The data flow diagram outlines the data exchange within the system. The request is always initiated on the client, and gets directed to the middle tier where it get processed according to the user's privileges and current business rules. The ultimately the appropriate information gets retrieves from the database via stored procedures and returned back to the browser merged with the HTML and JavaScript in ready-to-be-displayed form.

Class diagram

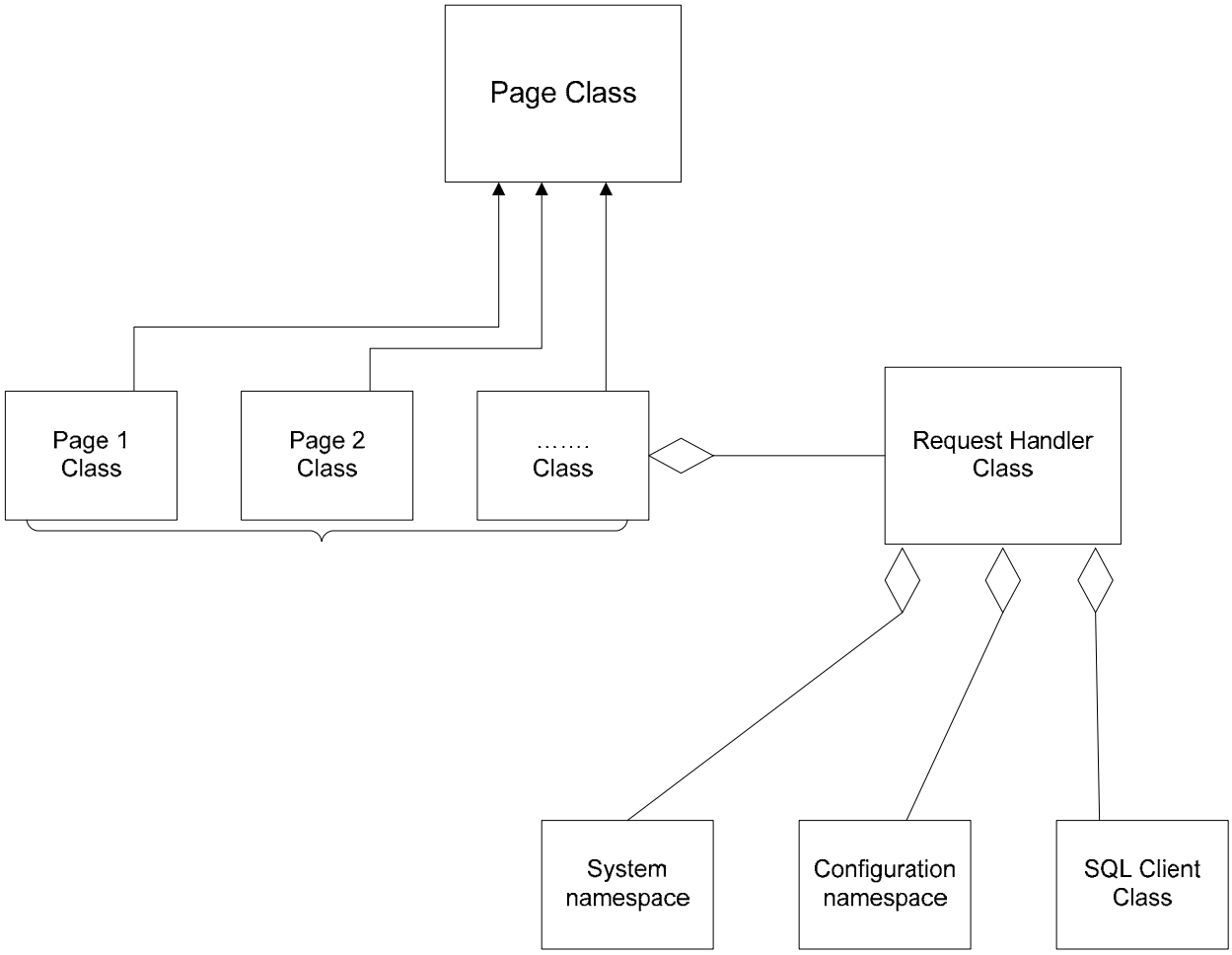

Figure 7: Class Diagram

The class diagram describes the class organization on the middle tier. Each page within the application is generated and handled by the appropriate class on the middle tier. All the page classes inherit from the Page class, which implements the common API for a page class. The page classes also reuse a singleton Request Handler class, which provides facilities reused across multiple pages. In particular, the Request Handler class uses standard Microsoft .NET Framework classes to interact with the container environment and communicate to the database.

Implementation Details and Deployment

Developer's Notes

It has been decided not to have any SQL statements coming to the database from the middle tier code. Instead, all the code should call a stored procedure. This implementation decision logically separates the C# code from the database table structure and establishes a database API for any code outside of the database. It also allows the database server to optimize (compile) the stored procedure increasing the performance of the database trips, prevents run-time errors coming from the database, and allows future database alterations without affecting other components of the system.

All the page classes are separated into two parts – one contains all the HTML code and presentational details, whereas the other part contains code which implements the business logic behind the page. This separation increases the readability and maintainability of the application simplifying future enhancements.

It has also been decided not to have any hard-coded state within the code in middle tier. All the state should come from either a database or a configuration file. This allows enforcing generality of the code making it readily available for a similar use with a different configuration or a database state.

It has been decided to use form-based authentication mechanism instead of Windows-based authentication. Form-based authentication is fully implemented within HTTP protocol and does not require any special support from the browser (except support for cookies), which makes it browser and operating system independent.

Database Schema Creation Script

The following were the scripts used to create the tables for the database.

Table: MajorDescription

```
CREATE TABLE [dbo].[MajorDescription] (
        [majorID] [varchar] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
        [description] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
) ON [PRIMARY]
GO
```

Table Name: Major

```
CREATE TABLE [dbo].[Major] (
        [majorID] [varchar] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
        [courseID] [varchar] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
        [category] [varchar] (2) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
) ON [PRIMARY]
GO
```

Table: MajorRequirement

```
CREATE TABLE [dbo].[MajorRequirement] (
        [majorID] [varchar] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
        [category] [varchar] (2) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
        [cat_desc] [varchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
        [totalcredit] [int] NULL
) ON [PRIMARY]
GO
```

Table: Course_Credit

```
CREATE TABLE [dbo].[Course_Credit] (
        [courseID] [varchar] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
        [credit] [int] NULL
) ON [PRIMARY]
GO
```

Table: Student

```
CREATE TABLE [dbo].[Student] (
        [studentID] [varchar] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
        [majorID] [varchar] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
        [UserId] [varchar] (20) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
) ON [PRIMARY]
GO
```

Table: StudentProgress
```
CREATE TABLE [dbo].[StudentProgress] (
        [studentID] [varchar] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
        [courseID] [varchar] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
        [courseStatus] [varchar] (1) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
) ON [PRIMARY]
GO
```

Table: Application
```
CREATE TABLE [dbo].[Application] (
    [studentSSNum] [varchar] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
    [expectedProgram] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [expectedCertificate] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [dateGraduate] [datetime] NULL,
    [firstMajor] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [secondConc] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [secondMajor] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [firstMinor] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [secondMinor] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [specializationArea] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [prevAudit] [varchar] (2) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [prevAuditDate] [datetime] NULL,
    [otherNameExist] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [otherLastName] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [dateSubmitted] [datetime] NULL,
    [applicationStatus] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [studentID] [varchar] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
) ON [PRIMARY]
GO
```

Table: CourseSubstitute
```
CREATE TABLE [dbo].[CourseSubstitute] (
        [majorID] [varchar] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
        [courseID] [varchar] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
        [courseIDSubstitute] [varchar] (10) COLLATE SQL_Latin1_General_CP1_CI_AS
NULL
) ON [PRIMARY]
GO
```

Table Name: UserData
```
CREATE TABLE [dbo].[UserData] (
        [UserId] [varchar] (20) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
```

```
	[Password] [varchar] (15) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
	[FirstName] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
	[LastName] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
	[Address] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
	[State] [varchar] (2) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
	[Zip] [varchar] (5) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
	[Phone] [varchar] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
	[City] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
	[UserGroups] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
) ON [PRIMARY]
GO
```

List of Store Procedures and description

Name: sp_CoreCoursesDetails
Description: Display Names of the Core Courses student took

```
CREATE    proc sp_CoreCoursesDetails
	@UserId nvarchar(20)
as
SELECT
StudentProgress.courseID,Course_Credit.credit,ISNULL(CourseSubstitute.courseIDSubstitute,'
N/A') as courseIDSubstitute
  FROM        Student
 INNER JOIN StudentProgress ON Student.studentID = StudentProgress.studentID
 INNER JOIN Course_Credit ON StudentProgress.courseID = Course_Credit.courseID
Left JOIN CourseSubstitute ON StudentProgress.courseID = CourseSubstitute.courseID
 INNER JOIN Major ON Student.majorID = Major.majorID AND StudentProgress.courseID =
Major.courseID
 INNER JOIN UserData ON Student.UserID = UserData.UserID
WHERE     (StudentProgress.courseStatus NOT IN ('R', 'F', 'N'))
AND     (Student.UserId = @UserId  AND (left(Major.category,1) = 'R'))
GO
```

Name: sp sp_CoreCoursesTotalCredits
Description: Student's total credits from the core courses

```
CREATE  proc sp_CoreCoursesTotalCredits
	@UserId nvarchar(20)
as
```

```
SELECT     SUM(Course_Credit.credit) AS TotCredits
  FROM         Student
 INNER JOIN StudentProgress ON Student.studentID = StudentProgress.studentID
 INNER JOIN Course_Credit ON StudentProgress.courseID = Course_Credit.courseID
 INNER JOIN Major ON Student.majorID = Major.majorID AND StudentProgress.courseID =
Major.courseID
 INNER JOIN UserData ON Student.UserID = UserData.UserID
WHERE     (StudentProgress.courseStatus NOT IN ('R', 'F', 'N'))
AND     (Student.UserId = @UserId  AND (left(Major.category,1) = 'R'))
GROUP BY Student.studentID, Major.category, StudentProgress.courseStatus
GO
```

Name: sp_CoreRqrCoursesCredits
Description: Number of core credits required for graduation

```
CREATE   proc sp_CoreRqrCoursesCredits
         @UserId nvarchar(20)
as

SELECT     totalcredit
  FROM      MajorRequirement
 INNER JOIN Student ON Student.majorID = MajorRequirement.majorID
 INNER JOIN UserData ON Student.UserID = UserData.UserID
WHERE     (Student.UserId = @UserId  AND (left(MajorRequirement.category,1) = 'R'))
GO
```

