|  |  |  |
| --- | --- | --- |
| (The Easy Adjustable Administration System) | | |
| Student nameRichard Galvin | Student numberD11125535 | SupervisorEdina Hatunic Webster |

# Chapter

# Introduction

All organisations no matter what their size need to collect and store information in order to carry out their everyday tasks. Data protection laws have been put in place in order to govern the collection storage and use of data particularly that relating to people. Where many small organisations struggle is acquiring a solution that is tailored towards their needs, is easy to use, will provide services to help manage recourses, adheres to data protection laws and above all is affordable. Teadas is a web application which has been designed to meet all those needs.

The case study for this project is that of a small adult education centre that provides courses in several different categories including ICT, Horticulture, Literacy and Culinary. Through cuts in funding over recent years it has been a struggle for the centre to create and manage an IT structure that will provide adequate data retrieval, storage and usage facilities. Through research and consultation it was found the centres requirements included a simple efficient way to collect and store student and tutor data and the means to use this data to schedule classes, manage resources such as PC and file server space allocation and the ability to track student attendance and progression.

1.1 Aims and Objectives

Teadas’s goal is to provide small businesses and education centres with limited resources and budgets a simple and secure way to collect, store and manipulate user data as well as automating everyday tasks such as timetabling/ staff scheduling, attendance and statistical reporting as well as automating basic network administration tasks such as server account and folder creation as well as group and permissions management. Teadas has been designed and developed with the following aims and objectives in mind.

1. **Simple and secure data collection and storage**
2. Regulated access
3. Attack prevention mechanisms
4. Simple user friendly interfaces
5. Easy to use options for bulk import of data
6. Data integrity checks
7. Clear concise error messages and warnings
8. **Automation of timetabling, scheduling and reporting**
9. Easy to use tools
10. Intelligent option provision
11. Automated alerts
12. Background functionality to manage network resources
13. **Simple data retrieval mechanisms and attractive data presentation**
14. Filter and search options were needed
15. Clear and concise data presentation

Although this application is being designed with an adult/ community education centre in mind it is hoped that is designed in such a way that with minimal configuration and structural changes it could used by any small to medium organization.

* 1. Chapter Walkthrough

This section provides a brief outline of the structure of the preceding chapters in this manual, their contents and their main objectives.

Chapter 2

This chapter presents the findings of research carried out into what requirements exist for data collection, storage and usage. It goes on to describe how the centre being used as a case study for this project currently carries out such tasks and the perceived limitations and shortcomings of this approach. Existing technologies available in the areas which provides similar functionality to what Teadas aims to provide are also discussed in this chapter.

# Chapter

# 2 Requirements Gathering

This chapter presents the findings of research conducted during the requirement gathering phase of this project. It examines what the requirements exist for the acquisition, storage and usage of personal data as prescribed by the Data Protection Agency. It goes onto examine how the centre chosen as the focus of this case study currently meets these requirements and what other difficulties it faces when carrying out various other administrative duties.

# 2.1 Data Protection

The following are the eight golden rules of data protection [5].

1. [Obtain and process information fairly](http://www.dataprotection.ie/docs/A-Guide-for-Data-Contollers/696.htm#1)

2. [Keep it only for one or more specified, explicit and lawful purposes](http://www.dataprotection.ie/docs/A-Guide-for-Data-Contollers/696.htm#2)

3. [Use and disclose it only in ways compatible with these purposes](http://www.dataprotection.ie/docs/A-Guide-for-Data-Contollers/696.htm#3)

4. [Keep it safe and secure](http://www.dataprotection.ie/docs/A-Guide-for-Data-Contollers/696.htm#4)

5. [Keep it accurate, complete and up-to-date](http://www.dataprotection.ie/docs/A-Guide-for-Data-Contollers/696.htm#5)

6. [Ensure that it is adequate, relevant and not excessive](http://www.dataprotection.ie/docs/A-Guide-for-Data-Contollers/696.htm#6)

7. [Retain it for no longer than is necessary for the purpose or purposes](http://www.dataprotection.ie/docs/A-Guide-for-Data-Contollers/696.htm#7)

8. [Give a copy of his/her personal data to an individual, on request](http://www.dataprotection.ie/docs/A-Guide-for-Data-Contollers/696.htm#8)

## 2.1.1 Challenges for Small Organisations in Relation to Data

Small education centres and businesses to store and manipulate data belonging to their users for various different reasons. These users may be staff or external users such as students or customers. The challenge these organisations face is to being able to store this data in a secure manner while maintaining a high level of availability. Although they may have some IT infrastructures in place many of these organisations lack the knowledge and skills in-house to utilise these structures properly and achieve their IT goals.

