

University of Essex-CE301 Dissertation Project



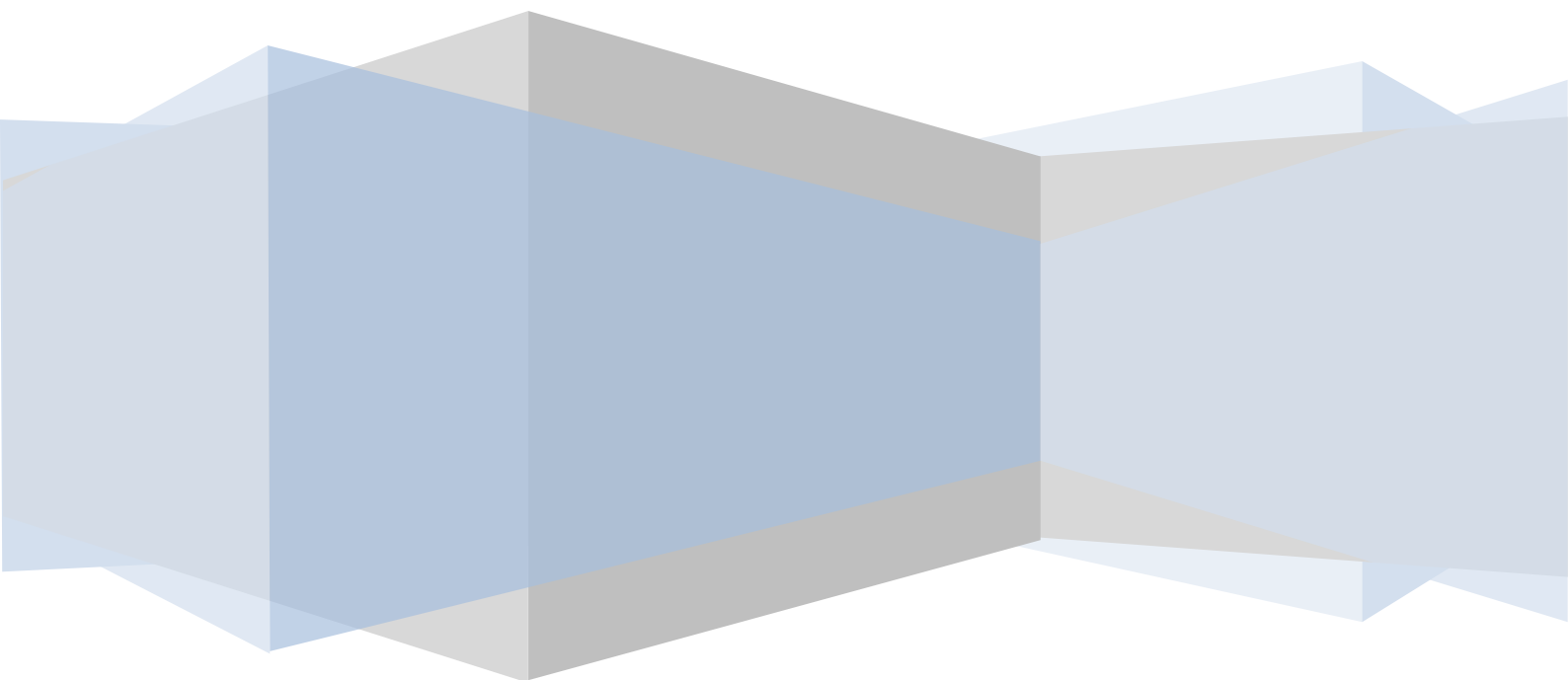
[1]

**Project 95- A Web Based Tool for Providing an Audit Service for the
Recognition of a Study, Abroad Period**

Studying BSc-Computer Science

Supervisor Anthony Vickers

Studying BSc-Computer Science



Final Report.....	1
1.0 Introduction	1
1,1 Abstract/Summary	1
1.2 Definitions, Acronyms and Abbreviations.....	2
2.0 Project Description.....	1
2.1 Specification	1
2.1.2 Changes to requirements and Specification	2
2.2 Design.....	2
2.2.1 Overall System Design.....	2
2.2.2 Interfaces Designs	3
2.2.2.1 Login page:	4
2.2.2.2 Main Administration page:	5
2.2.2.3 Student pages:	6
2.2.3 Database Design.....	8
2.3 Implementation	9
2.3.1 Additional implementation to specification	16
2.3.1.1 Database	16
2.3.1.2 Administrator pages.....	17
2.3.1.3 Student pages	19
2.3.1.4 All pages	21
2.3.2 Problems encountered	21
2.3.2.1 Database problems	21
2.3.2.2 Interface problems.....	22
2.5 Testing.....	27
2.5.1 Unit Testing	27
2.5.2 Integration testing	30
2.5.3 System testing.....	31
2.5.4 Compatibility testing.....	31
2.5.5 Acceptance Testing	32
2.5.6 Regression vs Retesting	32
2.5.7 Future testing.....	32
3.0 Project Planning/Reflection	32
3.1 Gantt Chart.....	32

4 Conclusions	33
4.1 Future Advancements	34
5 Acknowledgements	35
6 References	35
7 Appendix	36
Appendix A	36
Appendix B	37

Final Report

1.0 Introduction

This report will describe the different processes I went through whilst planning, designing and creating my project and the changes I made along the way. I will also be discussing the problems I faced and how I dealt with them, how the project could be expanded or what I would have done differently.

1,1 Abstract/Summary

In this day and age it is becoming ever more popular for students to become more adventurous with their degree. Not only undertaking joint degrees or industrial years but one desirable choice is to Study abroad. This can mean for their whole degree period studying in another country or just a term or a year of their degree.

The course they will take are agreed before the study abroad period by both Universities and the student, this is done by signing a paper learning agreement. When the period of study abroad has ended the host University provides a transcript listing the courses taken and the credit and grade awarded. The home University takes this transcript and transfers the courses, credit, and grade to the home University transcript database of the student. This process is not being managed correctly to acknowledge the students accreditation. This project will play a part in aiding to improve these problems to maintain a good base for Study Abroad programs for the University of Essex.

This project is a web based developed system which will provide functionality for electronically uploading and accessing each student's learning agreements, home and host transcripts. These are all validated and are only to be accessed securely with password protection.

University of Essex and Overseas students can apply to start a programme directly via online interfaces with preformatted forms. This information is saved and made accessible for easy access when required. This provides archiving and cataloguing of International programmes maintainable.

Processing applications, learning agreements and transcripts are managed by The Essex Study Abroad Office (Administration). The main Administration page alerts them to how many applications there are and at what stage of the study abroad process e.g. awaiting transcript, started study abroad period, awaiting signed Learning agreement etc.

They can upload/download the appropriate documents at any stage of the study abroad process. Staff can also make minor changes to the data stored in the Relational database e.g. adding or removing a course or department for Essex or a university from the programme.

This limits any technical administration interference in the process.

1.2 Definitions, Acronyms and Abbreviations

Relational Database – “ Database in which all data are represented in tabular form. The description of a particular entity is provided by the set of its attribute values, stored as one row or record of the table, called a tuple. Similar items from different records can appear in a table column. The relational approach supports queries that involve several tables by providing automatic links across tables. Payroll data, for example, can be stored in one table and personnel benefits data in another; complete information on an employee can be obtained by joining the tables on employee identification number. In more powerful relational data models, entries can be programs, text, unstructured data in the form of binary large objects (BLOBs), or any other format the user requires.”[2]

Redundancy(data redundancy)- “The occurrence of values for data elements more than once within a file or database.”[3]

MD5- “An algorithm created in 1991 by Professor Ronald Rivest that is used to create digital signatures. It is intended for use with 32 bit machines and is safer than the MD4 algorithm, which has been broken. MD5 is a one-way hash function, meaning that it takes a message and converts it into a fixed string of digits, also called a message digest.

When using a one-way hash function, one can compare a calculated message digest against the message digest that is decrypted with a public key to verify that the message hasn't been tampered with. This comparison is called a hashcheck.” [4]

ODBC-“ Open Database Connectivity (ODBC) is a common framework for accessing and altering the contents of databases. It allows developers to use the same coding conventions regardless of the actual database platform implemented on the backend. When a new database type is installed, administrators merely need to install an ODBC driver that supports that platform and existing ODBC software should function normally.”[5]

Sql query –“ Queries are the primary mechanism for retrieving information from a database and consist of questions presented to the database in a predefined format. Many database management systems use the Structured Query Language (SQL) standard query format.”[6]

Datatable – “An on-screen display of the information in a database management system, presented in columnar format, with field names at the top.” [7]

Referential Integrity- “A feature provided by relational database management systems (RDBMS's) that prevents users or applications from entering inconsistent data. Most RDBMS's have various referential integrity rules that you can apply when you create a relationship between two tables.”[8]

Primary and Foreign Keys-“Primary and foreign keys are the most basic components on which relational database theory is based. Primary keys enforce entity integrity by uniquely identifying entity instances. Foreign keys enforce referential integrity by completing an association between two entities.”[10]

2.0 Project Description

My previous reports have outlined the relevant specifications of what is demanded of my system and how I have designed my system to support and accomplish these. Throughout the implementation process these have altered and adjusted due to not only the specification of the customer but also the capabilities of the software and my abilities as a programmer.

2.1 Specification

My project's specification was made up of requirements for my project that I aimed to achieve, these have not altered much from my interim report see reference[9].

The project is web based and is straight forward to use and easily accessed at any point to login. It adheres to a simple layout scheme with not too much bulk on each page. Navigation between the pages are consistent by having buttons and links in the same positions and also having a common format for each page.

The functionality of my system meets the requirements in several ways. One by providing an Administration page where they can keep track of the stages of the study abroad process including:

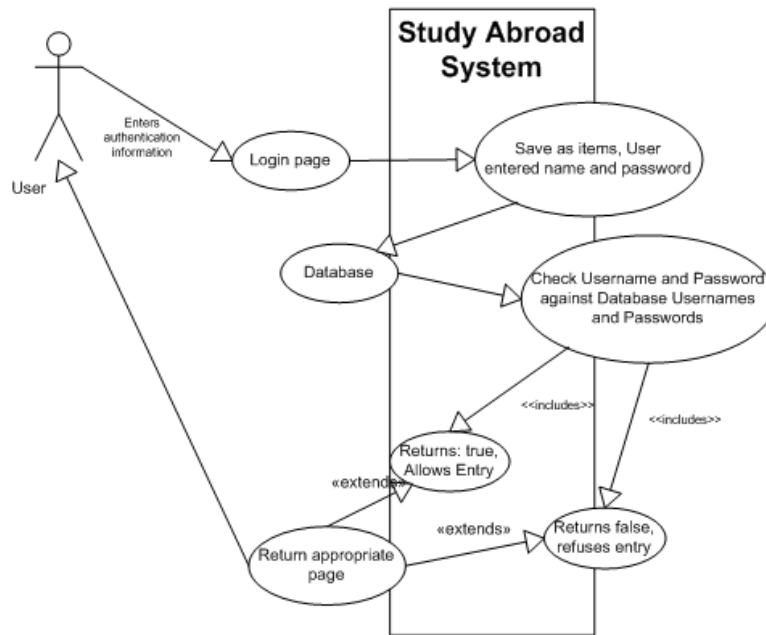
- ❖ Personal details and application submitted.
- ❖ Application accepted and Learning agreement made.
- ❖ Learning Agreement Signed,
- ❖ Study Abroad started
- ❖ Study Abroad ended
- ❖ Host Transcript received
- ❖ Home transcript received

This means all the International programmes are easily catalogued and maintained through the system through the storage in the backend database. It also allows the administration to more easily search for certain students.

The students also benefit by being able to complete an online application process for a period of study abroad whether incoming or outgoing. This reduces the use of paper material and the chance of losing the students data.

As a requirement all pages need to have restricted access corresponding to their login to stop unauthorised access to irrelevant interfaces. I achieved this requirements using sessions. Each user can only access the page they are related to e.g. an incoming student cannot access the administration page.

The use case diagram(found on the next page referenced from[9]) still accurately shows the login process for my system, For students returning the appropriate page is also broadened to check at what stage the Student is at, aka have they entered their personal details or not.



2.1.2 Changes to requirements and Specification

New users

Each username for the system has to be distinct; therefore when new students apply they will need a username and password. The requirement 3.2.5 from [9] quotes *'Students shall be able to complete an online application process for period of study abroad whether incoming or outgoing'*. I had initially stated that I would create a separate login page for new users where they can create themselves a new username and password. Instead I created a page where they enter their email address and name; this is submitted to the Administration page, not via a trigger but via the main screen. The Admin processes it and creates a Username and password.