Name: sp_ElectiveCoursesDetails
Description: List Names of the Elective Courses student took

```
CREATE     proc sp_ElectiveCoursesDetails
         @UserId nvarchar(20)
as

SELECT
StudentProgress.courseID,Course_Credit.credit,ISNULL(CourseSubstitute.courseIDSubstitute,'
N/A') as courseIDSubstitute
  FROM         Student
 INNER JOIN StudentProgress ON Student.studentID = StudentProgress.studentID
 INNER JOIN Course_Credit ON StudentProgress.courseID = Course_Credit.courseID
Left JOIN CourseSubstitute ON StudentProgress.courseID = CourseSubstitute.courseID
 INNER JOIN Major ON Student.majorID = Major.majorID AND StudentProgress.courseID =
Major.courseID
 INNER JOIN UserData ON Student.UserID = UserData.UserID
WHERE     (StudentProgress.courseStatus NOT IN ('R', 'F', 'N'))
```

```
AND     (Student.UserId = @UserId  AND (left(Major.category,1) ='E'))
GO
```

Name: sp_ElectiveCoursesTotalCredits
Description: Total Elective credits student have

```
CREATE  proc sp_ElectiveCoursesTotalCredits
        @UserId nvarchar(20)
as

SELECT     SUM(Course_Credit.credit) AS TotCredits
  FROM        Student
 INNER JOIN StudentProgress ON Student.studentID = StudentProgress.studentID
 INNER JOIN Course_Credit ON StudentProgress.courseID = Course_Credit.courseID
 INNER JOIN Major ON Student.majorID = Major.majorID AND StudentProgress.courseID =
Major.courseID
 INNER JOIN UserData ON Student.UserID = UserData.UserID
WHERE     (StudentProgress.courseStatus NOT IN ('R', 'F', 'N'))
AND     (Student.UserId = @UserId  AND (left(Major.category,1) ='E'))
GROUP BY Student.studentID, Major.category, StudentProgress.courseStatus
GO
```

Name: sp_ElectiveRqrCoursesCredits
Description: Number Elective credits required for the major

```
CREATE   proc sp_ElectiveRqrCoursesCredits
        @UserId nvarchar(20)
as

SELECT     totalcredit
  FROM      MajorRequirement
 INNER JOIN Student ON Student.majorID = MajorRequirement.majorID
 INNER JOIN UserData ON Student.UserID = UserData.UserID
WHERE     (Student.UserId = @UserId  AND (left(MajorRequirement.category,1) = 'E'))
GO
```

Name: sp_GetApplStatus
Description: Get status of submitted final audit application

```
CREATE  PROCEDURE [dbo].[sp_GetApplStatus]
      @UserId varchar(10)
as
```

```
SELECT    Application.applicationStatus+' on
'+CONVERT(VARCHAR,Application.dateSubmitted,101)
  FROM        Application,Student
 WHERE    Application.StudentID=Student.studentID and  Student.UserID= @UserId

GO
```

Name: proc sp_GetStudentName
Description: Returns Student first and last name

```
CREATE   proc sp_GetStudentName
       @UserId nvarchar(20)
as

SELECT    FirstName+' '+LastName
FROM        UserData
WHERE    UserId = @UserId
GO
```

Name: proc sp_InProgressCoursesDetails
Description: List Names of the Courses student currently enroll

```
CREATE   proc sp_InProgressCoursesDetails
          @UserId nvarchar(20)
as

SELECT
StudentProgress.courseID,Course_Credit.credit,ISNULL(CourseSubstitute.courseIDSubstitute,'
N/A') as courseIDSubstitute
  FROM        Student
 INNER JOIN StudentProgress ON Student.studentID = StudentProgress.studentID
 INNER JOIN Course_Credit ON StudentProgress.courseID = Course_Credit.courseID
Left JOIN CourseSubstitute ON StudentProgress.courseID = CourseSubstitute.courseID
 INNER JOIN Major ON Student.majorID = Major.majorID AND StudentProgress.courseID =
Major.courseID
 INNER JOIN UserData ON Student.UserID = UserData.UserID
WHERE     (StudentProgress.courseStatus='R' ) AND (Student.UserId = @UserId)
GO
```

Name: proc sp_ InProgressCoursesTotalCredits
Description: Number of the credits for the courses in progress

```
CREATE  proc sp_InProgressCoursesTotalCredits
        @UserId nvarchar(20)
as

SELECT     SUM(Course_Credit.credit) AS TotCredits
  FROM        Student
 INNER JOIN StudentProgress ON Student.studentID = StudentProgress.studentID
 INNER JOIN Course_Credit ON StudentProgress.courseID = Course_Credit.courseID
 INNER JOIN Major ON Student.majorID = Major.majorID AND StudentProgress.courseID =
Major.courseID
 INNER JOIN UserData ON Student.UserID = UserData.UserID
WHERE     (StudentProgress.courseStatus='R' ) AND (Student.UserId = @UserId)
GROUP BY Student.studentID,  StudentProgress.courseStatus
GO
```

Name: proc sp_InsertStudentApplication
Description: Insert the Final Audit application which student filled out on-line

```
CREATE     PROC sp_InsertStudentApplication


        @ssn varchar(10)='',
        @lastname char(20)='',
        @firstname char(20)='',
        @middlename varchar(20)='',
        @program char(20)='',
        @certificate varchar(50)='',
        @graduatedate datetime='',
        @firstmajor varchar(50)='',
        @secondconc varchar(50)='',
        @secondmajor varchar(50)='',
        @firstminor varchar(50)='',
        @secondminor varchar(50)='',
        @specialization varchar(50)='',
        @prevaudit char(10)='',
        @prevauditdate datetime='',
        @othernameexist char(10)='',
        @otherlastname char(20)='',
        @datesubmit datetime = '',
        @UserID varchar(10) = ''
AS
```

```
declare @vstdid varchar(10)
select @vstdid=studentid from student where userid=@UserID

INSERT INTO Application
(studentSSNum,expectedProgram,expectedCertificate,dateGraduate,firstMajor,
secondConc,secondMajor,firstMinor,secondMinor,specializationArea,prevAudit,prevAuditDate,
otherNameExist,otherLastName,dateSubmitted,applicationStatus,StudentID)
VALUES
(@ssn,@program,@certificate,@graduatedate,@firstmajor,@secondconc,@secondmajor,
@firstminor,@secondminor,@specialization,@prevaudit,@prevauditdate,
@othernameexist,@otherlastname,@datesubmit,'Submitted',@vstdid)
GO
```

Name: proc sp_OtherCoursesDetails
Description: List Names of the Courses student took which are not required by major

```
CREATE       proc sp_OtherCoursesDetails
        @UserId nvarchar(20)
as

SELECT
StudentProgress.courseID,Course_Credit.credit,ISNULL(CourseSubstitute.courseIDSubstitute,'')
as courseIDSubstitute
  FROM        Student
 INNER JOIN StudentProgress ON Student.studentID = StudentProgress.studentID
 INNER JOIN Course_Credit ON StudentProgress.courseID = Course_Credit.courseID
  LEFT JOIN CourseSubstitute ON StudentProgress.courseID = CourseSubstitute.courseID
INNER JOIN UserData ON Student.UserID = UserData.UserID
WHERE     (StudentProgress.courseStatus NOT IN ('R', 'F', 'N'))
AND     (Student.UserId = @UserId ) AND (StudentProgress.courseID NOT IN
                    (SELECT     StudentProgress.courseID
                      FROM         StudentProgress, Major, Student
                      WHERE      StudentProgress.courseID = Major.courseID AND Major.majorID
= Student.majorID))
GO
```

Name: proc sp_ OtherCoursesTotalCredits
Description: Number of credits of the Courses student took which are not required by major

```
CREATE    proc sp_OtherCoursesTotalCredits
```

```
        @UserId nvarchar(20)
as

SELECT    SUM(Course_Credit.credit) AS TotCredits
FROM       Student
INNER JOIN StudentProgress ON Student.studentID = StudentProgress.studentID
INNER JOIN Course_Credit ON StudentProgress.courseID = Course_Credit.courseID
INNER JOIN UserData ON Student.UserID = UserData.UserID
WHERE    (StudentProgress.courseStatus NOT IN ('R', 'F', 'N'))
AND    (Student.UserId = @UserId )
AND (StudentProgress.courseID NOT IN
                    (SELECT    StudentProgress.courseID
                     FROM       StudentProgress, Major, Student
                     WHERE     StudentProgress.courseID = Major.courseID AND Major.majorID
= Student.majorID))
GROUP BY Student.studentID, StudentProgress.courseStatus
GO
```

Name: proc sp_SubstituteCoursesTotalCredits
Description: Number of credits for the substituted courses

```
CREATE   proc sp_SubstituteCoursesTotalCredits
        @UserId nvarchar(20)
as

SELECT    SUM(Course_Credit.credit) AS TotCredits
FROM       Student
INNER JOIN StudentProgress ON Student.studentID = StudentProgress.studentID
INNER JOIN Course_Credit ON StudentProgress.courseID = Course_Credit.courseID
INNER JOIN CourseSubstitute ON  StudentProgress.courseID = CourseSubstitute.courseID
INNER JOIN UserData ON Student.UserID = UserData.UserID
WHERE    (StudentProgress.courseStatus NOT IN ('R', 'F', 'N'))  AND   (Student.UserId =
@UserId )
GO
```

Name: proc sp_ValidateUser
Description: validates user name and password

```
CREATE proc sp_ValidateUser  @UserId nvarchar(20), @Password nvarchar(15) , @IsValid
Int  output
```

```
as
if (select count(*) from UserData where Userid= @UserId and Password = @Password ) = 1
   select @ISValid = 1
else
   select @IsValid = 0
return

GO
```

User Training

Once the system is deployed, the users will need to be trained to use the system. The students, due to their numbers, will be impossible to train in-person. Therefore, a web page will need to be created that describes the usage of the system for students. A hard-copy manual can be printed and distributed among students in addition to the web page. The university staff will be required to participate in a training class during which they will learn how to use the functionality of the system pertaining to them. Lastly, an administrator will require in-person training with the developer of the system to make sure s/he knows sufficient deployment and administration details to support the system going forward.