Education centres need to store sensitive data regarding staff and students including PPS numbers. Any application developed will need to adhere to these rules in order to ensure the centre remains compliant. The centre chosen as a case study for this project has provided a useful insight to the struggles they face to in managing data and network resources. Due to financial constraints it has been left to tutoring/ administration staff and volunteers to carry out these duties. Considering the lack of time, resources and specialist skills available to the centre a gallant effort has been made in this respect. Having said that the collective feeling is that some new technology could help provide compliance with data protection rule.[6].

2.1.2 Challenges in Relation to Resource Management

Having conducted research into and witnessed at first hand the effects that cutbacks in funding over the past few years such centres it can be concluded many have been forced to severely trim back their staffing levels to a point were some staff are carrying out duties which had been carried out by multiple members of staff in the past. Many administrative duties such as timetabling, scheduling, reporting and student/ tutor participation are carried out manually as it is felt there is a lack of affordable integrated software on the market [1], [2]. Indeed this view is shared by other small to medium business owners. [3], [4].

## 2.2 Similar Technologies

As TEAd@S is intended to be an all in one administration system that will incorporate functionality which might usually be provided in separate applications no one application reviewed provides all the functionality proposed in TEAd@S and vice versa. Instead applications that provide much of the functionality proposed and those that provide specific functionality have been chosen. There are many different software tools on the market targeting the different academic markets. Through research it has been found such tools are quite limited in this country particularly those intended for the adult and community education market. Most applications on the market provide functionality aimed more at delivering course material rather than providing the resources to set up and manage courses.

2.2.1 Nielson Heurisitics

Where possible trial versions of the different applications have been tested using Nielson’s Heuristics as a guide. Jakob Nielson proposes 10 general principles concerning user interaction design which are outlined below [7].

1. Visibility of system status

User should know where they are in the system and should be kept informed of what is going on.

1. Match between system and real world

The system should match how the user communicates and approaches tasks as much as possible

1. User control and freedom

Should provide easy to find undo and exit options for the user

1. Consistency and standards

Language and wording should be consistent throughout the system and comparable with is already available in the same category on peer systems

1. Error prevention

Should prevent errors as much as possible providing users with confirmation messages before completing actions

1. Recognition rather than recall

Options, objects and actions should be visible to the user. Help should be visible or easy to find

1. Flexibility and efficiency of use

System should be useable from a novice and an expert’s point of view. Expert users should be provided with options to accelerate use

1. Aesthetic and minimalist design

Only provide information that is needed. Keep as uncluttered as possible.

1. Help users recognise, diagnose and recover from error

Provide clear concise error messages and solution suggestions

1. Help and documentation

Should be easy to find, short and concise

|  |  |  |  |
| --- | --- | --- | --- |
| Heuristic | Moodle | SalesPulse | EFront |
| Visibility of system status | 10 | 8 | 7 |
| Match between system and real world | 7 | 8 | 8 |
| User control and freedom | 8 | 8 | 5.5 |
| Consistency and standards | 8 | 8 | 6 |
| Error prevention | 8 | 7 | 5 |
| Recognition rather than recall | 8 | 8 | 6.5 |
| Flexibility and efficiency of use | 8 | 8 | 6 |
| Aesthetic and minimalist design | 8 | 7.5 | 8.5 |
| Help users recognise, diagnose and recover from error | 8 | 7 | 5 |
| Help and documentation | 7 | 9 | 0 |

### 2.2.1 Education Solutions

MIT’s suite of provides much of the functionality proposed in TEAd@S albeit in a suite rather than one application. Unfortunately a trial couldn’t be secured for this product so no review was possible [7].

## 2.2.2 Sales Pulse

Adult education centres are required by funding bodies to use the SalesPulse Adult Education application. This is a web application which is cloud based with a client side written in HTML, ASP and JavaScript. It is mainly used by the various bodies to collect information in order to sanction funding and tutor payments. The application allows the creation of course categories and courses, the insertion of tutor and student data and the allocation of students and tutors to specific courses. Timetables and reports can also be generated.

Once logged into SalesPulse the user interface is aligned to the left hand-side and appears cluttered and cramped with a small font. Once adjusted to the appearance the user interface it is quiet intuitive and easy to use. SalesPulse Adult Education offers resources to users depending on which projects they are involved with. The main screen offers a choice of all projects across the top of the screen with access to those the user is offering courses in. On the main screen options for entering and retrieving data are provided depending on what project has been selected. IFrames provide forms for adding data and for presenting data retrieved from the database. A menu bar provides tabs for each sub category which when selected allows the user to tailor search queries and enter data depending on what the category is.

The format of each different section is consistent throughout the application. Easy to uses search options are provided as well as instructions on how they can be used. Although many search options are provided one particularly useful one is the provision of the letters of the alphabet.