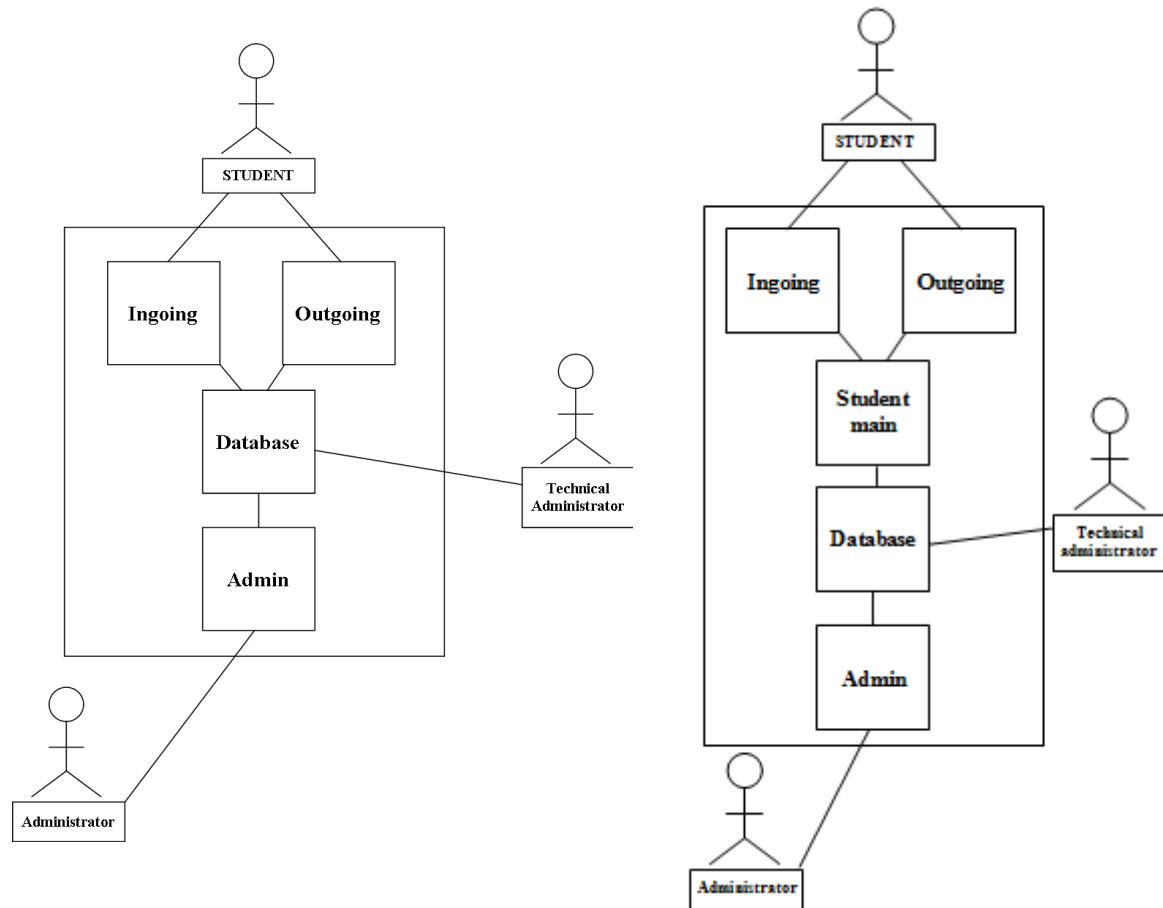
The requirement 3.2.12 from [9]: *'The database should be created and redundancy and errors reduced'*. When designing my database and creating their relationships I followed these requirements. For the student initial application input the database had to be altered for some of the Primary keys that were foreign keys to allow the insertion of data. How redundancy is still retained as the UserID's are checked on Login therefore are verified to be correct and valid.

2.2 Design

Throughout the implementation stage I was referring to my designs not only when selecting CSS styles, object placement for functions but also the system and database design.

2.2.1 Overall System Design

I initially created a system environment diagram (below left from [9]) to show the interaction of actors and the intended interfaces. These pages altered slightly, as a joint student page was created after the main introductory separate incoming and outgoing pages (shown below right)



(Above left –old, above right-new)

2.2.2 Interfaces Designs

The web page interface designs followed the current Essex webpage designs and had a simple consistent layout throughout the login, student and Administrator pages.

The initial designs comprised of a header with the University logo and a banner of photos and simple images, with the main body underneath. This is where the main functionality of the system occurs. My system design is very similar to this with the same header layout but with an extra study abroad logo I created. Then the main body below but with an extra footer at the bottom of each page, with a slightly altered coloured scheme.

The main body of each page whether a student or an admin page are have the same Css formatting style and only differ in size and contents. From my original designs, I kept the Navigation bar and overview layout. The navigation bar on:

- ❖ the administration page contains all the links that direct to pages where minor changes can be made to the database (e.g. add or editing Universities, Courses, Departments, Students etc).
- ❖ the student application pages contains the application stages titles which are highlighted bold in relation to what stage they are completing.
- ❖ the main student page is blank but kept for formatting consistency.

The body of the main body consists of a variety of information either the application forms, alerts or tables etc.

The snapshots below show the original layout of each page, followed by the actual system current design. As you can see there are only slight differences depending on the requirements that were demanded of the system and also by what looked suitable.

2.2.2.1 Login page:

University of Essex

Study Abroad Office

home

Log in

Username

Password

[New user?](#)

The login page has only altered slightly to the original design :

- ❖ There is a opening message of 'Welcome to Essex Study Abroad'
- ❖ The login area was taken out of the central box as It seemed too dark in colour and because I used the Visual Studio incorporated Login Function, the layout and structure are slightly different. I could have centre aligned it like the designed but I thought it looked neater having it left aligned.

University of Essex

UNIVERSITY OF ESSEX
STUDY ABROAD SCHEME

Study Abroad Office

WELCOME TO ESSEX STUDY ABROAD

Please Login:

User Name:

Password:

☐ Remember me next time.

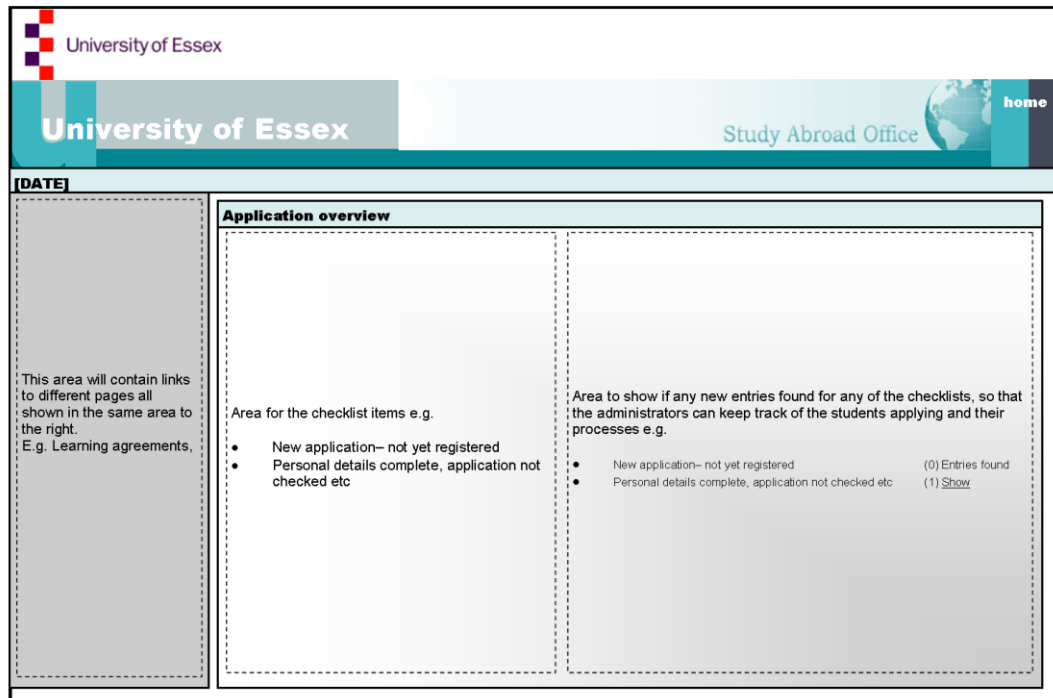
[New User? Dont have a Login? Request one?](#)

last reviewed: 19 January 2008

Wivenhoe Park, Colchester CO4 3SQ, United Kingdom. Tel +44 (0) 1206 873333
© Copyright 2008 University of Essex. All rights reserved.

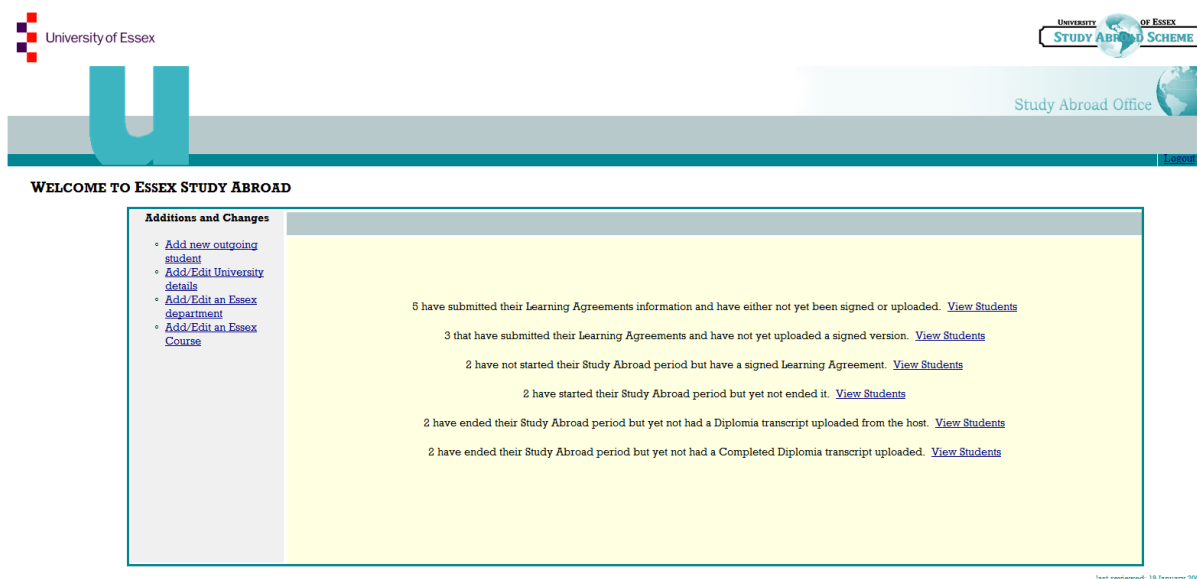
2.2.2.2 Main Administration page:

Following my initial design for the admin pages, the main body consist of two areas. A grey navigation area and an overview viewing area(Shown below).



I changed the colour scheme but kept to the original Navigation bar, head and overview layout. The navigation bar contains all the links that direct to pages where the Administrators can make minor changes to the database.

The main view originally had two different sections, one with the amount of entries for that stage and one for the text line. Instead I have one line per stage of the process with a dynamic label with a hyperlink. These hyperlinks all forward to pages with the exact same layout that provide the functionality to upload or check a stage of a process for students. (Larger screenshot can be seen in Appendix B)



2.2.2.3 Student pages:

The screenshot shows a web application interface for the University of Essex Study Abroad Office. At the top, there is a header with the University of Essex logo and name on the left, and 'Study Abroad Office' with a globe icon and a 'home' link on the right. Below the header is a navigation bar with a '[DATE]' placeholder. The main content area is divided into two columns. The left column, titled 'Application', contains a list of sections: 'Personal details', '> Contact person in case of emergency' (which is bolded), 'Academic information', 'Housing', 'Contact person at home university', and '...'. At the bottom of this column is a 'Submit' button. The right column, titled 'Application overview', contains a large dashed rectangular box. Inside this box, text reads: 'This area will contain the forms for students to fill in their information e.g. A label and a entry box and submit button. For each section to the left.'

Similarly, the student pages all follow the same Application, Navigation bar, Head and Body design (shown in the initial design above).

The navigation bar contains the headers for the different sections of the application process forms. These change to bold when depending on which section you are on.

The one alteration I made was to move the Submit button to the main application overview area. This was for many reasons including:

- ❖ I wanted more than one button.
 - One for going back to a previous section.
 - One for going to the next section once that section was completed.
 - An extra one for the additional functionality of saving the information and allowing for the user to quit and then return to the form at a later point. (This is a not currently functional but is a future function to be added to the system)
- ❖ Each section of the application is a separate div therefore to be able to make one visible and store each section of data; each button needs a separate functionality. Therefore there is a fresh set of buttons in each sections body.

These changes can viewed in the images on the following page.

A main student page was also added to give students access to their documents and start and end dates of their Study Abroad Programme (see section 2.3.1.3 Student Pages).