Appendix A: Application Screen Shots

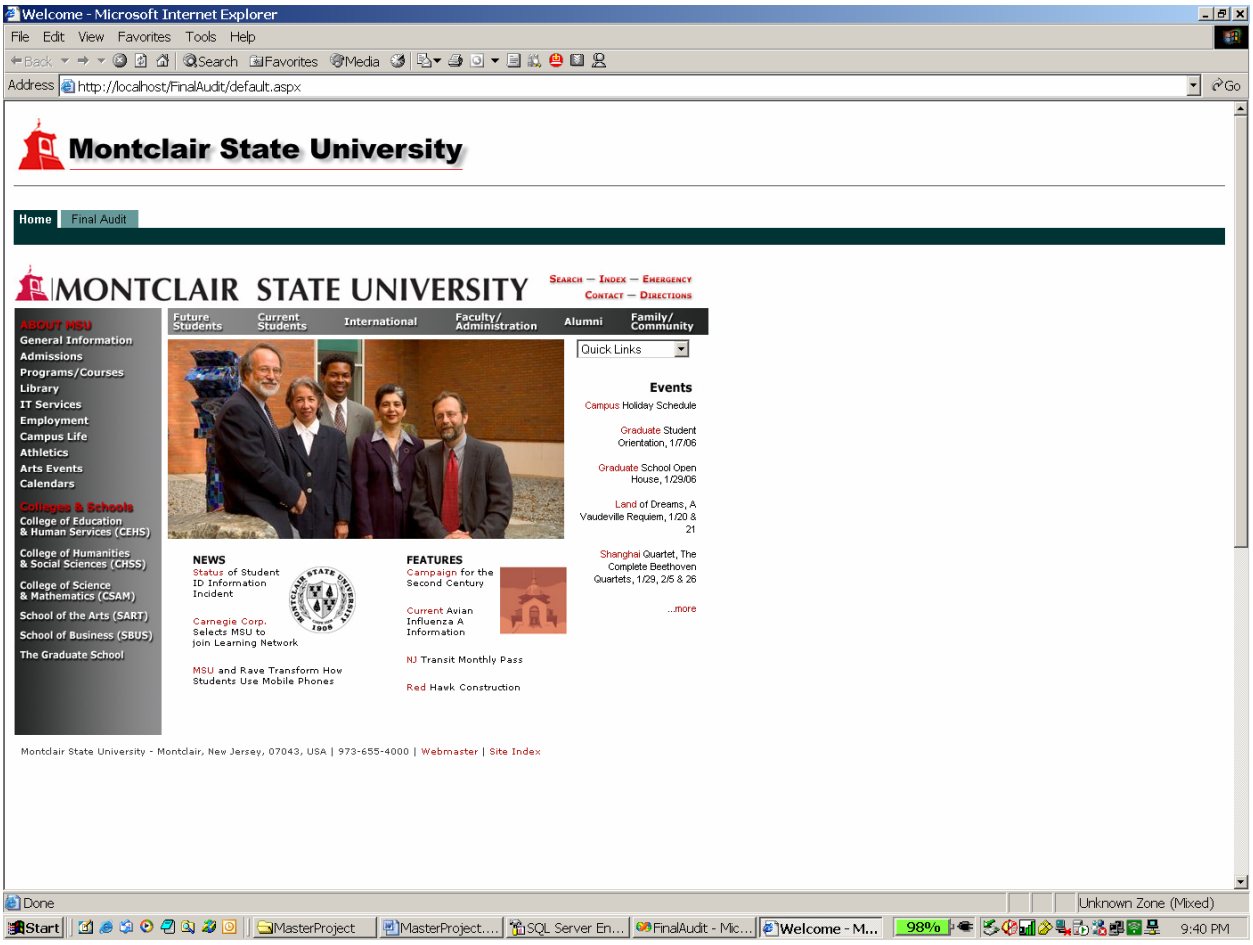

Figure 8: Main Screen

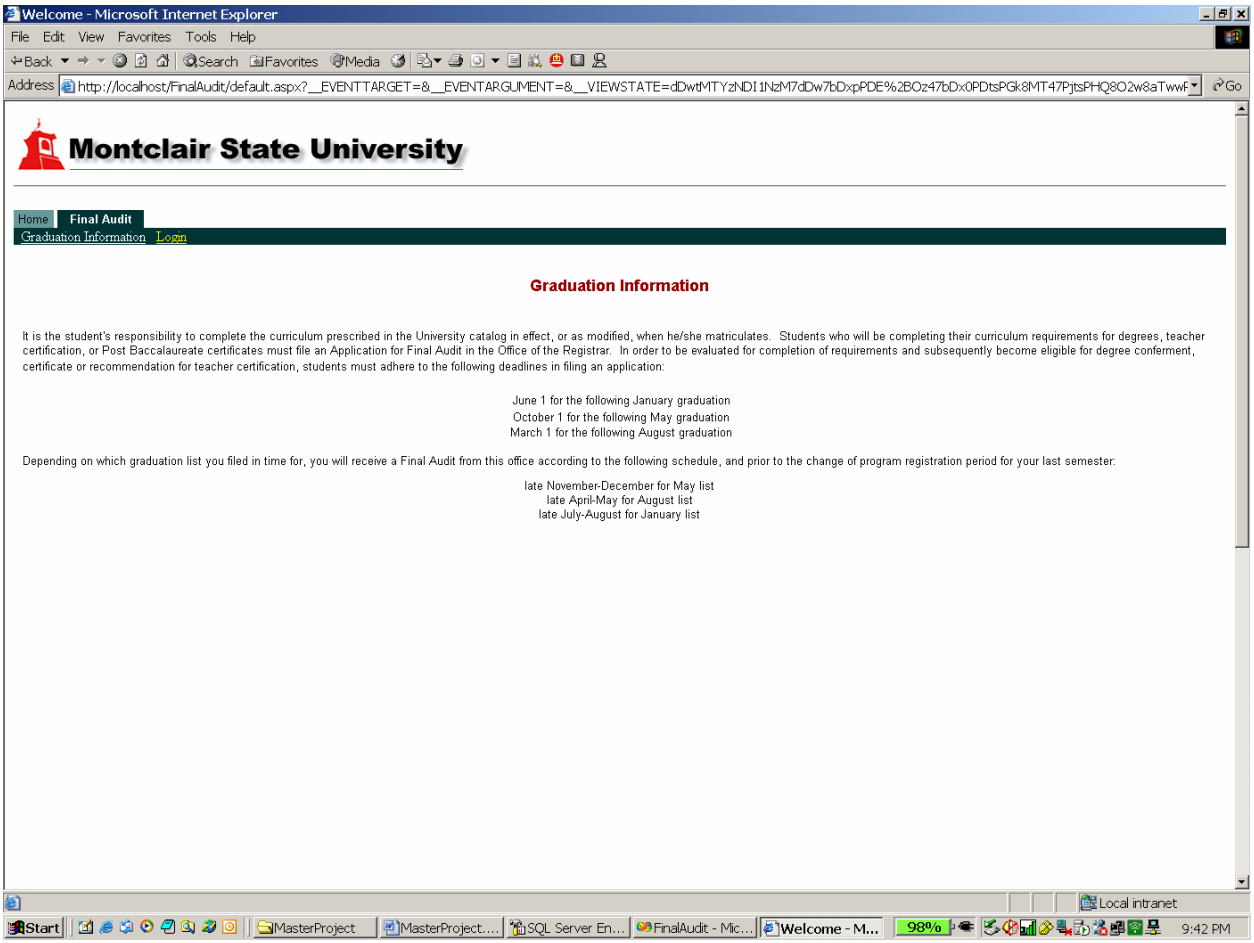

Figure 9: General Information Screen

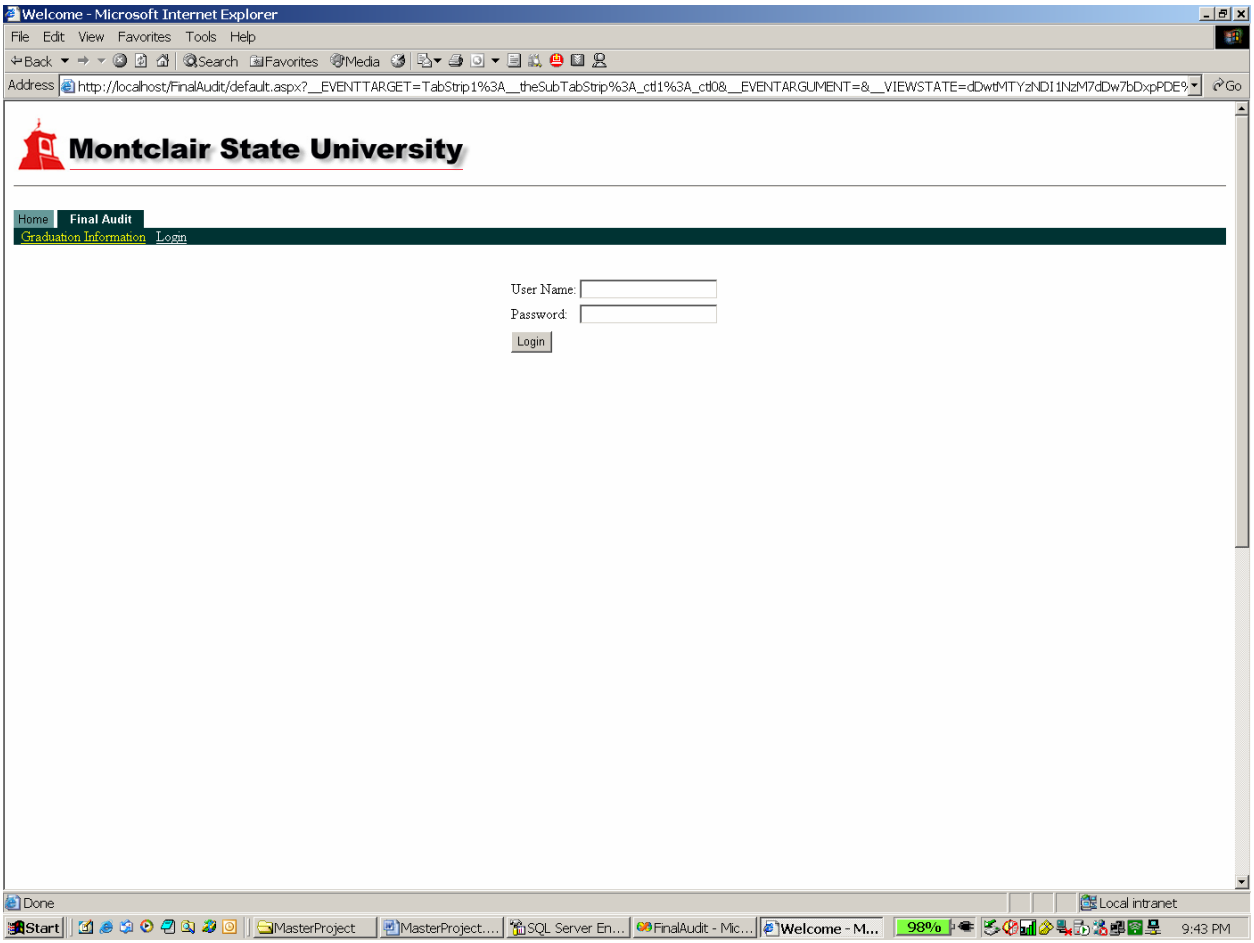

Figure 10: Login Screen

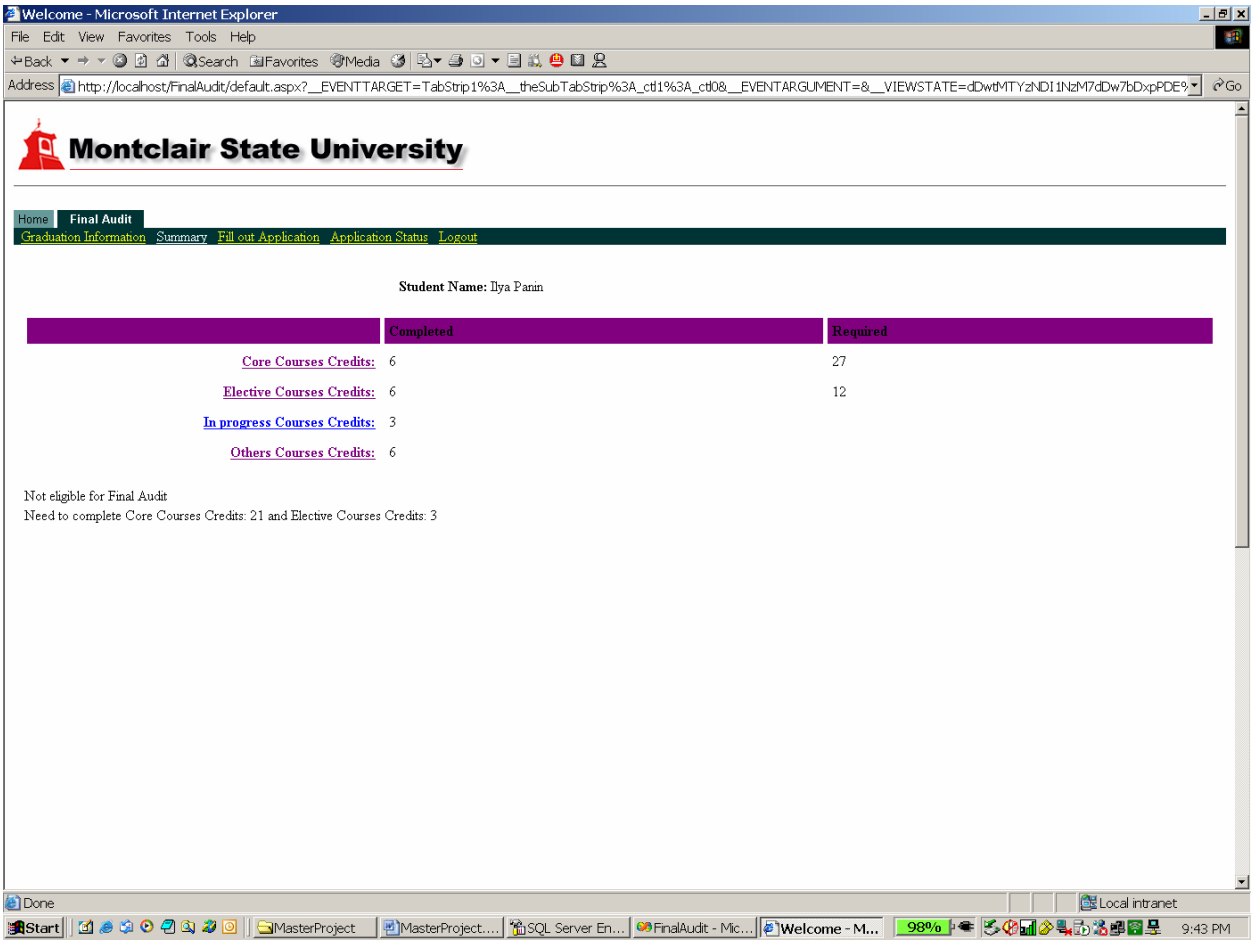

Figure 11: Student Summary Screen

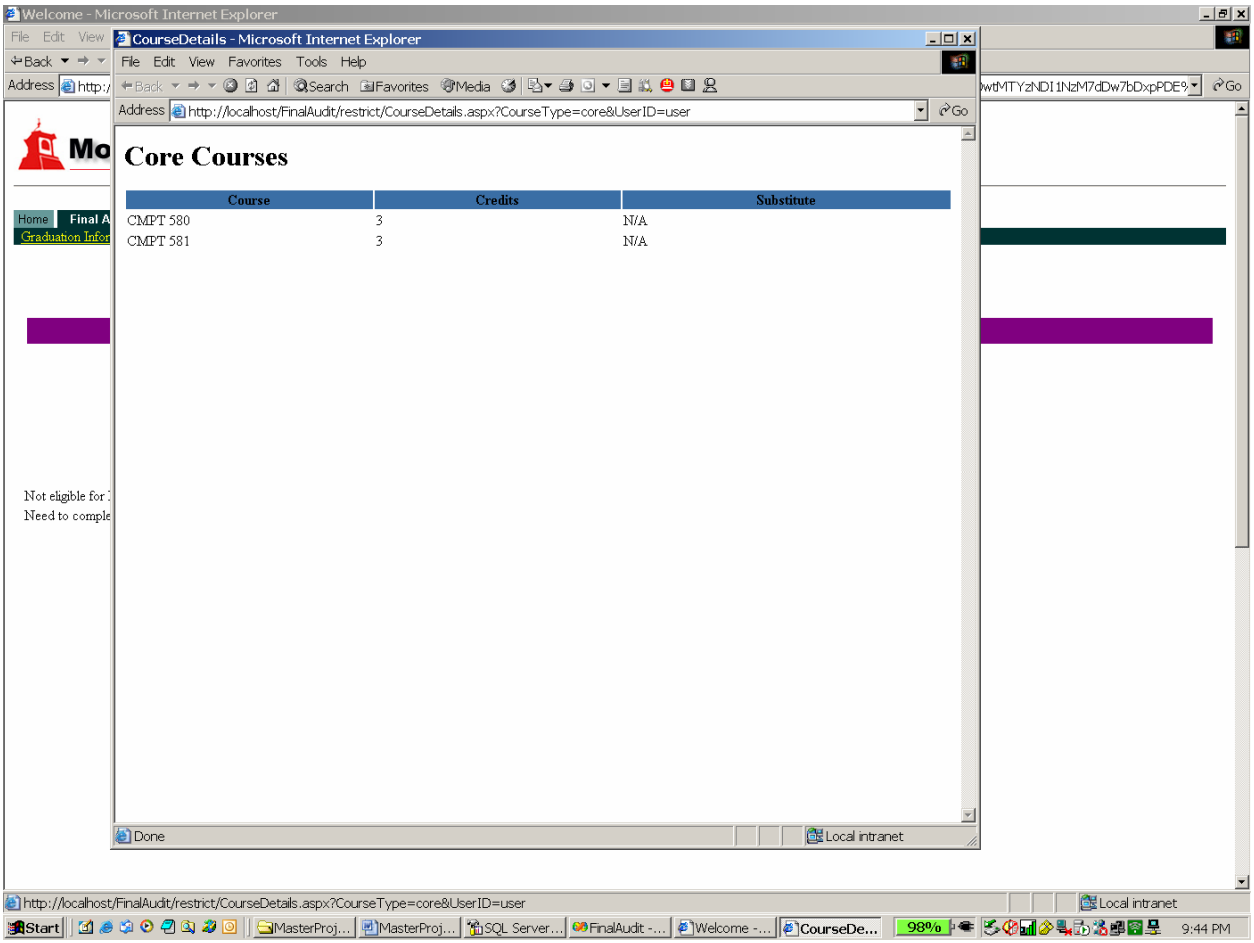

Figure 12: Core Courses Details Screen