In general the help provided by this application is sufficient. Instructions on the various screens are quite limited although user guides are available for most sections and tasks on the main screen. The application is a little inconsistent in informing the user exactly where they are in the system. In some categories the user is kept informed at the top of the screen to exactly what section and subsection they are in. Other sections either don’t provide this at all or only in part. After a period of use it became clear the most efficient way to get back to the main menu which can involve up to three steps depending what section the user is in.

Although SalesPulse contains functionality that is the most comparable to that proposed in TEAd@S and is designed for adult education centres it does present some shortcomings for centres that provide courses other than those that fall under the adult education umbrella. Data collected and stored by Sales pulse is relevant only to courses affiliated to funding bodies [9]. Timetables generated in this application are produced for tracking of course hours rather than as use as a reference for tutors and students. Reports generated in Sales Pulse are rendered in excel format only and do not represent all data required for end of year reports. Sales Pulse also does not allow for data integration and bulk data importation although these options have been explored in the past [10].

## 2.2.3 Moodle

Whilst formulating a project idea providing functionality to facilitate the deliverance of course material in TEAd@S was considered. This led to research of applications that currently provide this type of functionality. One of these is Moodle and even tough this type of functionality will not be included it was decided to include the review in this report as the application does provide some of the functionality proposed for TEAd@S. Some of this functionality includes:

* adding users
* groups
* courses
* adding tutors and students to courses
* adding grades
* reporting
* user administration
* backups

Moodle uses HTML, PHP and JavaScript to present content to users. Its main page is uncluttered with large font and provides all options available to the user on the left hand-side and any relevant content is displayed in the centre of the screen. Along the top of the screen is a status bar which keeps the user informed exactly what part of the system they are in at any given time. This type of layout is used throughout the application giving the user the impression that they it is the page content that is changing and not the actual page. This provides for easy navigation and clear system status but it does result in cluttered displays at in particular when generating reports.

Users can be added to Moodle either individually through a form or by bulk upload using csv delimited files. Categories and course can be added and updated through a course and category screen. This screen offers options to sort and move categories as well as adding courses to categories. In truth this screen is slightly confusing and requires some trial and error before figuring out what each option is for. A nice feature in this section is the ability to restore courses and categories from backups. Backups can be run manually or be configured to run automatically. The user can choose to back-up the whole of the application specific parts.

As a demo version of Moodle was used to review Moodle some of the criticism aimed at it may be unfair but there did seem to be a lack of easy accessible help although plenty of documentation is available from the Moodle website. All icons used throught Moodle also provides information regarding their use if hovered over. Error prevention is lacking slightly in places particularly when adding users. As stated previously TEAd@S is not intended to be a Moodle type application but ideas on features and layouts have been provided by the application as well as how the application should perform overall. [11]

## 2.2.4 E-Front

This application provides similar functionality to that available in Moodle and so only a short review has been carried out on it. This application provides a friendly user interface which incorporates images with text to present user options. Forms for adding content are also visually appealing and with a simple layout.

|  |
| --- |
|  |

Fig 2. Print screen of EFront demo home screen. (http://demoefrontlearning.com)

|  |
| --- |
|  |

Fig 3. Print screen of EFront Demo user information form (www.demoeforntlearning.com)

During testing this application scored badly in some of the heuristics being used as a testing guide. Error prevention and detection is inconsistent throughout the application. Once a task has been completed either successfully or not there is no clear exit/ return path or progression path. An interesting feature throughout the application is the use of type ahead lists instead of drop dropdown lists for certain fields. Depending on the amount of options available this can be significantly more efficient. There is little or no help available in this application but as this is a demo version it may not be available. [12]

2.2.5 Server Management Technologies

As it is proposed that TEAd@S will provide automated network administration programs that already provide this functionality were investigated. Below is a table of applications available for Windows. No programs for Linux were found during this research through research of network administrator forums it seems scripts are the main method of carrying out such tasks [13], [14].

|  |  |  |
| --- | --- | --- |
|  | Adaxes from Softerra | [ADManager](http://www.manageengine.com/products/ad-manager/index.html) |
| Platform | Windows | Windows |
| User management | Yes | Yes |
| Group Management | N/A | Yes |
| Password management | Yes[1] | Yes |
| Bulk User/ group/ password management |  | Yes |
| Import /export files | No | Yes |
| Automation of user admin tasks | Yes | Yes |
| Reporting | Yes | Yes |
| Backup management | No | No |
| Server cleanup/maintenance | No | Yes |
| Cost | Per AD user license | Free for 100 systems |

Table 2. table of applications an associated features

Note

1.separate application

## 2.3 Further Research

As this application was being developed with a particular case study in mind it was imperative that the perspective users be involved at every step from design, development and testing. Different forms of research where carried out involving the centre’s staff from the various different roles the application is aimed at. much of this research was invaluable when specify the applications requirements apart from some issues which arose and are outlined below.