[Logout](#)**WELCOME TO ESSEX STUDY ABROAD**

Incoming Application		
<ul style="list-style-type: none">Personal DetailsAddress DetailsContact person in case of emergencyAcademic InformationHousingContact person at home University	First Name	
	Surname Name	
	Date of Birth (dd/mm/yyyy)	
	Gender	
	Nationality	
	Contact Phone Number	
	Email address	
	Please give details (if applicable) of any disability or special needs (including dyslexia or medical conditions)	
	<input type="button" value="Next"/> <input type="button" value="Next and Save"/>	

Incoming
External
Students
Page

last reviewed: 09 March 2011

Wivenhoe Park, Colchester CO4 3SQ, United Kingdom. Tel +44 (0) 1206 873333
© Copyright 2008 University of Essex. All rights reserved.[Logout](#)**WELCOME TO ESSEX STUDY ABROAD**

Outgoing Application		
<ul style="list-style-type: none">Personal DetailsAcademic InformationStudy Abroad Choices	Essex Registration Number:	
	First Name	
	Surname Name	
	Date of Birth (dd/mm/yyyy)	
	Gender	
	Contact Phone Number	
	Email address	
	Please give details (if applicable) of any disability or special needs (including dyslexia or medical conditions)	
	<input type="button" value="Next"/> <input type="button" value="Next and Save"/>	

Essex
Outgoing
Students
Page

last reviewed: 18 January 2009

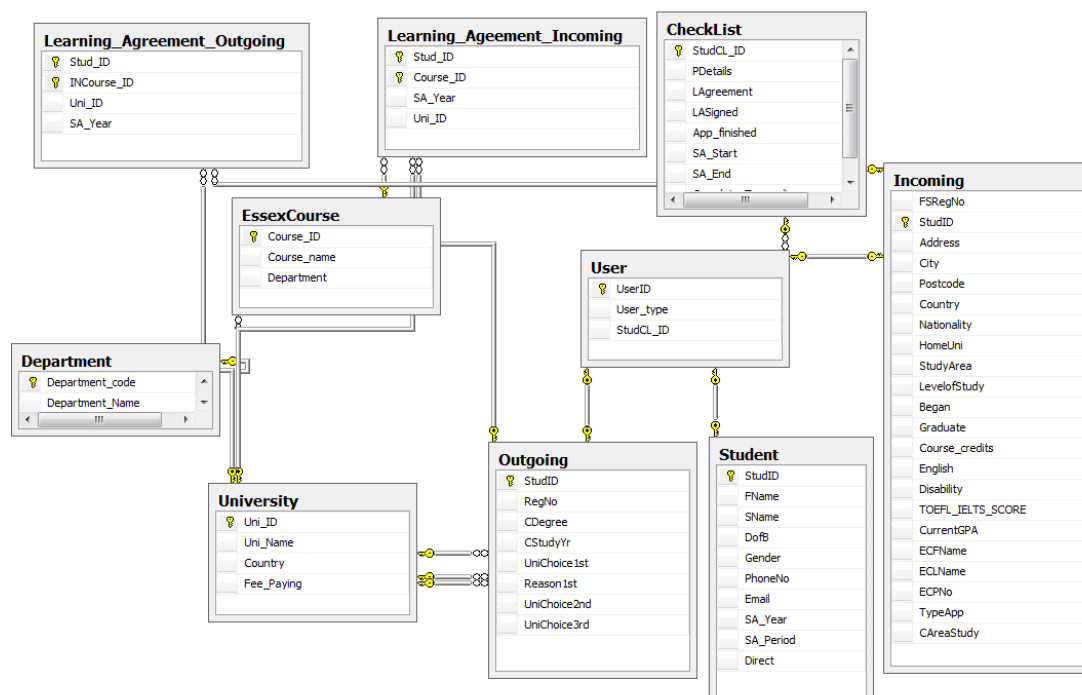
Wivenhoe Park, Colchester CO4 3SQ, United Kingdom. Tel +44 (0) 1206 873333
© Copyright 2008 University of Essex. All rights reserved.

The rest of the systems connecting pages from these Administrators and Student pages all follow the same layout design, demonstrated below.

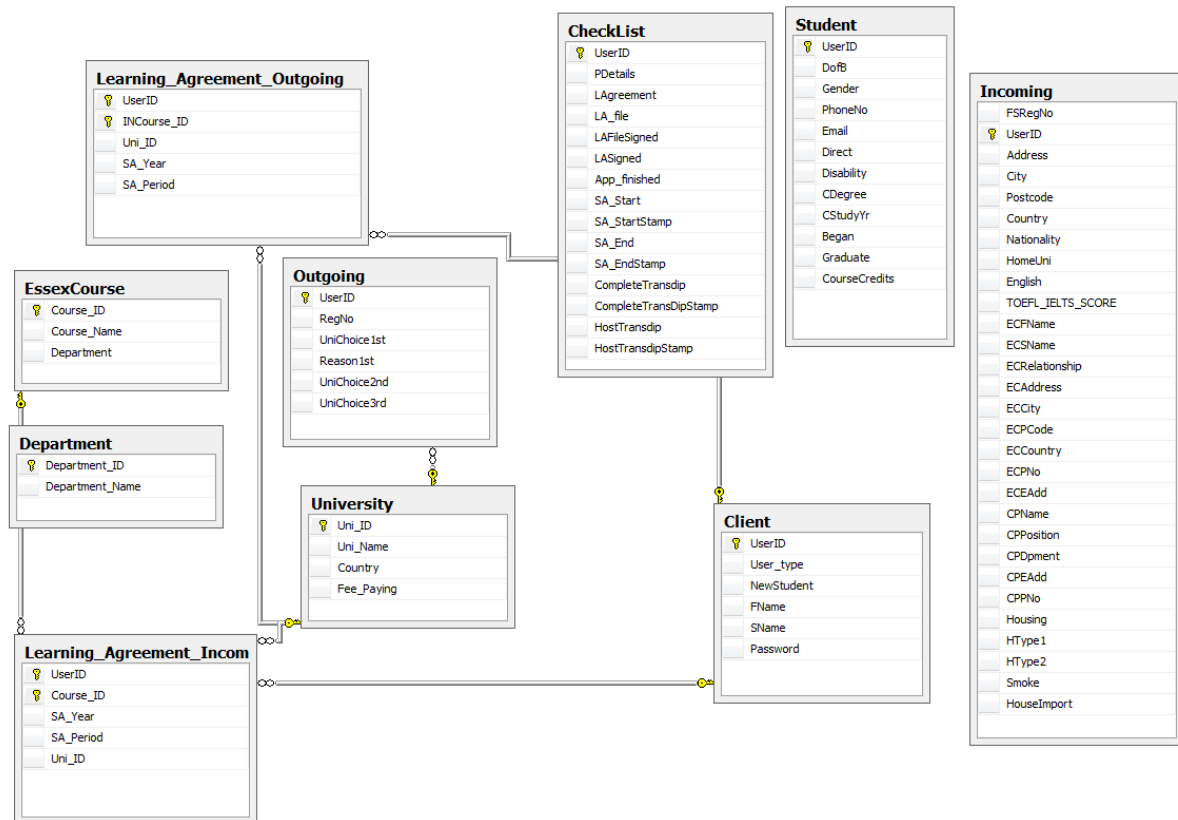


2.2.3 Database Design

A vital part of my system is the back-end database; without this no information could be saved, edited or searched, making my program irrelevant and meaningless. The structure is very important as to how data is stored as data redundancy needs to be incorporated to reduce errors e.g. duplicate users. The structure has several interlinking tables each separately or together handling different aspects of the system e.g. learning agreements or student personal details etc.



Above shows the initial design for the Entity Relationship diagram and below shows the new current one. Throughout my implementations period, my database design has altered to accommodate the needs of the project and the structures of the pages e.g. the student application page has different fields for incoming and outgoing students therefore the information that needs to be saved can increase or decrease.



Some fields had to be moved between tables depending on where was most efficient and space saving for them to be saved. For instance data that was common to both incoming and outgoing application forms were fields in the Student table e.g. current degree (CDegree) and current study year (CStudyYr).

The relationships between the client table and student, incoming and outgoing tables for the UserID foreign key were also removed. This was due to an insertion error described in more details in section 2.3.2.1.

Another minor change was the Checklist table expanded in size as more stages needed to be tracked. Fields were also created to save file names and a linking field that could be used as an indicator as to whether that stage was complete or not e.g. La_file contains file path, LAgreement is 0 or 1 depending on if the stage is complete or not.

For how the database was implemented into the system refer to sections 2.3.1.1.

2.3 Implementation

Over the past year I have been planning and designing a system; Over the past 6 months I have been implementing these designs in relation to the requirements created from the specification the customer provided.

The main requirements and how I achieved them in relation to functionality were as follows: (All specification shown in bold below are Referenced from [9])

R.3.2.3 The system should provide appropriate functionality to support a medium for study abroad programmes within Universities.

R.3.2.4 The system should easily catalogue International programmes maintainable.

The student pages and administrator pages incorporated together allow for initial data entry, and for both the students and administration are able to track the progression of each Study Abroad programme.

This means when transcripts and agreements are transferred between universities they can be tracked, checked and logged.

The Administration page allows for International programmes to be catalogued and organised depending on what stage they are at e.g. students who have a Learning agreements without approved signatures or have finished their study abroad but have had no transcript from their host university (see R.3.2.7 and R.3.2.8 for more details).

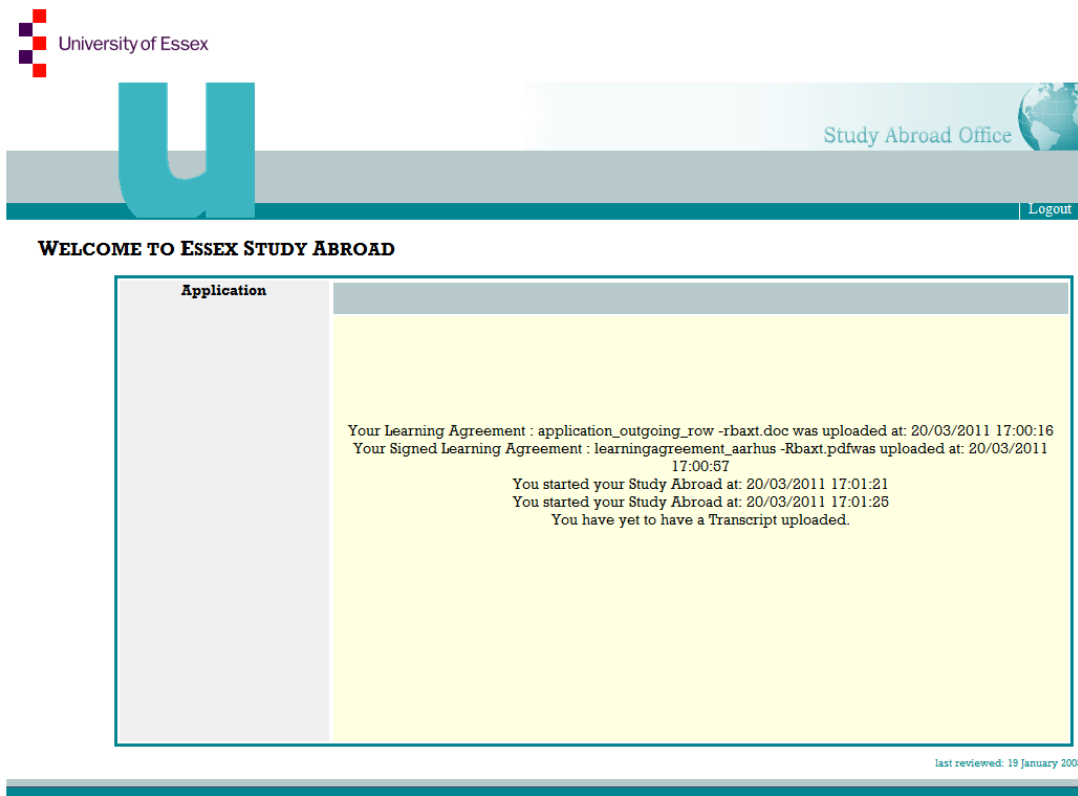
R.3.2.5 Students shall be able to complete an online application process for period of study abroad whether incoming or outgoing.

Students log in and are directed to one of three pages:

- ❖ Incoming student Entry page- This page is for students who are not from Essex University and have not entered information relating to their personal details and application to study abroad. The page consists of several entry forms to gain the required information.
- ❖ Outgoing student Entry page- This page is for students who are from Essex University and have not entered information relating to their personal details and application to study abroad. The page consists of several entry forms to gain the required information.

(See design section for images of these application pages).

- ❖ Main student page- If either an incoming or outgoing student has filled in and submitted all the required information for a learning agreement to be made; they are then directed to a main page where they can check the progress of their study abroad and view the documents when uploaded by the Administration. (Shown below)



University of Essex


Study Abroad Office

Logout

WELCOME TO ESSEX STUDY ABROAD

Application
<p>Your Learning Agreement : application_outgoing_row -rbaxt.doc was uploaded at: 20/03/2011 17:00:16</p> <p>Your Signed Learning Agreement : learningagreement_aarhus -Rbaxt.pdfwas uploaded at: 20/03/2011 17:00:57</p> <p>You started your Study Abroad at: 20/03/2011 17:01:21</p> <p>You started your Study Abroad at: 20/03/2011 17:01:25</p> <p>You have yet to have a Transcript uploaded.</p>

last reviewed: 19 January 2008



WELCOME TO ESSEX STUDY ABROAD

Please Login:


User Name:

Password:

☐ Remember me next time.

[New User? Dont have a Login? Request one?](#)

An Incoming student however does not initially have an Essex login username and password to be forwarded to these pages. Therefore a 'New User Page' was created that can found via the link on the login page(Shown left). Here the users can send their name and email address etc and request login details, then sent to a confirmation page (shown below).



University of Essex



Study Abroad Office

WELCOME TO ESSEX STUDY ABROAD

Please Login:

First Name:

Surname:


University:

Email Address:


last reviewed: 19 January 2008

Wivenhoe Park, Colchester CO4 3SQ, United Kingdom. Tel +44 (0) 1206 873333
© Copyright 2008 University of Essex. All rights reserved.

(Entry page above)



University of Essex



Study Abroad Office

Thank you, we will send you your username and password shortly to your provided email address.

When received please follow the usual login process.

last reviewed: 19 January 2008

Wivenhoe Park, Colchester CO4 3SQ, United Kingdom. Tel +44 (0) 1206 873333
© Copyright 2008 University of Essex. All rights reserved.

(Confirmation page above)

At this present time these are only dummy pages, therefore the information is not forwarded anywhere(see section 4.1 for further details).

R.3.2.6 Student application information will be stored with the general student's information e.g. Name, Date of Birth, nationality etc. Extra information would be necessary if an incoming student.

As stated previously, each student has to go through an application process where they enter information into a preconfigured set of forms. This data will be stored in the Back-end database in a set of linked tables (See Design Section 2.2.3).

I created this database in SQL Server 2008, however I later found you could make it directly in Visual Studio 2010. This would be easier for transport purposes between machines if needed, as it keeps it internal and linked directly to the systems front side web interfaces.

However, as I had already created an example database in SQL server in my design stage that was complex so I continued this way, using an ODBC connection to link to the database to the Visual Studio Web pages.

R.3.2.7 Administrators for the International office should be able to receive this Application, check it and have it accepted. Then create a Learning agreement from it.