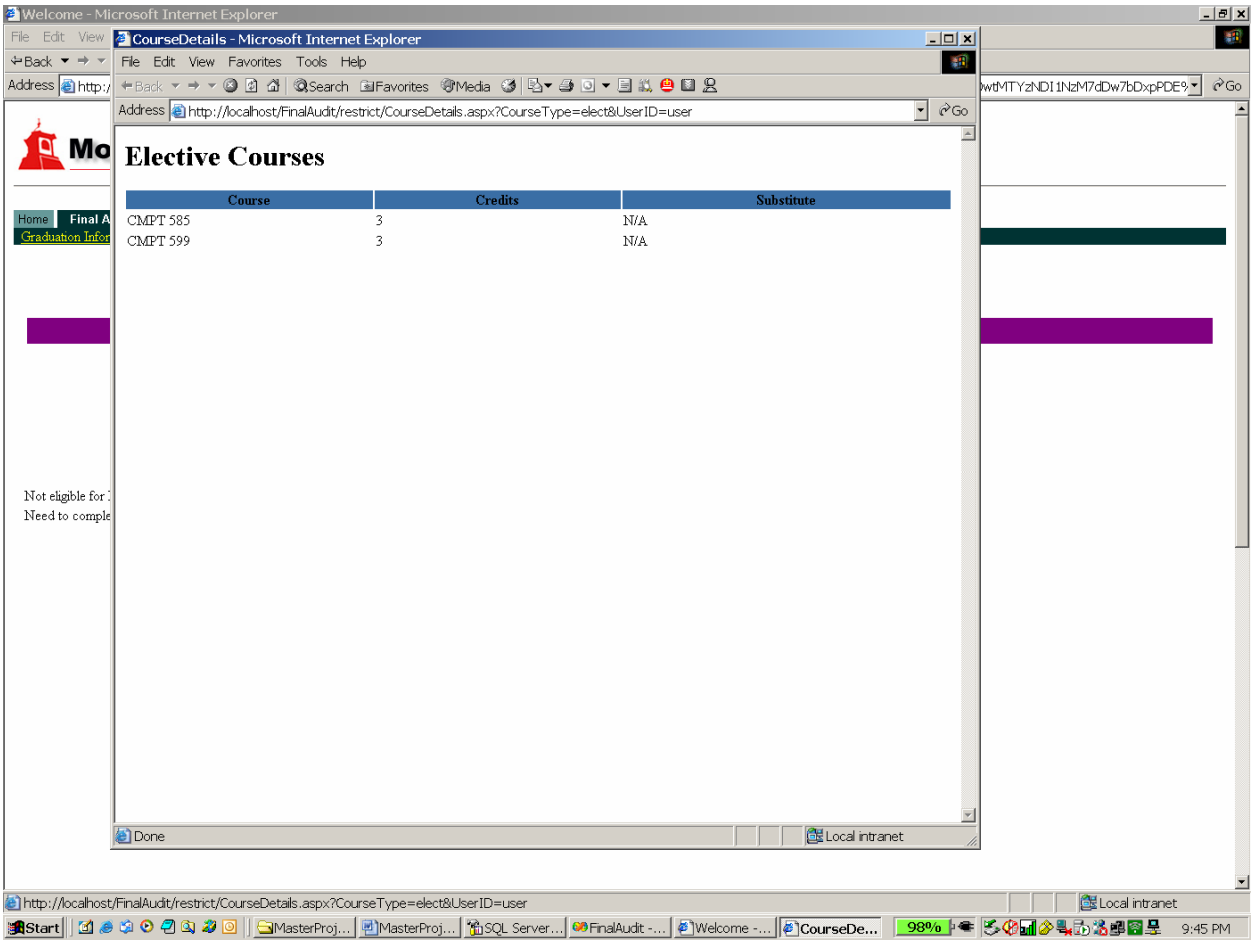

Figure 13: Elective Courses Details Screen

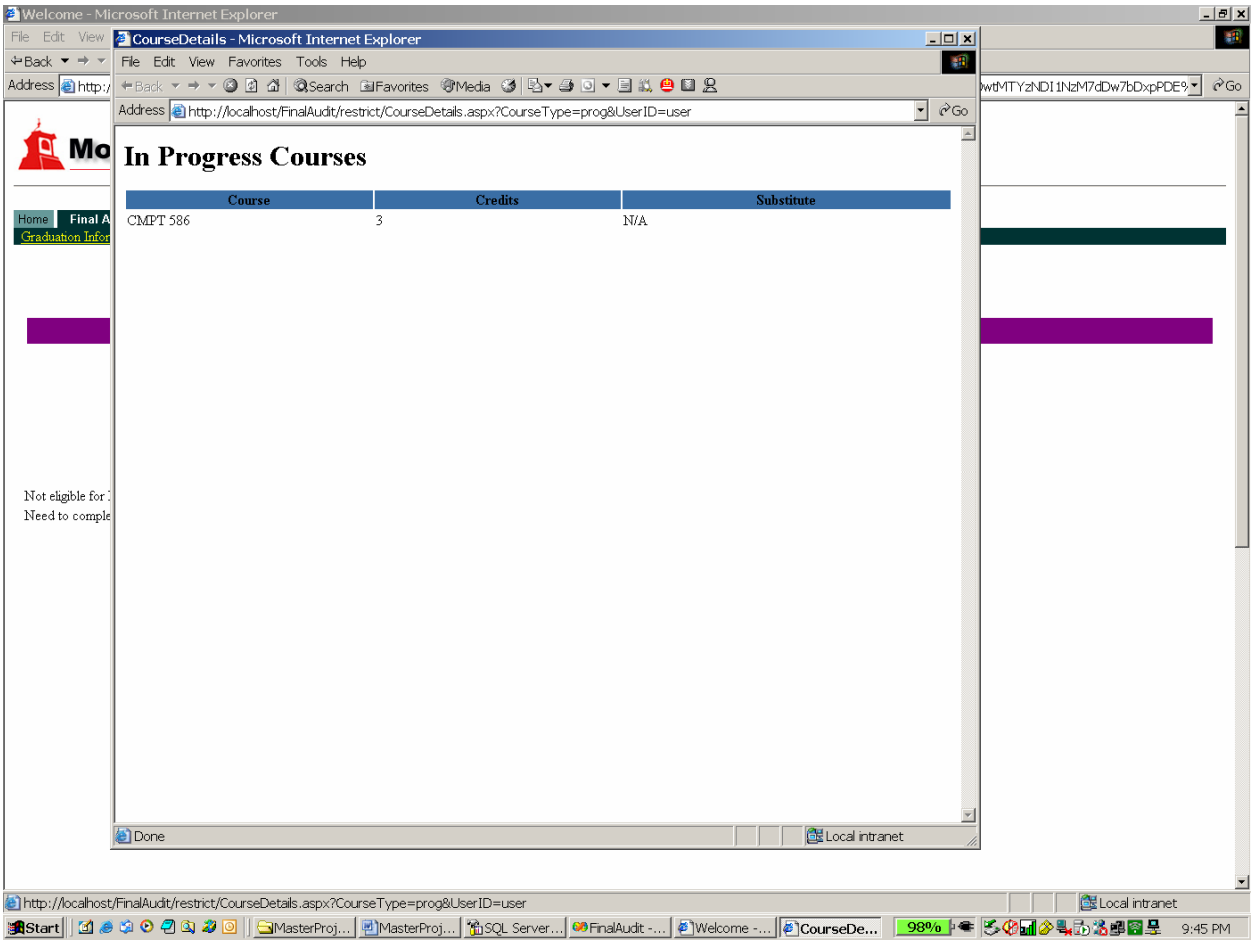

Figure 14: In Progress Courses Details Screen

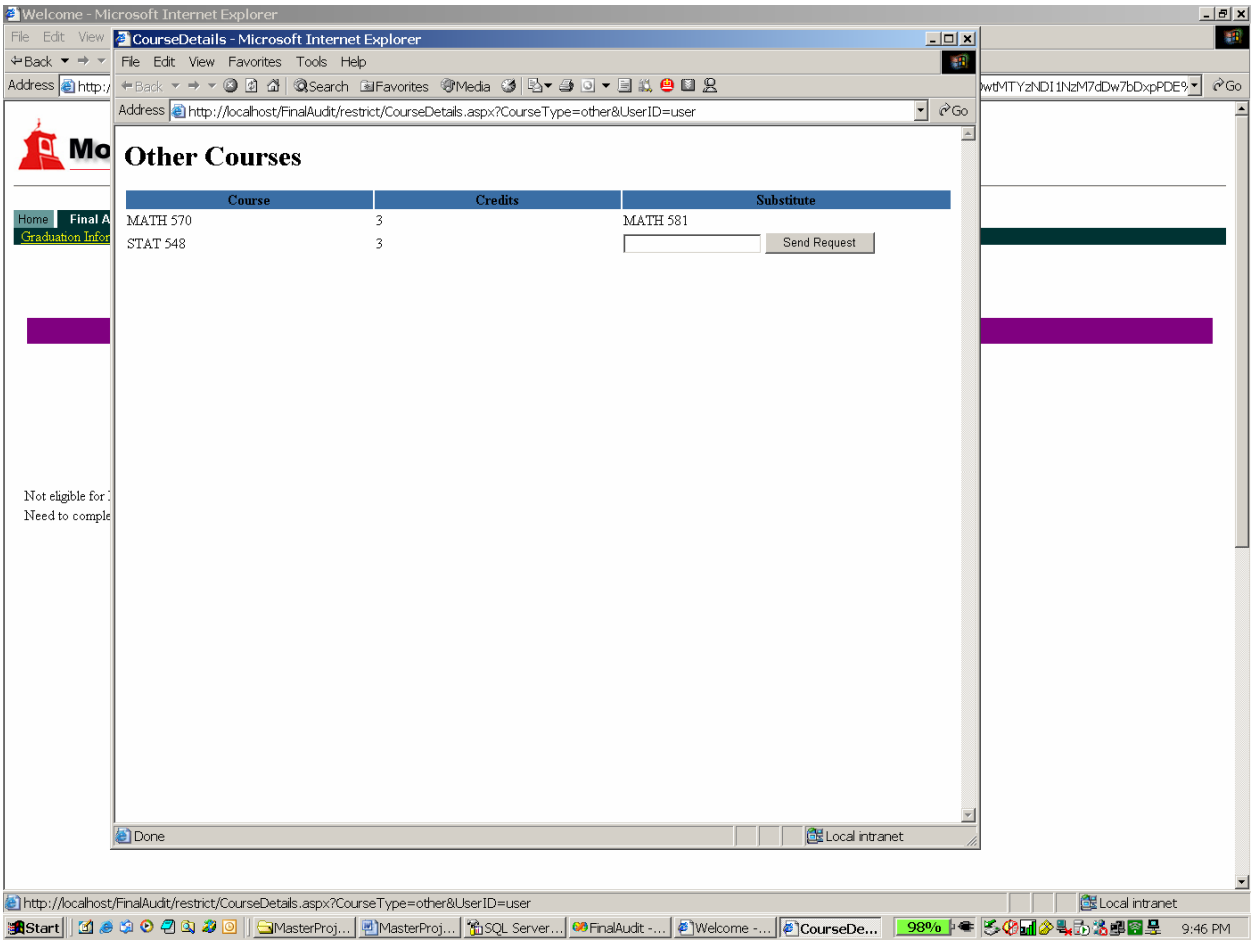

Figure 15: Other Courses Details Screen

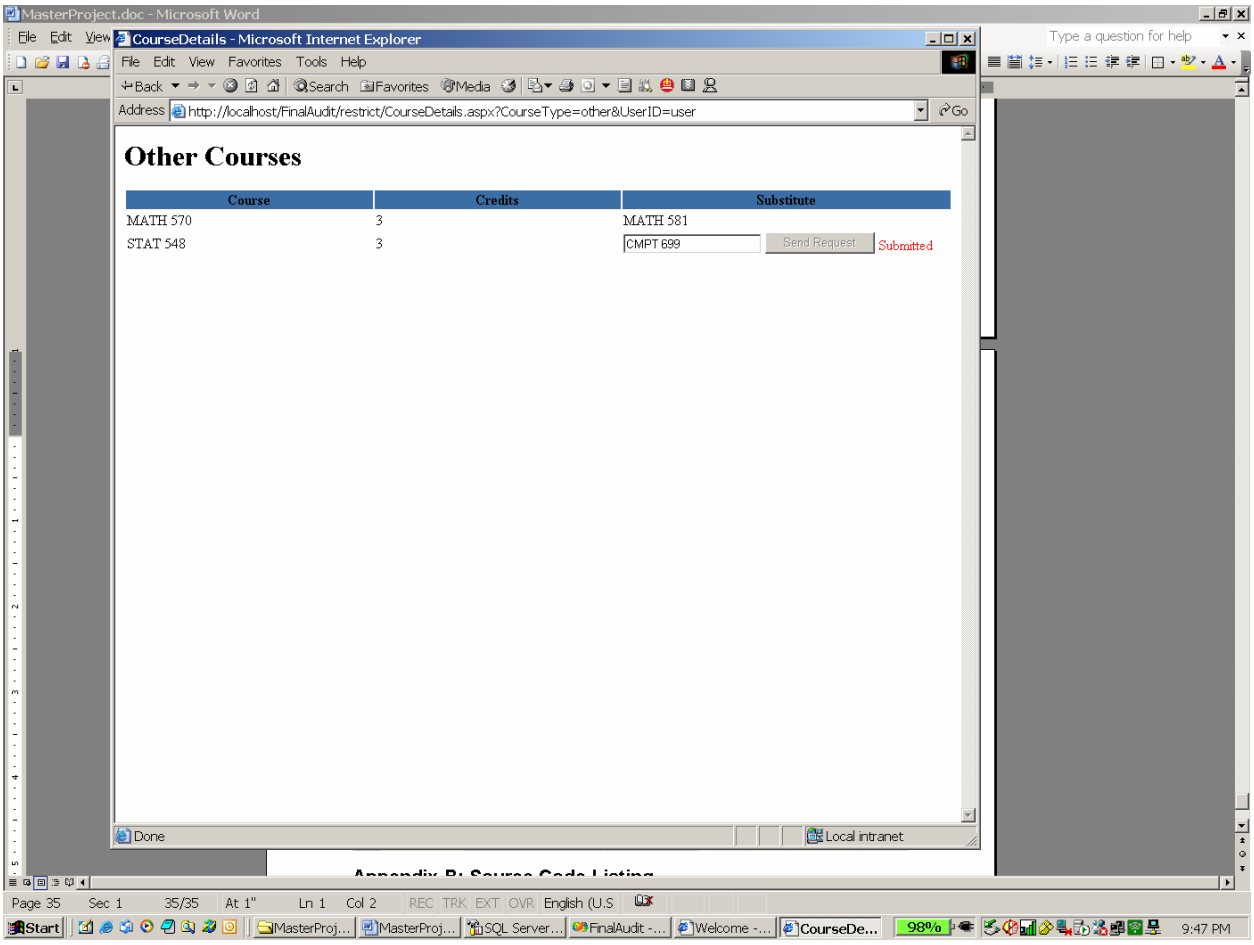

Figure 16: Submitted request screen

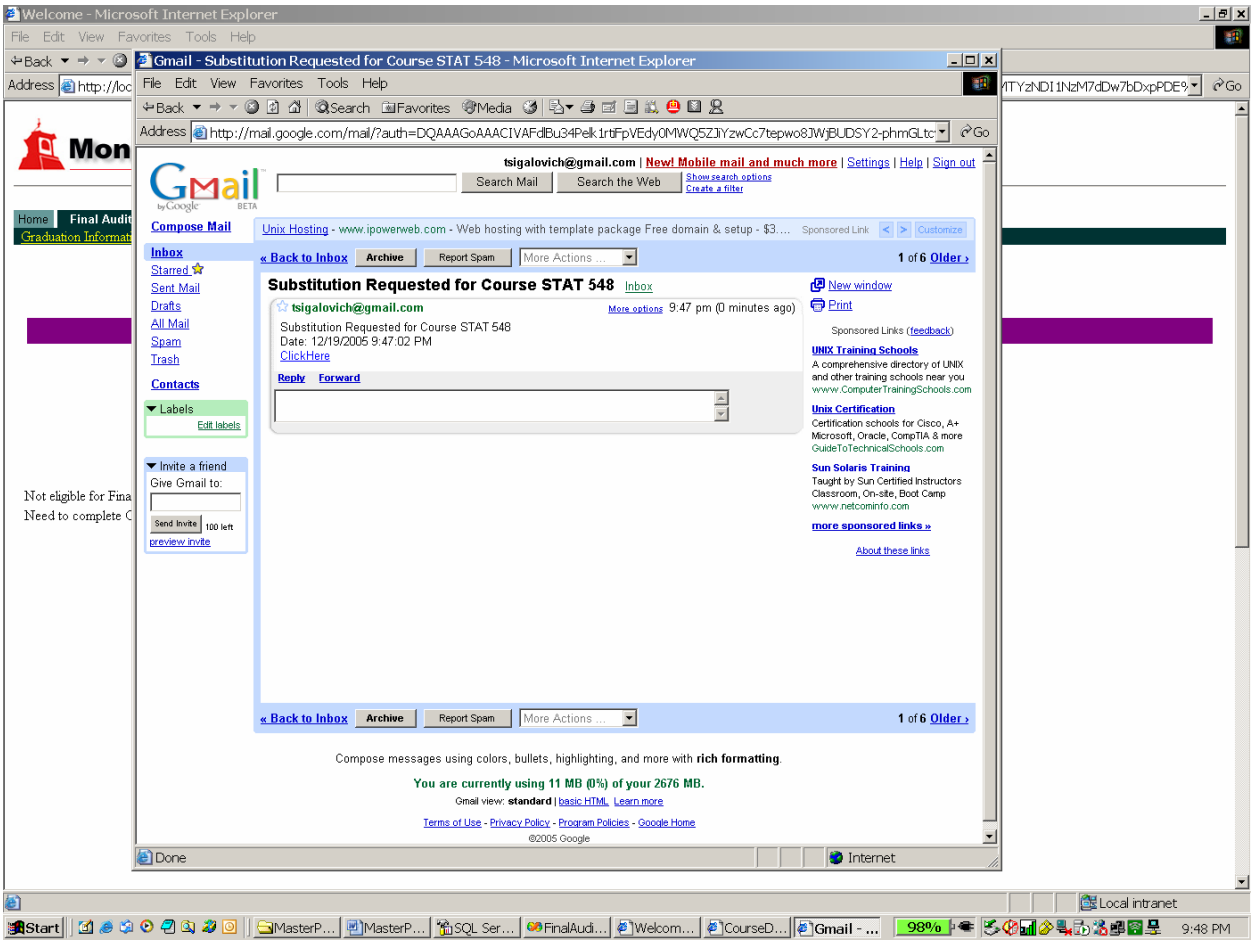

Figure 17: Substitution request email Screen