R.3.2.8 The system shall allow the administrators then to keep track of whether

- i. Applications have been accepted,
- ii. Learning agreements have been signed and by what parties.
- iii. Whether the study abroad has started and when it has ended.
- iv. Also has the Host University sent a transcript of completion?

The main Administration page has several alerts on the first page made up of a dynamic label and a hyperlink. These state how many students there are for the different states of the Study abroad process. Each line has a count that automatically changes depending how many students are at each stage. This was achieved through SQL queries and counting variables.

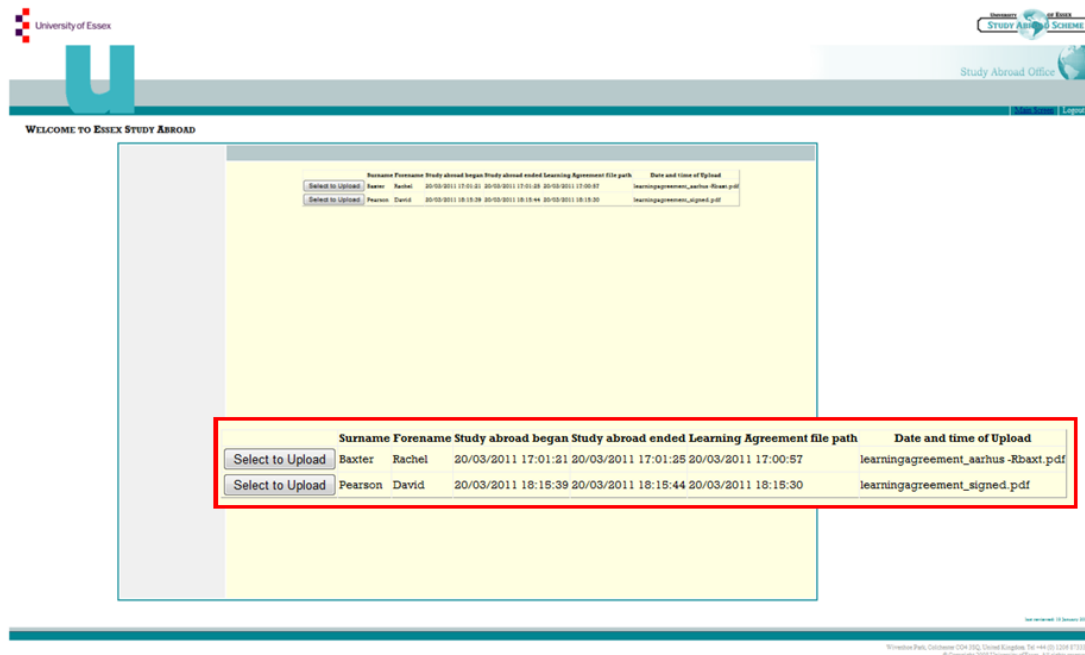
Administrators can then click on these links to see the relevant details of each student in relation to the stage e.g. the learning agreement details (university choice or dates of studying) and then upload or view the learning agreement file.

When each Hyperlink is clicked it directs to the corresponding page. This can be of two types:

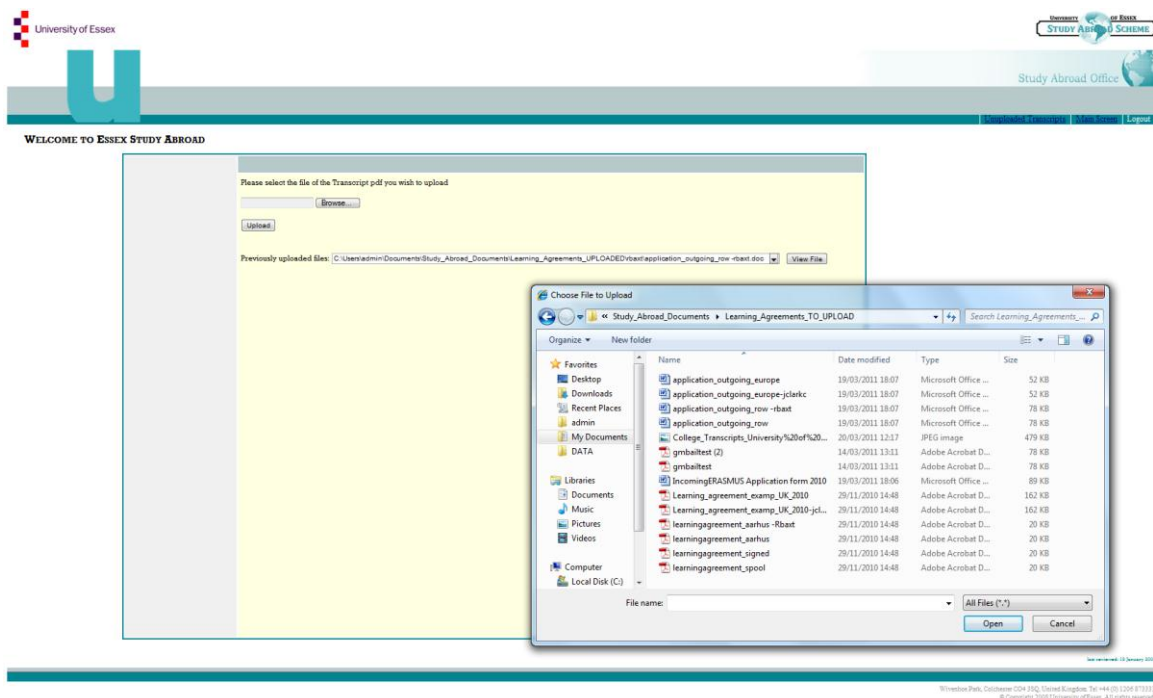
- ❖ A gridview displaying relevant Student details with a button field.
- ❖ A gridview displaying relevant Student details with a checklist field.

If there are zero students present for a stage then the main administration page is just refreshed and no further direction is taken.

The button fields refer to upload states of documents and the checklist fields to the Starting and ending stages of the Study Abroad (Below is an example for the Learning Agreement Unsigned Upload).



For the gridviews with button fields, admin can click a button relating to a certain student that then direct to a new page where they can upload a document e.g. Learning Agreement, Signed Learning Agreement or Transcript for that student. It also shows and allows them to view any previous documents uploaded for that student (Shown Below).

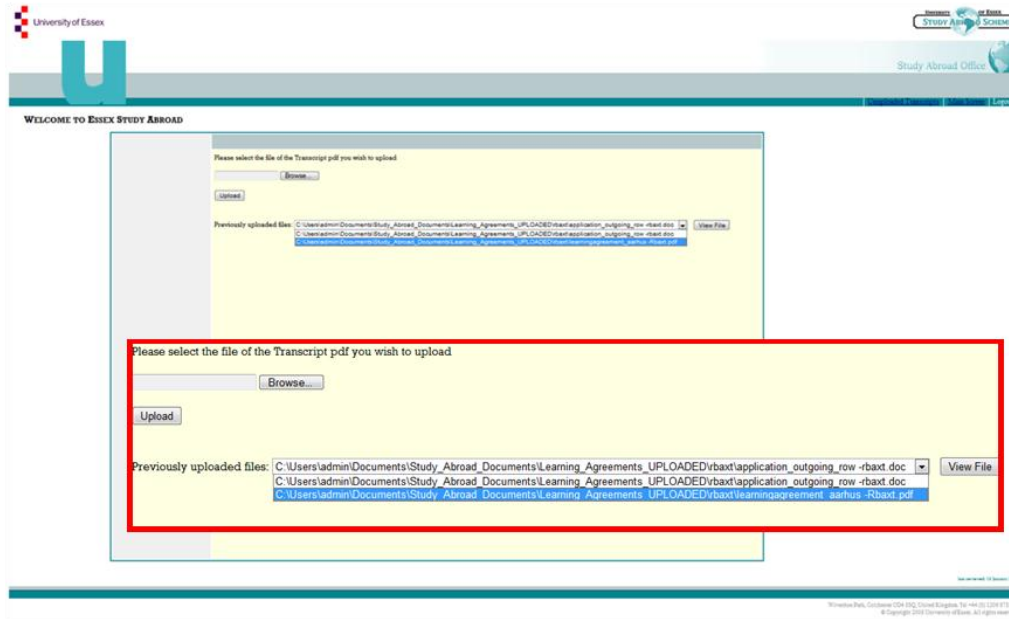


For the previously uploaded documents section to work, it needs to find the correct files for each student. Therefore, each student has a folder created for them under their username, where documents can be uploaded into. So when coding to find these files, the start of the folder pathnames is the same for each student just the end section is altered for each username e.g. each students folder (Example below :AdminLASigned.AdminLearnAgreeUserID = studentname).

```

if (!IsPostBack)
{
    PrevFiles_IN.Items.Clear();
    foreach (string s in
Directory.GetFiles("C:\\Users\\admin\\Documents\\Study_Abroad_Documents\\Learning_Agreement
s_UPLOADED\\" + AdminLASigned.AdminLearnAgreeUserID + "\\"))
    {
        PrevFiles_IN.Items.Add(s);
    }
}

```



For all uploading and viewing functions the System.IO namespace was used. This allowed basic file directory support which included reading and writing to files.

Therefore as shown in the code below, I could access each computers directory and allow the user to find a file. When a file had been found (aka FindFile.HasFile) path is saved and the document saved into the student's folder. This meant once the Previous Files list (PrevItems_IN) was cleared and reloaded the new file would appear in the list as well.

```

protected void ButUploadLA_Click1(object sender, EventArgs e)
{
    if (FindFile.HasFile)
    {
        savePath += FindFile.FileName;
        FindFile.SaveAs(savePath);
        ErrorMessage.Text = "File uploaded: " + FindFile.FileName;
        String sqlString = "Update Checklist SET LAFileSigned= '" + FindFile.FileName + "',
LASigned=GetDate(),App_finished=1 WHERE UserID='"+AdminLASigned.AdminLearnAgreeUserID + "'";
        DataTable dtMyTable = SQLConnect.sqlConnection(sqlString);

        PrevFiles_IN.Items.Clear();
        foreach(string s in Directory.GetFiles("C:\\Users\\admin\\Documents\\Study_Abroad_Documents
\\Learning_Agreements_UPLOADED\\" + AdminLASigned.AdminLearnAgreeUserID + "\\"))
        {
            PrevFiles_IN.Items.Add(s);
        }
    }
    else{
        ErrorMessage.Text = "No file uploaded";
    }
    Page_Load(null, null);
}

```

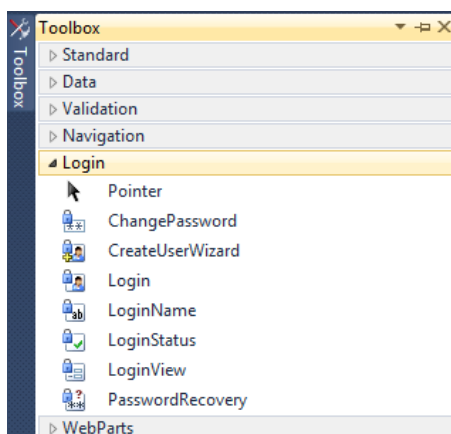
Alternatively, the tables with checklist fields work differently. They are not changed one at a time but when the submit button is pressed any student rows with a checklist box that is checked, a date and time stamp is then created and inserted into the corresponding fields/rows in the database. No further action is needed.

R.3.2.9 Access levels need to be set to restrict access so that only the data relevant to the corresponding login is accessible.

R.3.2.10 Unauthorised access needs to be prevented for the interfaces.

The system will restrict each user to only access their relevant pages via the login page. Extra programming would need to be done so that users can't maliciously spam or break their information or pages or access other student pages (see section 4.1 for more details).

For the login page I initially used Labels and textboxes to create the input fields, then manually coded in the backside of the page to check the login username and password against the database. I then altered this to use the Visual Studio provided Login function from the toolbox (below left). I did this as it's more efficient and allowed me to authenticate more proficiently, additionally incorporating MD5 for encryption and a 'remember me' function. I could then code to direct to the correct page for each user (below right).



E.g.

```
if (usertype == 0)
{
    e.Authenticated = true;
    Session["UserType"] = "1";
    Response.Redirect("Admin.aspx");
    //GO TO ADMIN
}
```

On the Login page, I also added Sessioning, (Previously explained in Section 2.1). This added control and security to who was able to view what pages. For instance, a user who is a student could not change the web URL to that of the administration's.

At the top of each page the session is checked in relation to if it has been started and the correct usertype is present (student example shown below)

```
if (Session["LoggedIn"] != "true")
{
    Session.Clear();
    Response.Redirect("~/Login.aspx");
}
if(Session["UserType"] != "1")
{
    Session.Clear()
    Response.Redirect("~/Login.aspx");
}
```

I then created a logout link at the top of each page, to then wipe these session variables and redirect to the login page (for more detail see Section 2.3.1.4).

2.3.1 Additional implementation to specification

My specification was aimed at the essential requirements necessary for my system to work consistently at a basic level. However, through my implementation I added lots of additional features/functions whether big or small, that would increase the overall capabilities and performance standard of the system.

2.3.1.1 Database

As described earlier the database can be found in SQL Server and accessed through an ODBC connection. A vast majority of the Study Abroad System involves accessing the database either to query to get a set of information or to save, update or delete information in the tables.

Originally, I would have had to code for each query to have its own Connection, sql string to send, and datatable to get back. This meant the example code below would have had to be copied, pasted and altered each time for each query.

```
SqlConnection myConn5 = new SqlConnection(@"...");
String sqlString5 = "SELECT...";

SqlCommand cmdMyCom5 = new SqlCommand(sqlString5, myConn5);
DataTable dtMyTable5 = new DataTable();
myConn5.Open();

SqlDataReader myReader5 = cmdMyCom5.ExecuteReader();
dtMyTable5.Load(myReader5);
myConn5.Close();
```

This would not be very efficient, creating more code and more work than that was necessary. Therefore, I created a class called SQLConnect and a Datatable method. Within this I copied this code and altered it so it was generic to all necessary connections to the database(shown below).

```

public class SQLConnect
{
    public static DataTable sqlConnection(String sqlString)
    {
        SqlConnection myConn = new SqlConnection(@"Data Source=CSEESLD32;Initial
                                                Catalog=SAbroadDb;Integrated Security=True");
        SqlCommand cmdMyCom = new SqlCommand(sqlString, myConn);
        DataTable dtMyTable = new DataTable();
        myConn.Open();

        SqlDataReader myReader = cmdMyCom.ExecuteReader();
        dtMyTable.Load(myReader);
        myConn.Close();
        return dtMyTable;
    }
}

```

The class could then be accessed in all the other pages via *SQLConnect.sqlConnection()* and a sqlstring could be sent to it and it would return the appropriate datatable, unless an exception is thrown.

The example below is for the Admin page where the sql string queries the database to check how many students have no signed learning agreement uploaded.

```

String sqlString = "SELECT COUNT(UserID) FROM Checklist WHERE LASigned IS NULL";
DataTable dtMyTable = SQLConnect.sqlConnection(sqlString);

```

This practice I used throughout my project for all database connections.

2.3.1.2 Administrator pages

The Administrators pages main aim was to provide functionality for tracking student's progress through the stages of the Study Abroad Process. This means managing Learning agreements and the course of a student's period abroad. I achieved all this via the main page (described before in Design section 2.2.2.2 and Implementation section 2.3 requirements 3.2.7 and 3.2.8).

As previously described, there is the extra functionality that the Administrator could view the Uploaded documents, open and save them.

Another extra detail I added was the option of the Administration making minor database changes (available as featured on the administration page shown below). This means they did not need to bother the technical staff for little changes.

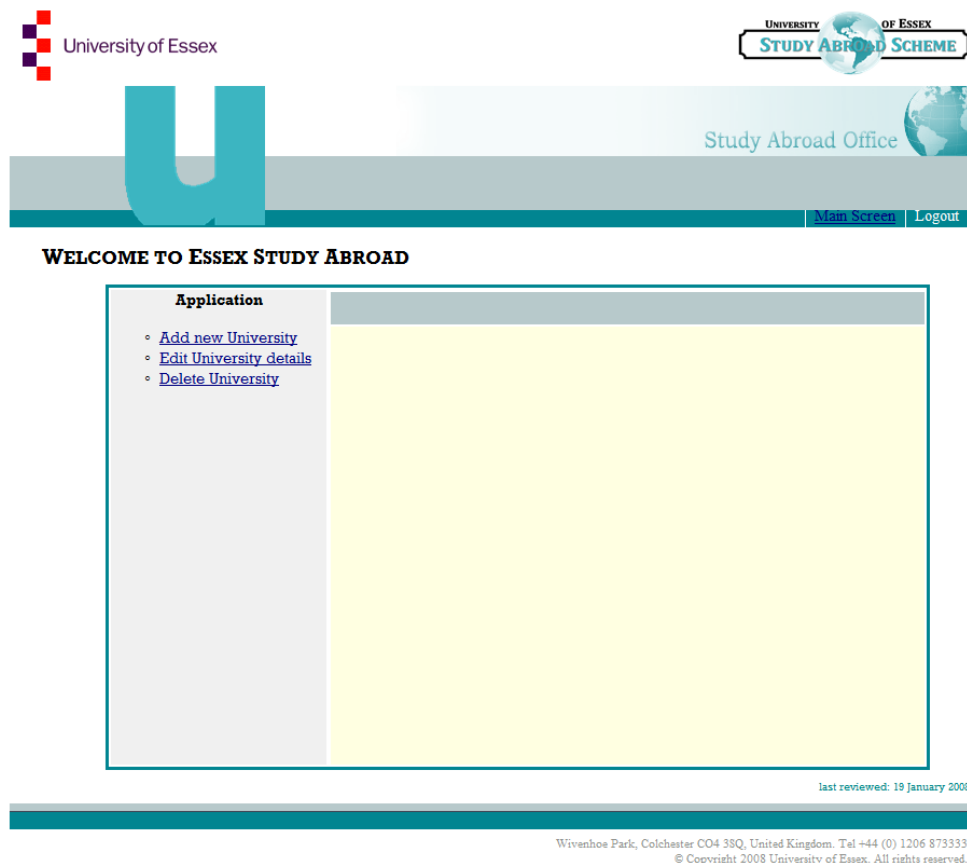
The screenshot shows a web application interface. On the left is a sidebar menu titled "Additions and Change" with a red border. It contains four links: "Add new outgoing student", "Add/Edit University details", "Add/Edit an Essex department", and "Add/Edit an Essex Course". The main content area has a light yellow background and displays a list of student status reports. Each report line includes a count, a description of the student's status, and a "View Students" link.

Count	Status Description	Action
5	have submitted their Learning Agreements information and have either not yet been signed or uploaded.	View Students
3	that have submitted their Learning Agreements and have not yet uploaded a signed version.	View Students
2	have not started their Study Abroad period but have a signed Learning Agreement.	View Students
2	have started their Study Abroad period but yet not ended it.	View Students
2	have ended their Study Abroad period but yet not had a Diploma transcript uploaded from the host.	View Students
2	have ended their Study Abroad period but yet not had a Completed Diploma transcript uploaded.	View Students

Therefore, I provided the following options:

- ❖ Student Main with the following options:
 - Add a new Essex student (new outside users will be handled a different way)
 - Edit a student's Basic Information
 - Delete a student from the Client table.
- ❖ University
 - Add a university to the program
 - Edit a universities details
 - Delete a university from the Program.
- ❖ Department
 - Add an Essex department
 - Edit an Essex department details
 - Delete an Essex department.
- ❖ Course
 - Add an Essex course
 - Edit an Essex course details
 - Delete an Essex course.

All these pages follow the same layout design consistent with the rest of the system and like the Admin page all hyperlinks (add, edit and delete) remain in the navigation bar.



Shown above is the initial page you see when you click one of the links. This is considered to be the main page for each type of database modification. These examples are for University.

All the changes involve one of three types of sql query, an Insertion, update or deletion. For the edits to make a change you must provide the code for the course, student, department or university then edit the other relevant details. This however meant that you couldn't change the code only its details. (This is further discussed in section 4.1).

The screenshot shows the 'University of Essex' logo and 'STUDY ABROAD SCHEME' header. Below is a 'Study Abroad Office' banner with a globe icon. A navigation bar contains links: 'University Main', 'Main Screen', and 'Logout'. The main heading is 'WELCOME TO ESSEX STUDY ABROAD'.

The 'Application' section is divided into two columns. The left column contains links: 'Add new University', 'Edit University details', and 'Delete University'. The right column contains a form titled 'Please enter the details you wish to change:'. The form fields are: 'University Name:' (text input), 'Country of University:' (text input), and 'Will the Students need to pay extra fees?' (radio buttons for 'Yes' and 'No'). A 'Submit' button is located below the form.

At the bottom right of the form area, it says 'last reviewed: 19 January 2008'. At the very bottom, there is a footer with the address 'Wivenhoe Park, Colchester CO4 3SQ, United Kingdom. Tel +44 (0) 1206 873333' and copyright information '© Copyright 2008 University of Essex. All rights reserved.'

The task of adding and editing students was a little more difficult as the passwords in database are in MD5 format. Therefore using the code similar to that on the login page, I converted the password entered by the user to MD5 format before inserting into the database. This was done for both the add and edit functions.

2.3.1.3 Student pages

For the student page a main student page was added. There are two ways a student can access this page and they were:

- ❖ Logging in and automatically being redirected to the page.
- ❖ Being directed to the application stage, filling you details in and then being directed.

Once they are able to view the student main page the user will see a list of statements. Initially, it was made so these were dynamic labels that would tell the user if their documents had been uploaded or the dates of finishing or starting their program. If not it would just inform them the contradiction e.g. You have yet to have a learning agreement uploaded or you have not yet started your study abroad etc.

I advanced on this to make the learning agreement labels also hyperlinks, to allow the students to click and view their documents and save them if necessary.

Therefore I split each normal text link into three sections. Two labels with a hyperlink in between. My first difficulty was that I had to connect the function to the Navigation link.

```
<asp:Label class="labelsNorm" ID="LA" runat="server" Text=" "></asp:Label>
<asp:HyperLink ID="LALink" runat="server" NavigateUrl="~/StudentFiles/ViewDocLA.aspx"
Text= " "> </asp:HyperLink>
<asp:Label class="labelsNorm" ID="LA2" runat="server" Text=" "></asp:Label><br />
```

These lines were still created dynamically as before but this time the hyperlink text was the file name for either the normal or signed Learning agreement or one of the transcripts. Each hyperlink then navigated to its one page, similarly to the Logout page which were blank excluding the background code. This code is similar to the upload pages except it simply takes the filename, adds it to the end of the standard pathname and then opens it up, then redirects back to the student page.

Example code below is for the normal Learning Agreement and a view of the student main.

```
public partial class ViewDocLA : System.Web.UI.Page
{
    public static string savePath = "";
    public static string SSelectedFile = "";
    public static string Document = "";

    protected void Page_Load(object sender, EventArgs e)
    {
        savePath="C:\\Users\\admin\\Documents\\Study_Abroad_Documents\\Learning_Agreements_UPLOADED\\\" + Login.UserName + "\\\";
        Document = savePath + StudentMain.LAFile;
        StudentMain.Alert = StudentMain.LAFile;
        ButPrevUpload(Document);
        Response.Redirect("/StudentFiles/StudentMain.aspx");
    }

    protected void ButPrevUpload(object sender)
    {
        SSelectedFile = Document;
        System.Diagnostics.Process.Start(SSelectedFile);
    }
}
```

University of Essex

Study Abroad Office

Logout

WELCOME TO ESSEX STUDY ABROAD

Application	
	Your Learning Agreement : application_outgoing_row-rbax.doc was uploaded at: 20/03/2011 17:00:16 Your Signed Learning Agreement : learningagreement_aarhus-Rbax.pdf was uploaded at: 20/03/2011 17:00:57 You started your Study Abroad at: 20/03/2011 17:01:21 You started your Study Abroad at: 20/03/2011 17:01:26 You have yet to have a Host Transcript uploaded. You have yet to have a Home Transcript uploaded.

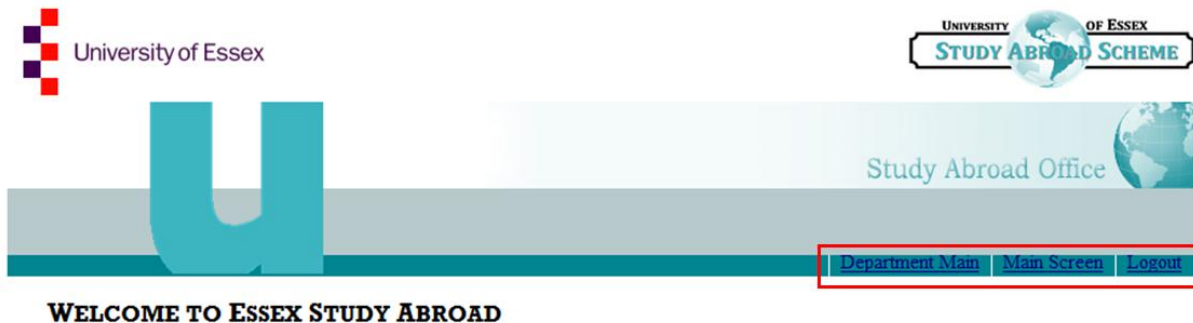
last reviewed: 19 January 2008

2.3.1.4 All pages

As all pages were near exact in layout and formatting, I was capable of adding a few little extra features.

Each page has a navigation menu strip as part of the header banner. This meant helpful links for the user could be included.

Each page consisted of a link for logging out and for the administration pages a link to go back to the main Admin page (the one with the alerts on) and/or back to the main page for that part of the system e.g. Main signed upload gridtable view page or main page for department changes etc Examples can be seen below.



The logout button emerged to be more difficult than first anticipated as it wasn't a simple redirect to another page as I also had to consider my sessioning. Therefore, I needed to attach a method to the logout link that when clicked cleared the session. So, I used the same process that was used for the main student page which allowed them to view their documents.

The URL of the logout hyperlink were set to direct to the logout page. This page was blank except for the hidden functionality behind which cleared the session then redirected back to the login page.

```
public partial class Logout : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Session.Clear();
        Response.Redirect("~/Login.aspx");
    }
}
```

2.3.2 Problems encountered

Creating this system has been a learning curve and like any other projects nothing works perfectly first try and there are challenges to overcome. Using my log book I was able to document these and how I resolved them.

2.3.2.1 Database problems

The back end database caused few errors, however there was one major issue that initially pended from being able to use it at all. Initially, I could not access my files that were created on SQL server 2008 to be able to transfer and continuously work on them. This was caused by a permission error caused by the university technicians. Unfortunately they were unable to change this, so I avoided this completely by obtaining an External hard drive and becoming the admin therefore giving myself permission. This meant I could backup and transfer my work more easily, without continuously having to reload a script to allow changes.

I also changed some of my original number (int) fields to string (varchars) because when inserting into the database if they were empty it caused an error to be thrown. However, I wanted some of these to be allowed to be null. Therefore, to confirm they are numbers being entered, the validation on entry to the form before submitting will verify this(to be added later).

The database has relationships that match data in key fields to one another. These are usually primary keys in one and become the foreign key in another. The main purpose of this is to enforce a link between the data to gain control; enforcing referential integrity ensuring changes are validated before made.