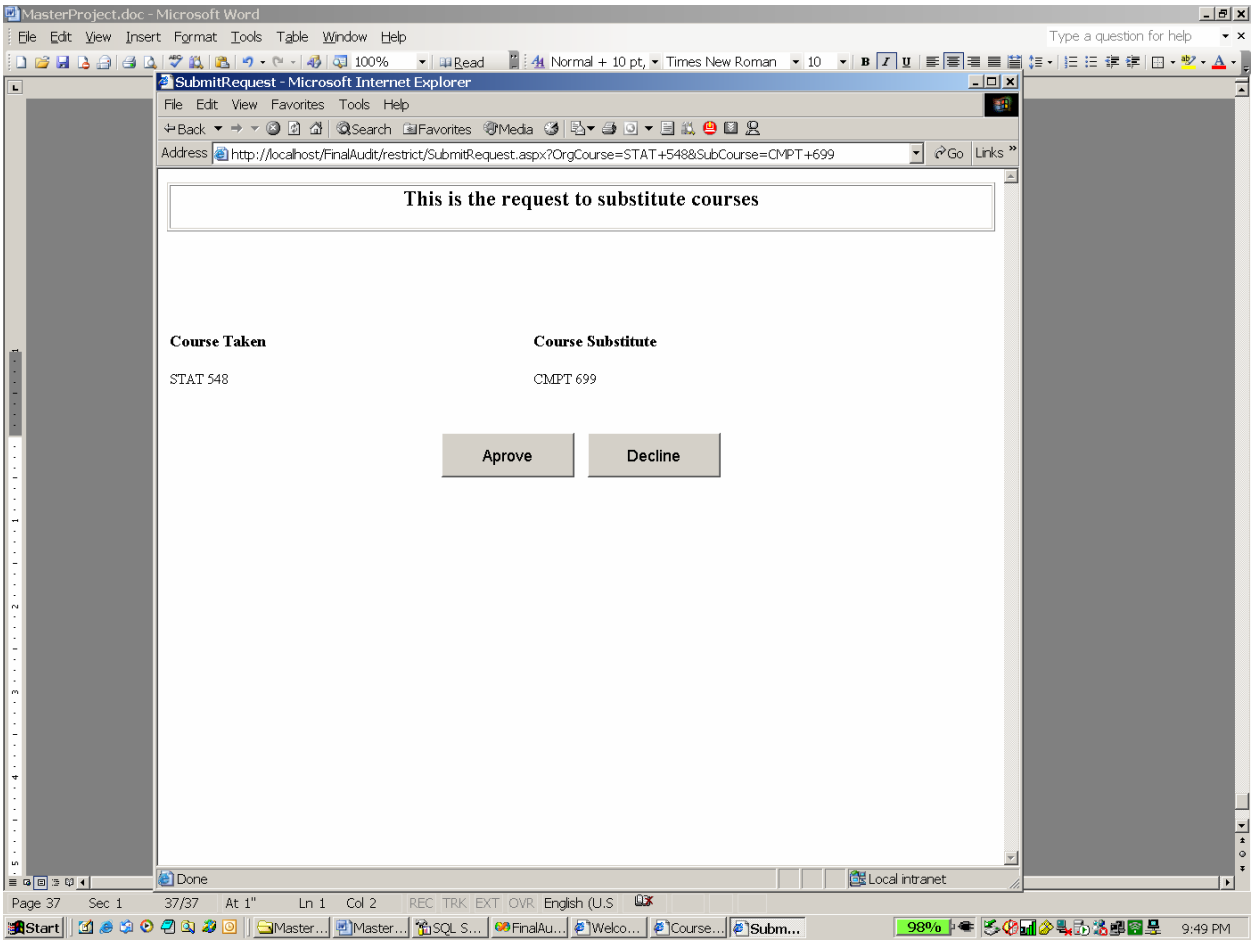

Figure18: Substitution approval Screen

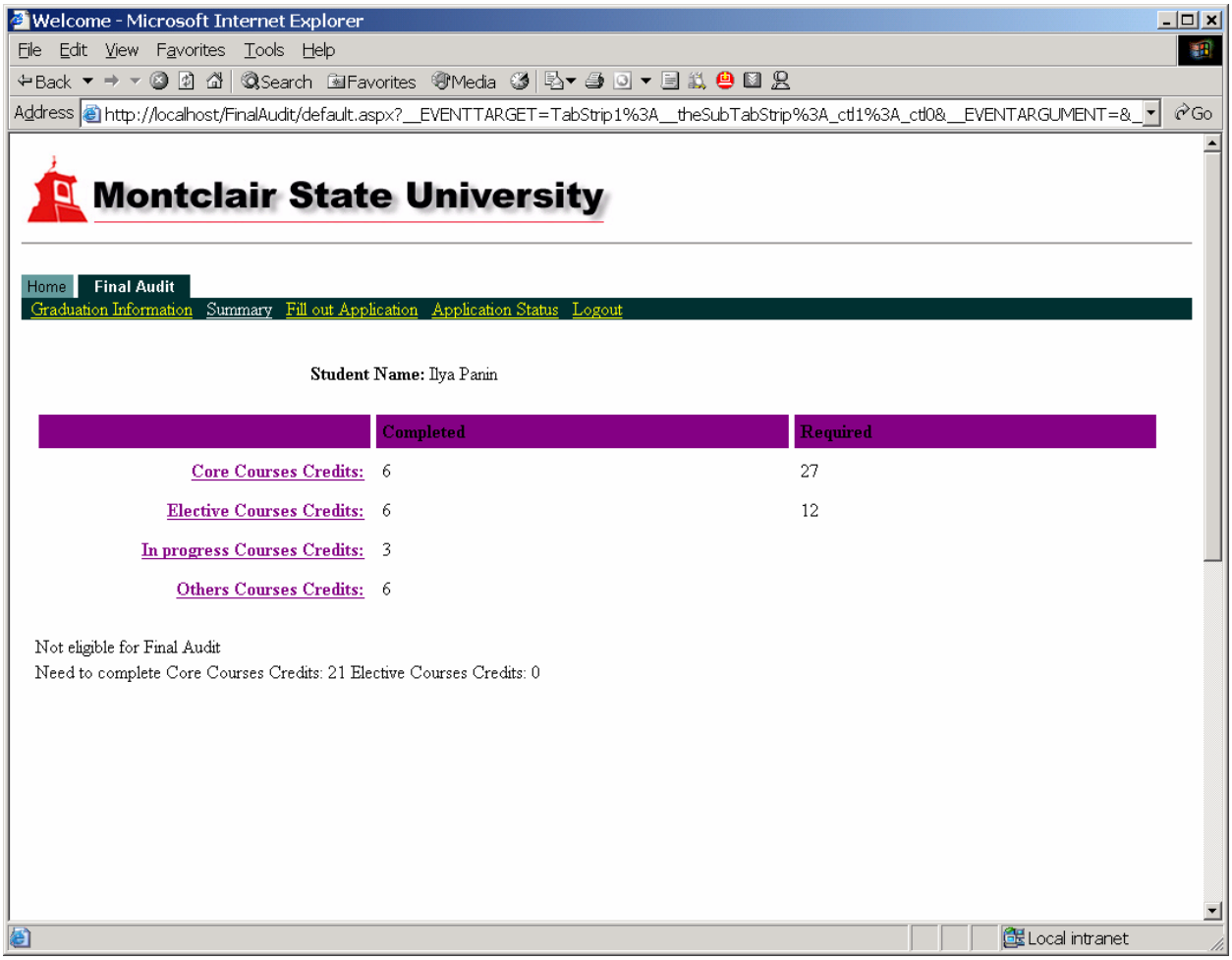

Figure19: Student Summary Screen

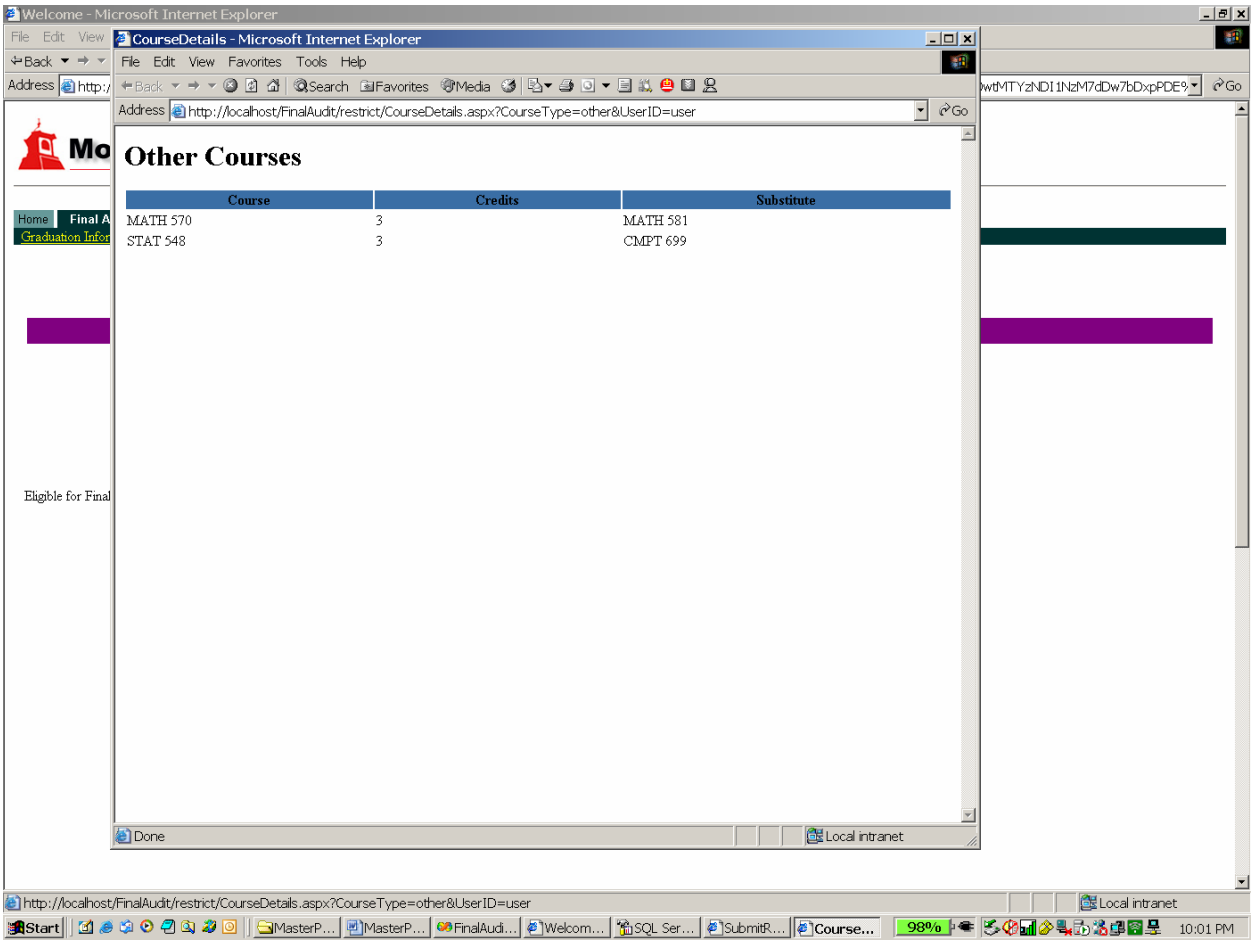

Figure 20: Other Courses Details Screen

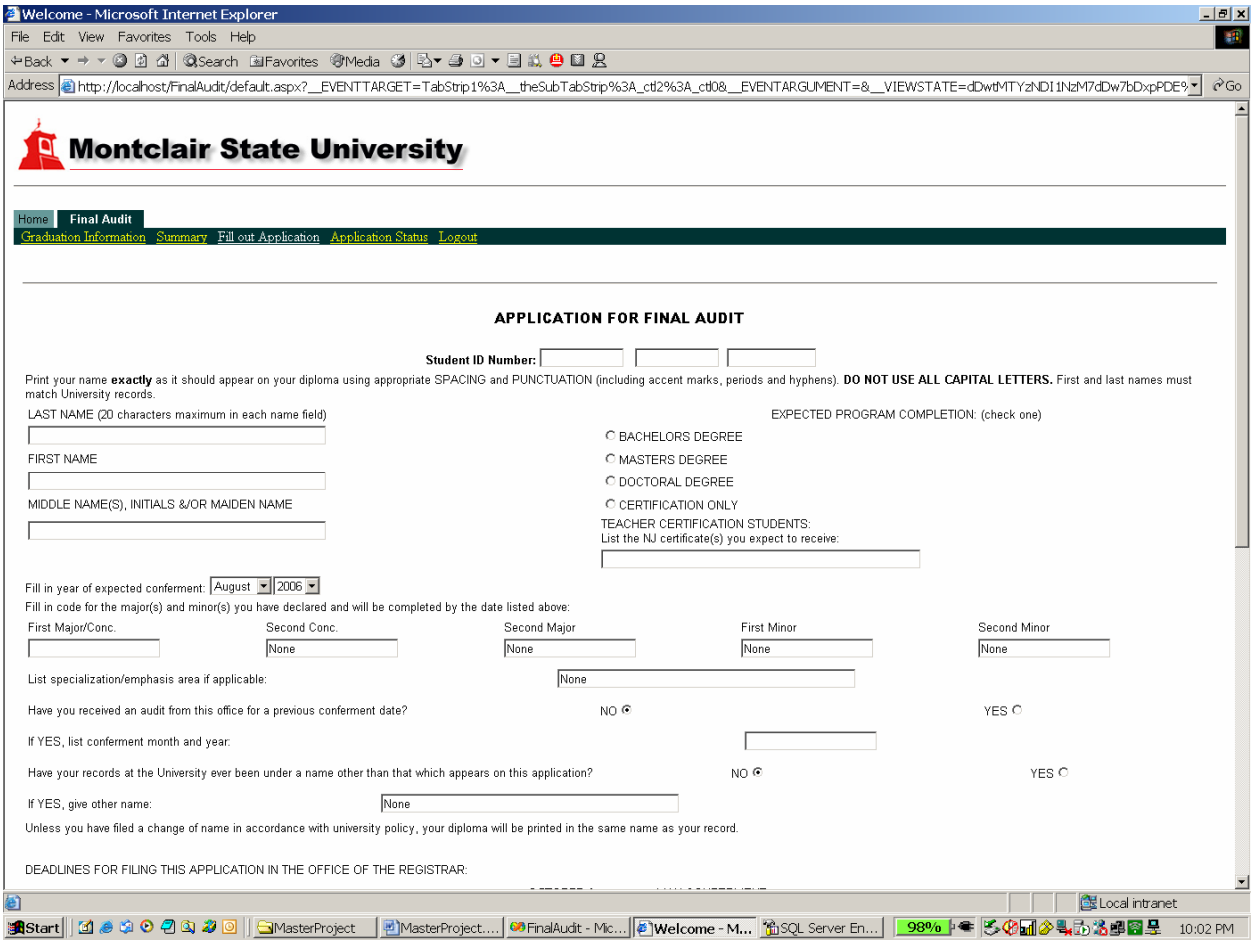

Figure 21: Final Audit Application Screen

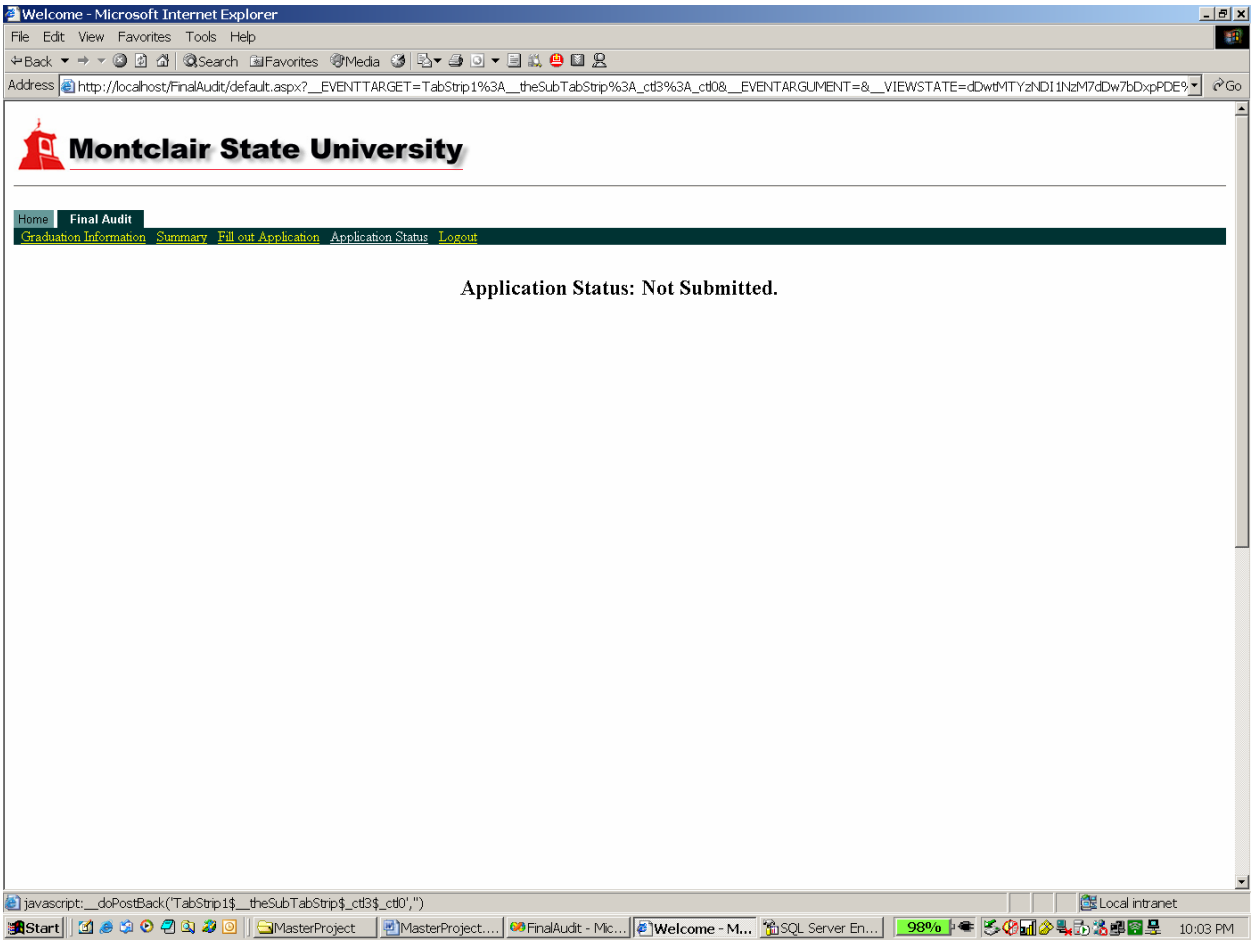