The student application pages consist of inserting data not only into the main Student table but related incoming, outgoing checklist tables etc. This caused an error due to the fact that the primary key in the User table and then this key is made into a foreign key in other tables. So when inserting it would throw an error because it could not verify the insert, if the primary key did not already exist as a foreign key in the other table. Therefore to solve this issue, I removed the relationship and verified the key to be correct and in the user table. This could be supported because the UserID is gained and saved when the user logs in. So, if the login is validated so is the UserID.

Also SQL server classed user as restricted word therefore the table was not being recognised. So I renamed the User table to Client instead.

2.3.2.2 Interface problems

The main issues that I came across were to do with the implementation of my webpage's. To organise the layout I used an External Style Sheet (containing CSS). This initially differentiated between browsers (e.g. IE and firefox). The header and banner misalignments were fixed by altering the layout via the css, making the positions relative, manually setting heights and widths for some objects and adding margins and padding.

2.3.2.2.1 Student page

When it came to the functionality there were several barriers that had to be overcome. The student application pages consist of several forms that collect data and navigate back and forth between them. Each part of the form is a Div made up of labels and textboxes connected with its corresponding title label in the navigation bar section.

The first problem occurred when I went from one page to another then reloaded the page without completing the forms or entering with a different user, it still remained on the same page. Therefore, I needed to make it so the first pages visibility was always true (so displayed) and all other div's were not visible. This corrected the problem but then the page was stuck on that page and would not forward. This was because it was making the first div the current one on refresh and not just when you first entered the application process. Thus it needed this to occur only when the page was not Postback. This was achieved through the code below.

```
if (!IsPostBack)
{
    appBodyHL1.Visible = true;
    appBodyHL1.Attributes.Add("style", "visibility:visible");
    appHL1.Attributes.Add("style", "font-weight:bold");
}
```

After these problems were solved I coded for the pages to go back and forth with a set of buttons separate to the divs that related to them all as a group. This did not work as the buttons would only

execute once. So go back and forth between two pages(/div's). To solve this problem I made each div have its own set of buttons that when pressed made that div invisible and its linked header font to normal instead of bold. If it was the back button the previous div would be made visible and header bold or the next button, the same with the next inline div. This was correct for all pages excluding the first and last pages.

For the fourth div that handled housing, there was the question 'Do you want to request housing or not?' with a yes or no answer. Depending on what they answered extra code was added to forward them to the housing page or to skip that and move onto another page.

The Next button on the final page would forward the User to the main student page but firstly should save the information the user has entered into the Study abroad database. There are three insert queries and Initially one or more could fail and the others could still succeed and occur in the database. This would make my database faulty as it would not include all the correct information and if retried would fail as the UserIDs would already exist. Therefore all three of these queries are within the same try and catch sql exception block. So if one fails, they all fail. Shown below is edited and generalised example to represent the process.

```
try
{
String sqlString = "Insert INTO Student (UserID,...) VALUES ('"+ Login.UserName +...+ "'");
DataTable dtMyTable = SQLConnect.sqlConnection(sqlString);
String sqlString1 = "Insert INTO Incoming(UserID,...) VALUES ('"+ Login.UserName +...+ "'");
DataTable dtMyTable1 = SQLConnect.sqlConnection(sqlString1);
String sqlString2 = "Insert INTO CheckList(UserID,PDetails) VALUES
('"+Login.UserName+"',1)";
DataTable dtMyTable2 = SQLConnect.sqlConnection(sqlString2);
upload++;
}
catch (SqlException)
{
ErrorMessage.Text = "Database connection failed, incorrect data";
}
```

There was also the connecting problem that the user was redirected without correct insertion. Therefore, I used a variable that would be initially defined as 0 and only made 1 if the try was successful. Then using a simple If state to check if it was 1 and if it was redirection could occur.

Once this was all tested to be correct and working, I found that it was continuously redirecting to the login page. The cause was found to be due to sessioning. In the login page if the student has completed the application page they get usertype 1, if they have not they get 2 or 3 depending on if they are incoming or outgoing. However, once they have completed the application process they still have the usertype 2 or 3 so get rejected by the main student page. In the if statement before forwarding, I set the session usertype to 1 resolving this problem. (The code below shows the explanations above)

```
if (upload == 1) {
StudentFiles.StudentMain.Alert = "You have successfully uploaded your details";
Session["UserType"] = "1";
Response.Redirect("StudentMain.aspx");
}
```

For both the document links on the main student page and the logout links on each page, I came across the identical problem of being unable to link a method to the hyperlink. I solved this by attaching empty pages to the links with just functionality. As described in the previous implementation section 2.3.1.3 and 2.3.1.4.

2.3.2.2.1 Administration pages

The administration pages consist of vast amount of coding to support all the functionality it provides to the user. The single most essential feature was for administration to view which students are at what stage and then be able to upload or check to complete this stage for them.

A major problem was that the tables of students and their data were being generated dynamically through the back .aspx.cs page and the gridview was altered accordingly.

There was an issue of how I was going to attach a button in relation to each row. Initially, I attempted an ItemTemplate Field but then there were issues connecting them to relate to each row and collect the associated data. Then, I discovered the ButtonField.

This meant an onselectchange method could be attached; however this would throw the error:

Invalid postback or callback argument. Event validation is enabled using <pages enableEventValidation="true"/> in configuration or <%@ Page EnableEventValidation="true" %> in a page. For security purposes, this feature verifies that arguments to postback or callback events originate from the server control that originally rendered them. If the data is valid and expected, use the ClientScriptManager.RegisterForEventValidation method in order to register the postback or callback data for validation.

After vast research, this was solved by using onRowCommand instead of onselectchange (see code on following page) and enabling the EventValidation to be true in the page declaration, shown below.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="AdminLASigned.aspx.cs"
EnableEventValidation="true" Inherits="StudyAbroad_0838219.AdminLASigned" %>
```

It could then be implemented to get the index number of the row selected and compare this to the datatable. This meant the relevant information as to which student was selected could be saved before redirecting to the upload pages.

Then there was trouble with hiding unwanted columns. An example of this is where for a query to work several UserID fields from different tables need to be compared. However, only one of these UserIDs needed to be shown in the gridview.

Originally, I edited the header names and which ones were visible and not visible in the page load method. This temporarily worked but not with the extra column.

I then attempted creating the columns statically in the html code using BoundFields (as shown below). However, I considered this inefficient and preferred to keep the code separately in the behind the scenes.aspx.cs.

```
<Columns>
<asp:BoundField DataField="SName" HeaderText="Surname" SortExpression="SName" />
...
<asp:BoundField DataField="UserID2" HeaderText="UserID2" SortExpression="UserID2" />
</Columns>
```

In the Page_Load i found that it allowed for the header text to be set but not the visibility of cells as it couldn't recognise them being created as the datasource and databind execution lines were in the same method. To get around this problem, an 'onDataRowBound' method was created and contained the lines to link the required desired header and rows together, making these cells invisible.

There was one last problem with this feature. If you clicked on the corresponding link on the main administration page and it redirects to this page. However, if there are no students present for this stage you see a blank page. Therefore, a line was added to check if the datatable was empty by counting the rows. If it was it would throw an exception from the try and catch block, then redirect back to the admin page, giving the impression of the page being refreshed and stopping it from getting redirected.

The following code demonstrates all these methods in action for the example of the signed learning agreement page.

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        try
        {
            String sqlString = "SELECT t1.SName, t1.FName, t5.LA_file, t5.LAgreement, t5.LASigned, t1.UserID, t5.UserID FROM Client t1, Checklist t5 WHERE t1.UserID = t5.UserID AND t5.LASigned IS null AND t5.LAgreement IS NOT null ORDER BY t1.SName ASC";
            DataTable dtMyTable = SQLConnect.sqlConnection(sqlString);

            if (dtMyTable.Rows.Count != 0)
            {
                LASignedTable.DataSource = dtMyTable;
                LASignedTable.DataBind();
                LASignedTable.HeaderRow.Cells[1].Text = " Surname ";
                LASignedTable.HeaderRow.Cells[2].Text = " Forename ";
                LASignedTable.HeaderRow.Cells[3].Text = " Learning Agreement file path ";
                LASignedTable.HeaderRow.Cells[4].Text = " Date and time of Upload ";
            }
        }
        catch (SqlException)
        {
            ErrorMessage.Text = "Database connection failed";
            Response.Redirect("Admin.aspx");
        }
    }
}

protected void LASignedTable_OnRowDataBound(object sender, GridViewRowEventArgs e)
{
    If(e.Row.RowType==DataControlRowType.Header|e.Row.RowType==DataControlRowType.DataRow)
    {
        e.Row.Cells[5].Visible = false;
        e.Row.Cells[6].Visible = false;
        e.Row.Cells[7].Visible = false;
    }
}

protected void LASignedTable_RowCommand(object sender, GridViewCommandEventArgs e)
{
    string currentCommand = e.CommandName;
    int index = Convert.ToInt32(e.CommandArgument);
    GridViewRow row = LASignedTable.Rows[index];
    AdminLearnAgreeUserID = row.Cells[6].Text;
    AdminLearnAgreeName = row.Cells[2].Text + " " + row.Cells[1].Text;
    AdminLASignedUpload.UploadName2 = "For the Student: " + AdminLearnAgreeName;
    Response.Redirect("AdminLASignedUpload.aspx");
}
}
```

When implementing the same features but with the checkbox field, I encountered more programming issues. At first, I used the ButtonField equivalent CheckBoxField. This created the checkbox for each line but proved difficult when I created my Button and the corresponding onclick() event as they would not associate with each other.

So for checkbox I reverted back to ItemTemplate field with the type of CheckBox. The onclick() submit button event could then be implemented so that several checkboxes could be clicked at once; Allowing all those students to have their study abroad programs started or ended. This saved the repetition of the task for each student individually. This was achieved via a for each loop that went through each gridview row check if checked= true; if it was then the userID was stored, a query executed to update the checklist table with a datetimestamp. The loop then continued until completed, subsequently then redirecting back to the main administration page.

The code below shows these functions in action.

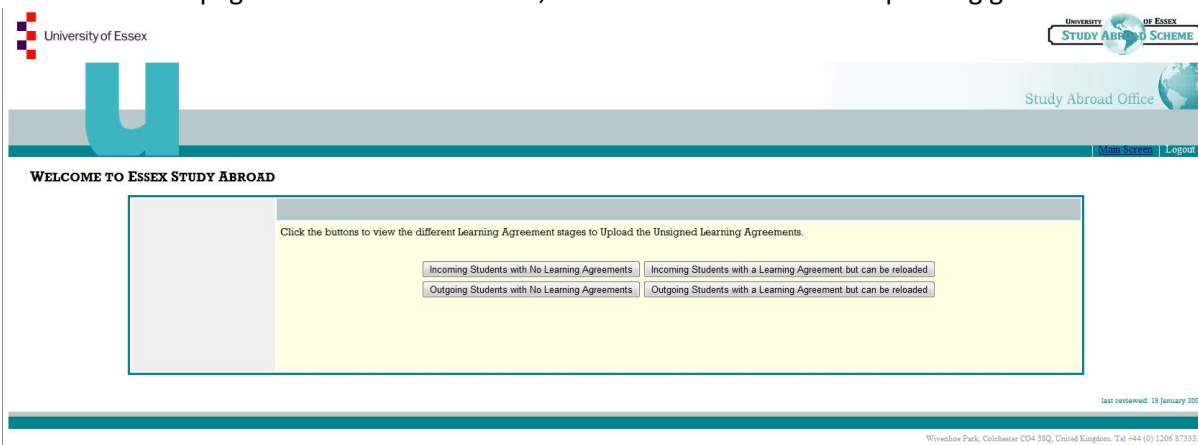
```
<asp:TemplateField HeaderText="Study abroad Started?">
<ItemTemplate>
<asp:CheckBox ID="chkSABegin" runat="server" />
</ItemTemplate></asp:TemplateField>

protected void ButSABegin_Click(object sender, EventArgs e)
{
int i =0;
foreach (GridViewRow row in SABeginTable.Rows)
{
if (((CheckBox)row.FindControl("chkSABegin")).Checked == true)
{
StartUserID = row.Cells[6].Text;
String sqlString = "UPDATE Checklist SET SA_Start =1,SA_StartStamp=GetDate()
WHERE UserID='" + StartUserID + "'";
DataTable dtMyTable = SQLConnect.sqlConnection(sqlString);
i++;
}
}
Admin.AdminAlertV = i + " Study Abroad Programmes were successfully started";
Response.Redirect("~/Admin.aspx");
}
```

Other minor issues were the following:

- ❖ The queries for finding students who had or haven't had their learning agreements uploaded involve checking whether one of the fields in checklist is empty or not. However, =null is not recognised so IS NULL has to be used instead.

- ❖ Incoming and Outgoing students have different entry fields for their learning agreements. Therefore they each need a separate gridview for uploading and another encase you wish to re-upload a learning agreement. I solved this issue by creating an intermediate stage from the main administration page which has four buttons, which forward to the corresponding gridview.