Figure 22: Final Audit Application Status Screen

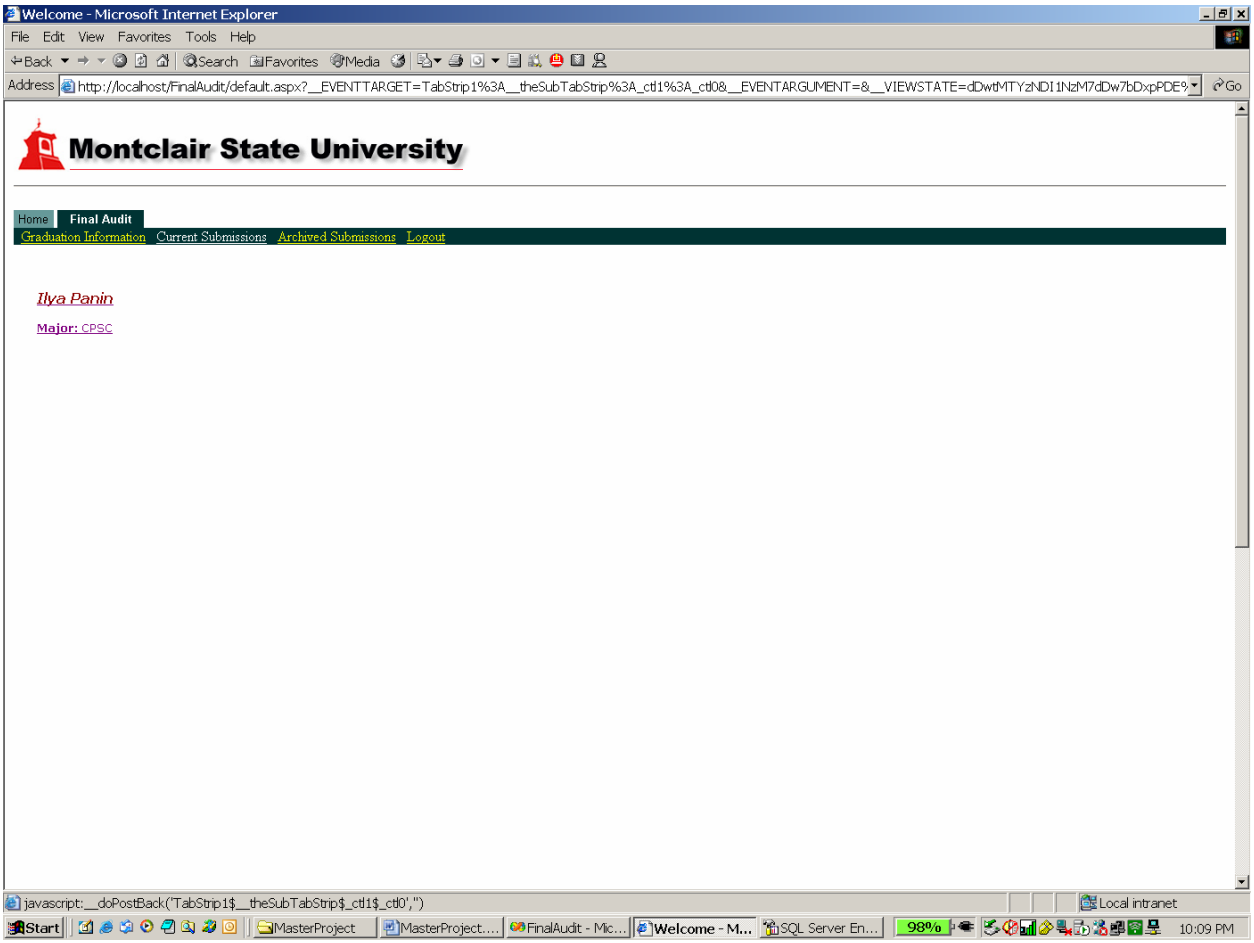

Figure 23: Registrar's Office clerk Screen

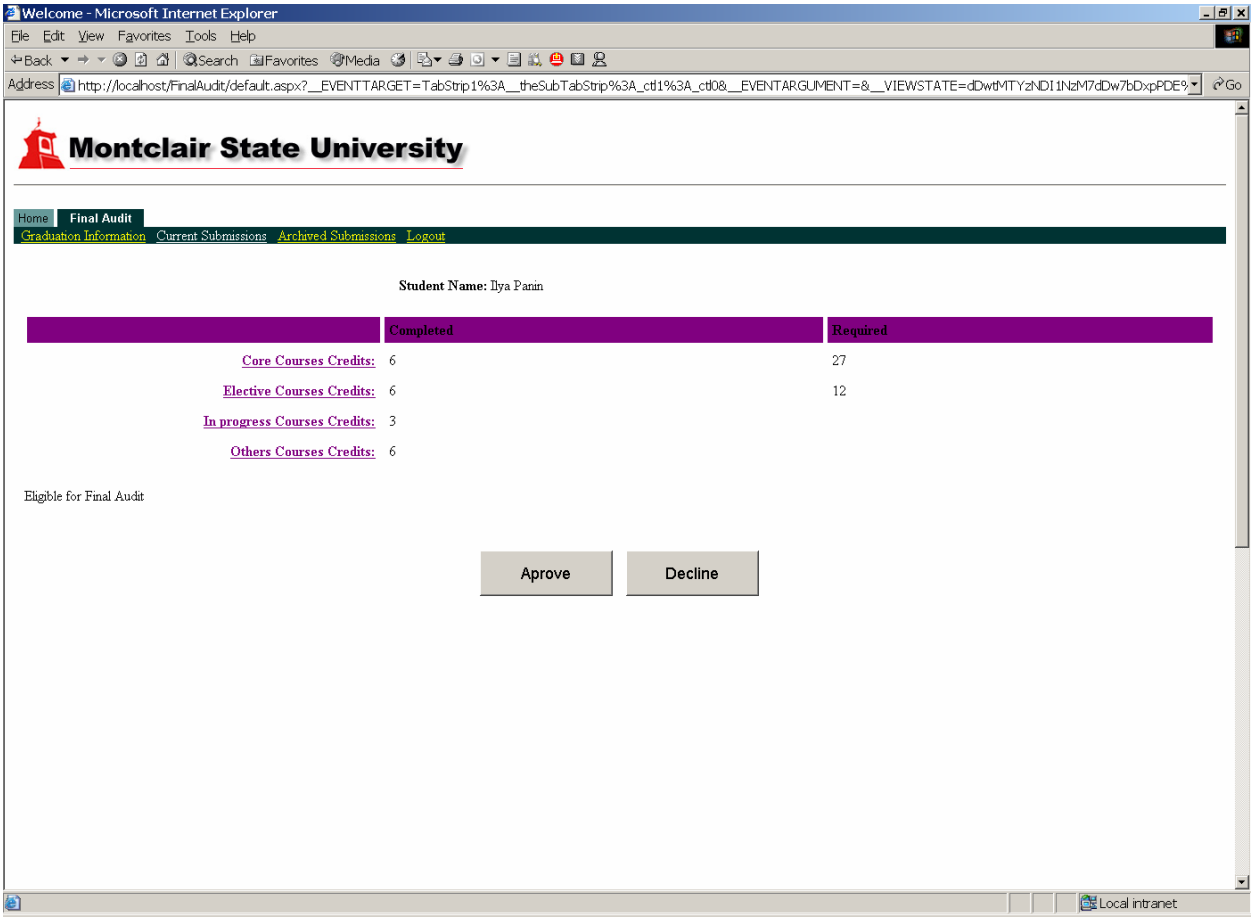

Figure 24: Registrar's Office Approve Screen

Appendix B: Source Code Listing

The following is the list of all the programs that are part of the Final Audit Application:

CourseDetails.aspx
CourseDetails.aspx.cs
CourseDetails.aspx.resx
fa.htm
finalaudit.aspx
finalaudit.aspx.cs
finalaudit.aspx.resx
status.aspx
status.aspx.cs
status.aspx.resx
stdlist.aspx
stdlist.aspx.cs
stdlist.aspx.resx
StdSum.aspx
StdSum.aspx.cs
StdSum.aspx.resx
StudentProgress.cs
SubmitRequest.aspx
SubmitRequest.aspx.cs
SubmitRequest.aspx.resx
default.aspx
default.aspx.cs
Login.aspx
Login.aspx.cs
TabStrip.ascx
TabStrip.ascx.cs

Bibliography

[1] http://msdn.microsoft.com: Online documentation
[2] http://www.codeproject.com
[3] Sams Publishing: ASP.NET Unleashed By Stephen Walther
[4] Que Publishing: Special Edition Using® Microsoft® ASP.NET By Richard Leinecker
[5] Sams Publishing:MICROSOFT VISUAL C# .NET 2003 UNLEASHED By Kevin Hoffman, Lonny Kruger
[6] SQL Server Books Online - the documentation for Microsoft® SQL Server™ 2000
[7] Database Management Systems By Ramakrishnan, Gehrke