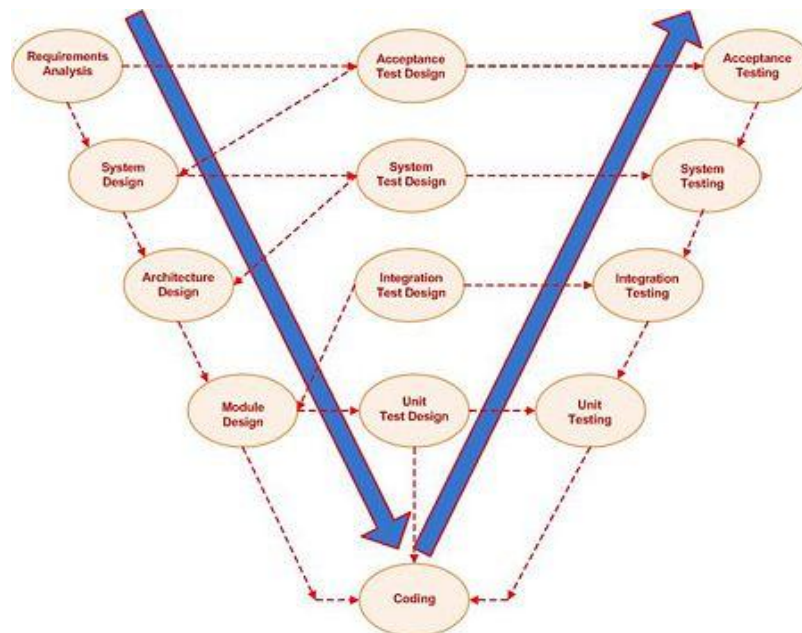


❖ Initially, when opening a previously uploaded file, it opened the first one of the list only. This was because the page is refreshed on the upload to view. Therefore, the initial prevUploadList only needed to reload when it was not Postback. However, this then did not add the new file to the list when uploaded, this was due to a PrevList.clear() line after the upload. I removed this and everything worked.

2.5 Testing

Testing is an important part of any project cycle, as it examines the quality of the product, ensuring that it meets the business and technical requirements set by the employers and functionally works as expected. Testing allows you to find the fault earlier, reducing the time and price to fix it.

For my project I followed the development process of the V-Model. (As stated from [11]) The V-Model is an extension to the standard Waterfall model where the process steps curve upwards after the coding phase to hence form a V shape. The V model demonstrates the relationships between each phase of the development life cycle and its associated phase of testing. (shown below)



[11]

This meant that as I progressed through the implementation stage I could test each section as I went. For these testing phases, several different methods of testing were used to get an overall conclusion on the reliability of the system.

2.5.1 Unit Testing.

As taken from [12] Unit testing is a method where software is broken into smaller testable sections of code, isolated from the rest of the code and tested to determine if it behaves how it is expected to.

Here is an example of one of the code coverage test cases I performed showing a section of code (a unit) and all the possible flow paths in the method. For each flow path there are keys and values that can be used to test and see if the results are of expected.

Login Process:

```

String sqlString = "Select FName, SName, User_Type, NewStudent FROM Client where
UserID = '" + UserName + "' AND Password = '" + Password + "'";
DataTable dtMyTable = SQLConnect.sqlConnection(sqlString);

A      if (dtMyTable.Rows.Count != 0)
      {
          string PersonName = (string)dtMyTable.Rows[0][0] + " " +
(string)dtMyTable.Rows[0][1];
          int usertype = (int)dtMyTable.Rows[0][2];
          NewStudent = (int)dtMyTable.Rows[0][3];
          Session["LoggedIn"] = "true";

B          if (usertype == 0)
          {
              e.Authenticated = true;
              Session["UserType"] = "1";
              Response.Redirect("Admin.aspx");
              //GO TO ADMIN
              }

C          else if (usertype == 1)
          {
              String sqlString2 = "Select UserID, PDetails FROM CheckList WHERE
D              UserID = '" + UserName + "'";
              DataTable dtMyTable2 = SQLConnect.sqlConnection(sqlString2);

              if (dtMyTable2.Rows.Count == 0)
              {
                  PDComplete = 0;
              }

E              else
              {
                  PDComplete = (int)dtMyTable2.Rows[0][1];

F              }

              if(PDComplete == 1){
                  e.Authenticated = true;
                  Session["UserType"] = "1";
                  Response.Redirect("/StudentFiles/StudentMain.aspx");
                  //Go to student Main Page
                  }

G              //GO TO STUDENT ENTRY PAGES
              else if (NewStudent == 1)
              {
                  e.Authenticated = true;
                  Session["UserType"] = "2";
                  Response.Redirect("/StudentFiles/IncomingStudent.aspx");
                  }

H              else if(NewStudent ==0)
              {
                  e.Authenticated = true;
                  Session["UserType"] = "3";
                  Response.Redirect("/StudentFiles/OutgoingStudent.aspx");
                  }

I              }

J              }

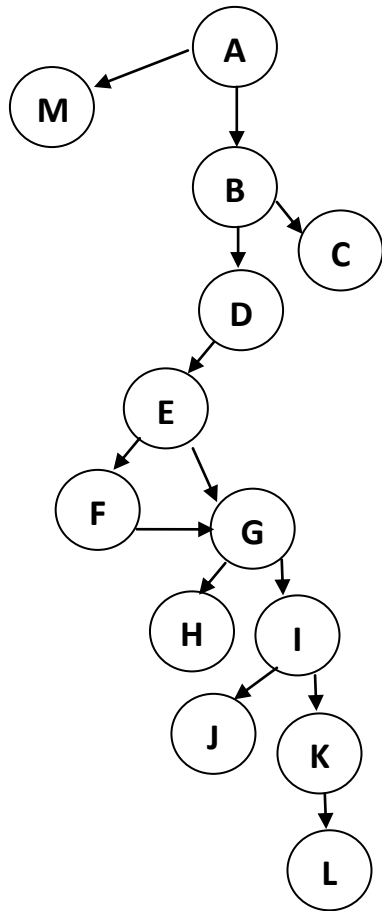
K              }

L              }

M              }
          else
          {
              e.Authenticated = false;
          }
      }

```

Flow graph:



Paths:

 $A \rightarrow M$ $A \rightarrow B \rightarrow C$ $A \rightarrow D \rightarrow E \rightarrow I \rightarrow J$ $A \rightarrow D \rightarrow E \rightarrow K \rightarrow L$ $A \rightarrow F \rightarrow G \rightarrow H$

User and Password table linked with the Checklist table. Used for Keys and Values in the test cases.

	Client table					Checklist
UserID	User_type	NewStudent	FName	SName	Password	PDDetails
Aking	1	NULL	Amanda	King	21232f297a57a5a743894a0e4a801fc3	NULL
dgpear	1	1	David	Pearson	1828baf0209ea35eb09fd75b9a0887f9	1
gmbail	1	0	Grahmn	Bailey	decf08a21c8202088e54aac5cc57a65a	1
jclarkc	1	0	James	Clarke	decf08a21c8202088e54aac5cc57a65a	1
jhaswe	1	1	Joanne	Haswell	1828baf0209ea35eb09fd75b9a0887f9	1
kbaile	1	0	Karis	Bailey	decf08a21c8202088e54aac5cc57a65a	1
lkerby	1	0	Lee	Kerby	decf08a21c8202088e54aac5cc57a65a	1
msherd	1	1	Mitchel	Sheridan	1828baf0209ea35eb09fd75b9a0887f9	1
rbaxt	1	0	Rachel	Baxter	decf08a21c8202088e54aac5cc57a65a	1
sphelp	1	1	Sam	Phelps	1828baf0209ea35eb09fd75b9a0887f9	1
spool	1	1	Sam	Pool	1828baf0209ea35eb09fd75b9a0887f9	1
sthomp	1	0	Sam	Thompson	decf08a21c8202088e54aac5cc57a65a	1

Test cases:

Path	Test case	Expected result	Achieved Result
A→M	Key: jgarret Value= password Keys:[users] Values:[passwords]	Authentication fail. Remains on Login page, error message is shown.	Correct
A→B→C	Key: aking Value= admin Keys:[users] Values:[passwords]	Saves session. Redirects to Administration page.	Correct
A→D→E→I→J	Key: ldwit Value= newstudent Keys:[users] Values:[passwords]	Saves session. Redirects to Incoming entry page.	Correct
A→D→E→K→L	Key: scox Value= essexstudent Keys:[users] Values:[passwords]	Saves session. Redirects to Outgoing entry page.	Correct
A→F→G→H	Key= sthomp, msherd, kbaile, sphelp, gmbail, jhaswe, jclarkc, spool, rbaxt, dgpear Value= essexstudent, newstudent, essexstudent, newstudent, essexstudent, newstudent, essexstudent, newstudent, essexstudent, newstudent, Keys:[users] Values:[passwords]	Saves session. Redirects to main student page.	Correct

2.5.2 Integration testing

Each section of the system could be tested individually however many of the components are linked. This is where Integration testing comes in beneficial.

For integration there are two main approaches (described and taken from[13]): Top-Down, Bottom-Down. There is another known as the Umbrella approach; this involves testing along any functional data and control flow path.

For my system I followed the Bottom-Up approach. The study Abroad system was split up into several sections:

- Login page
- Incoming page
- Outgoing page
- Main Student page
- Administration Main page
- Administration database changes pages:
 - Student
 - Department
 - Course
 - University
- Checklist stages:
 - Administration Learning agreement entry page buttons x4 and upload page
 - Administration Signed Learning agreement gridview and upload page
 - Administration start study abroad page
 - Administration end study abroad page
 - Administration Transcript gridview and upload page
- Header bar links

Login is explained in Unit testing.

For the Incoming and outgoing entry pages I tested by entering dummy data into the fields and checking that they not only entered into the database but when on the main administration page added the entries to the first alert of no Learning agreement uploaded table.

Concerning the Main student page there were two elements that needed to be tested. Firstly that each student could see whether they had the relevant documents uploaded or not. If so they could view them by selecting them. Secondly that each type of user got directed to the correct page. Through testing I found that if a student entered in data into the application pages when they were finished they were not directed to the student page. This problem is discussed in 2.3.1.3.

The main administration page functionality was tested concerning if a stage was completed or not completed the numbers increased, decreased or were at zero. This then linked with all the gridview, checklist tables ensuring that if one student has one stage completed they appear in the next stages table e.g. if their study abroad is started they should then be in the study abroad ended table. Each stage of upload e.g. learning agreement signed or unsigned and transcripts needs to be tested that:

- The files correctly
 - Upload into the correct folders.
 - Are the correct files
 - The message on the page states it has been uploaded
 - Path is added to database.
 - If a file cannot be uploaded an error message is shown
- All previously uploaded files are in the drop down list
- All can be opened on the press of the button
- When a file is uploaded it is added to the drop down list.

All database changes go through the same testing. Three sets of tests can they add a new item, edit and delete. These are all tested for existing and non existing items e.g. cannot add another course with an already existing code.

Each page has a set of links at the top of the page within the header. These vary depending on which page you are e.g. everyone has a logout link, if you're on the administration pages you have a main administration page link, and link to the previous main pages for that area etc This are all individually tested.

2.5.3 System testing

Once unit and integration testing was completed, system testing follows. Here all parts were tested as a whole with all students' processes being completed and checked in relation to the administration pages and visa-versus e.g. a student completes application, administration uploads learning agreements, student is able to view learning agreement and so on.

2.5.4 Compatibility testing

Compatibility testing concerns evaluating how well-suited the system is in different computing environments. This testing was limited due to the software I was programming in and the restricted available of hardware e.g. Bandwidth handling, networking, computer hardware capacity, ram etc.

However, I could test for browser compatibility. I tested my system in Firefox and Internet Explorer. The only difference I found was in CSS therefore I made allowances for this with margins and padding.

Unfortunately, I did not have access to Safari therefore could not test this web browser.

2.5.5 Acceptance Testing

There are a variety of different stages and types of acceptance testing. Many of these have been used throughout the implementation stages. Smoke testing are tests done directly after a piece of code has been put together or corrected. These are done to give assurance that the changes work and do not fail. I performed this throughout my implementation to make sure my code worked correctly,

This is then followed by the test suite where all functionality is tested to work based on the requirements. This I completed for all my specification requirements.

2.5.6 Regression vs Retesting

These are two very relevant words throughout the process of implementation as all tests involve one or the other or both. Retesting occurs when there is a bug that is fixed and needs to be retested continuously to ensure dependability. Regression is when a piece of code has been altered or added and the impact of this code needs to be checked in relation to other units of the system. This is to evaluate its effect. As I have just discovered there is a Microsoft Test Manager attached to visual studio. If there was more time I would investigate this and understand how to use it.

2.5.7 Future testing

All these testing techniques can be undertaken either being applied to black or white box testing depending on what you wish to test and what level.

Until, my project is fully operational, uploaded and linked to a server there are several tests I am limited in being able to accomplish.

For instance, Compatibility testing is limited as I cannot test the full capabilities and requirements of the system e.g. UNIX, MVS, ram requirements etc Therefore, I can also not perform stress testing or web performance testing, to test how many users my system can take, at what time and what delays might be expected. This is linked with Load testing also, which will test its capabilities between peak and off peak times.

One very important type of testing to be carried out is Penetration testing. This is to ensure the security and reliability of the system by replicating malicious attacks. This needs to be conducted by a professional before the launch of this system. As any security breach is a risk.

3.0 Project Planning/Reflection

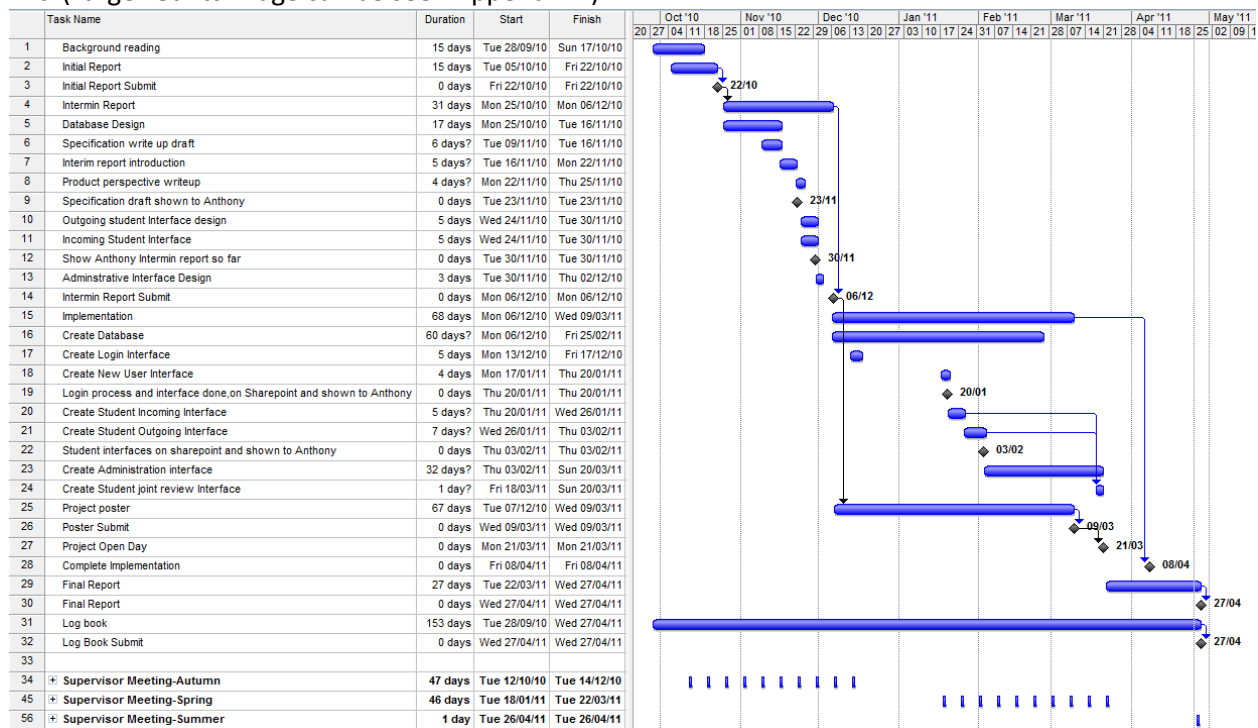
As can be seen, the development process of creating system is a long and complex process. Therefore, it is essential it is planned and managed accordingly; this can be done via a GANNT chart.

3.1 Gantt Chart

My GANTT has been continuously changing since my first revision in my Initial report. The timescales have to be adjusted for different tasks depending on the difficulties of the tasks, the way which tasks link together and other interferences (e.g. other assignments, illnesses, personal issues etc).

Below is my current up-to-date and final Gantt. Changes that have occurred are mainly to my Implementation stage. I placed creating the Administration page above that of the joint Student page, also extending its duration and limiting that of the student interface time.

This was due to the I considered it to be of higher importance as it proved more essential functionality for the system to work and meet the systems requirements. The student joint interface also seemed easier to create then first expected therefore I expected only needing one day to create it. (Larger Gantt image can be seen Appendix A)



I also pushed back the finish date of the implementation, because even though all the necessary functionality for the system to work was at an adequate level, there was still extra functionality that I wished to add along the way to improve the system to make it great. Therefore, I aimed to have these finished at latest a few days before the final report.

I also pushed back the start date of my final report start so I could concentrate on my implementation, also without complete implementation this limited how much could be done for my final report.

4 Conclusions

Overall, I believe my system has been successful meeting many if not all of the initial requirements of the system and has also had extra functionality added to improve the overall quality of the system (as described in section 2.1 Specification). The overall look of the system appears professional and keeps with the design of the Essex current web interfaces. This means it can be easily incorporated into the existing Essex network.

Examining my Gantt chart and my log book, I can see my schedule was revised several times to incorporate other assignments, complexity of tasks and miscellaneous issues. I think this could be expected with a big task such as this system. I believe I still kept the overall timescale of the project achieving a large proportion of my project in good time and quality.

My biggest challenges came in my Implementation stage; this was due to the fact that tasks that were initially planned were more complex than first anticipated. Referring to my Logbook it can see there were three big challenges:

- Controlling the visibility of the Application pages
- Linking gridviews to the buttons and check boxes
- Uploading Pdfs.

These delayed the progress of my project severally as it took a long time to research methods of completing them. Visual Studio and SQL server I believe were a good choice of software as they supported these problems and other functionality. They also allowed for easier transportation of the project as they are linking Microsoft party products.

4.1 Future Advancements

My project has a lot implemented functions and features within the scope of the project specification. Nevertheless there are still many more ideas I would have liked to implement into my system if I had more time. Below lists all my ideas and suggestions and how my system could further be improved.

- ❖ As previously shown and explained all my css is basically the same with minor changes, with a constant header, body and footer structure. Therefore it would be more logically to have a Master CSS page coded that could be used and edited for all pages.
- ❖ The new user page works in the sense of navigation but has no backend functionality, therefore is a dummy page. Similarly to all the other gridview pages, I would create an alert that notifies the Administrators on the main admin page of new users wanting login details. This would be linked to a checkbox gridview displaying all the data that once submitted loaded it to the database and emailed all recipients their details. This email functionality can be achieved through using the namespaces Outlook = Microsoft.Office.Interop.Outlook, System.Collections, and System.Reflections. This means I can access Microsoft Outlook on computers and create a template message that can be dynamically altered with each new students details and sent.
- ❖ This email functionality could also be used to inform students of when their documents have been uploaded.
- ❖ The name space Microsoft Outlook can also be used to access the address book and details to gain all the Essex students usernames and passwords or by accessing the Universities Management Information system. Making this data insertion each year into the database tables simpler.
- ❖ At this present time the Learning agreements are still filled in by hand and then scanned and saved into the system. This is still a lot of extra processing time for the Administration staff and it would be more logically for everything to be kept electronically. Currently for the learning agreement the relevant information is inserted into a gridview so the administration can view and copy it. The advancement would be for this data to be inserted directly into a Learning Agreement plan agreement that can be edited and saved by the Administration. This data can then be saved in the Learning Agreement Outgoing and Incoming tables I have already created but are not in use at this moment. Presently I have to enter each UserID and information in them by hand for other insertions to work. The course table also needs to be altered to support the Incoming students course choices.
- ❖ A further advancement on that would be that the learning agreements could also be signed electronically. However, this would require external electronic equipment
- ❖ The administration page also offers the option for the administration to make database changes. Their a few improvements to be made to these. They are:
 - The Edit options allow you can edit all the details but not the Code. Therefore I would make an initial page where they enter the code of the item they wish to edit. Then all previous details are shown then the changes are saved.
 - Students are currently only deleted from the client table, meaning students can no longer log in. Another feature could be created to delete students completely from the program whether it's just their personal details or their learning agreements also.

- ❖ The student page currently has three buttons at the bottom of each page (bar the first where no back button is needed). A 'Save and Next' button has been created but with no implemented functionality. This button would be used if a student was part way through the application process and waited to pause to gain the necessary information or simply to pause and return to it at a later point. This button should save all the information currently entered and the page currently on, then when the user next logs in return to the correct point in the process.
- ❖ The student application pages at this current point in time do not contain any validation. This means vital fields can be left blank or entered with incorrect data or incorrect type of data. This also presents a security risk, if other programmers decide to maliciously hack the system. This problem would be first priority to fix before use, i.e. so they could not submit DROP DATABASE etc Therefore, validation needs to be added to each field.

5 Acknowledgements

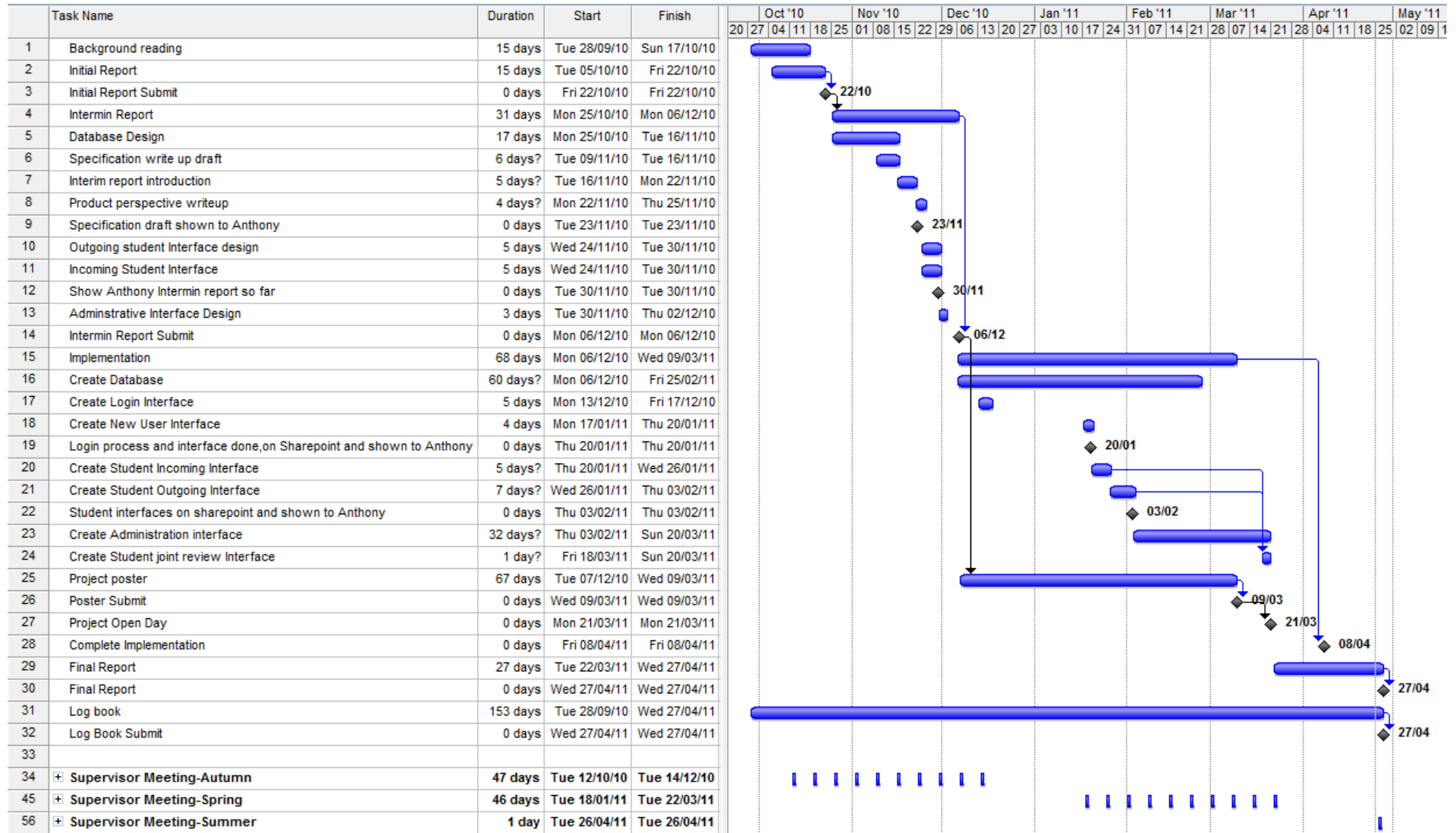
Craig Pilgrim – For his knowledge of appropriate testing.
 Luke D'Wit - Beta Tester

6 References

1. http://www.urv.cat/foreign_students/mobility/en_se_index.html, *Erasmus image*.
2. <http://encyclopedia2.thefreedictionary.com/relational+database>. *Definition of Relational Database*. 1994-2008 [cited.
3. McGraw-Hill Dictionary of Scientific & Technical Terms, E., <http://encyclopedia2.thefreedictionary.com/data+Redundancy>. 2003.
4. <http://www.webopedia.com/TERM/M/md5.html>, *MD5 Definition*. 2011.
5. Chapple, M., <http://databases.about.com/cs/development/g/ODBC.htm>. 2011.
6. Chapple, M., <http://databases.about.com/cs/administration/g/query.htm>. 2011.
7. McGraw-Hill Dictionary of Scientific & Technical Terms, <http://www.answers.com/topic/data-table>.
8. http://www.webopedia.com/index.php/TERM/R/referential_integrity.html, *Definition - Referential Integrity*. 2011.
9. Haswell, J., *Project 95- A Web Based Tool for Providing an Audit Service for the Recognition of a Study, Abroad Period: Interim report*. 2010.
10. databasedev.co.uk, http://www.databasedev.co.uk/primary_foreign_keys.html. 2003 - 2010.
11. <http://www.onestoptesting.com/sdlc-models/v-model.asp>, *V-Model*.
12. [http://msdn.microsoft.com/en-us/library/aa292197\(v=vs.71\).aspx](http://msdn.microsoft.com/en-us/library/aa292197(v=vs.71).aspx), M., *Unit Testing*. 2011.
13. [http://msdn.microsoft.com/en-us/library/aa292128\(v=vs.71\).aspx](http://msdn.microsoft.com/en-us/library/aa292128(v=vs.71).aspx), M., *Integration testing*. 2011.

7 Appendix

Appendix A



Appendix B



Study Abroad Office

[Logout](#)

WELCOME TO ESSEX STUDY ABROAD

Additions and Changes

- [Add new outgoing student](#)
- [Add/Edit University details](#)
- [Add/Edit an Essex department](#)
- [Add/Edit an Essex Course](#)

5 have submitted their Learning Agreements information and have either not yet been signed or uploaded. [View Students](#)

3 that have submitted their Learning Agreements and have not yet uploaded a signed version. [View Students](#)

2 have not started their Study Abroad period but have a signed Learning Agreement. [View Students](#)

2 have started their Study Abroad period but yet not ended it. [View Students](#)

2 have ended their Study Abroad period but yet not had a Diploma transcript uploaded from the host. [View Students](#)

2 have ended their Study Abroad period but yet not had a Completed Diploma transcript uploaded. [View Students](#)

last reviewed: 19 January 2008