## 2.3.1 Brainstorming Session

An initial brainstorming session was held with the centre coordinator and two members of staff who carry out both administrative and teaching duties. Some ideas were discussed as to what the application may include. The main points that came from the meeting are as follows:

* The centres current database was developed using legacy software which is now outdated
* Problems with data storage such as duplication, corruption and loss have occurred increasingly over the past few months
* It is feared requirements for data protection are not being met
* Data access is limited is limited as is how it is presented
* Many tasks are being carried out manually taking up staff hours which could be better spent on other duties
* There is no structured way to record and monitor student and tutor participation
* The same data is required to be entered for different organisations
* There is no way of tracking student progression
* There is a lack of knowledge regarding how to create and manage user accounts and how to create and manage folders and their permissions
* The centres IT resources are either being underutilised or not used correctly including:
* Duplication of files on the file server
* Files saved in incorrect folders
* Files being saved on PCs rather than the file server

## 2.3.2 Interview

The brainstorming session had provided an outline of what the user requirements for TEAd@S might be. A set of questions was sent to the centres director with the view of holding an interview a short time later. Through investigation and research a clear picture of what IT resources were available at the centre and how they were being utilised had been developed, clarification was now needed regarding data in particular the following:

* What data regarding what parties is being stored locally
* What data is required by third party organisations
* How was data was currently inputted and retrieved
* What reporting was required and how it was currently been carried out

Clarification was also sought on what type of learners attended the centre with a particular focus on their IT literacy skills.

## 2.3.3 Surveys

The interview helped clarify issues surrounding data and eliminate some provisional requirement particularly those that were being considered for learners. Three separate surveys were released during the requirement gathering phase, the purpose of which was to further clarify what the user requirements should be. One survey was created for tutors, another for administration staff and a third designed for other centres that provide similar education services.

|  |
| --- |
| Fig 4. Screen shot of survey results for proposed requirements from tutor survey (survey monkey) |

Results from the tutors survey help clarify what features would be useful to them for course delivery. From the table above it is evident that there is support for many features laid out in a provisional list of functionality which was presented for consideration. Other suggestions were also provided through the survey such as the provision of individual learning plans for students and the ability for tutors to upload links to QA surveys.

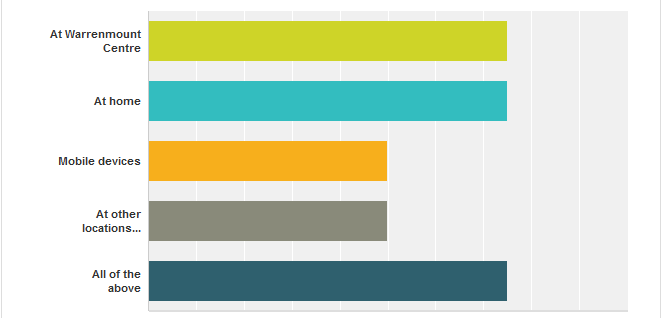
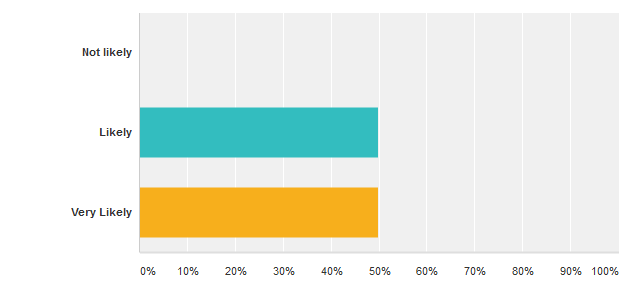


Fig 5. Screen shot of survey results for likelihood of use and location of use from tutor survey (survey monkey)

The survey also provided an insight into where tutors would like to be able to use the application. This will aid the developed in deciding which frameworks should be used and what platforms it should be aimed at. Other questions were designed to ascertain how tutors currently carry out tasks such as roll call and the recording and tracking of student performance.

The survey also provided an indication of the level of support for such an application and the willingness to utilise it if developed.

The second survey was designed to ascertain what features administration staff would find useful in an application. In hindsight due to the fact staff carries out different administrative tasks the survey could have been designed better to provide more informative results than what it did. The survey did provide useful feedback in relation to perceived strengths, weakness and difficulties associated with applications currently in use at the centre including SalesPulse which was reviewed previously.

|  |
| --- |
| Fig 6, Screen shot from survey results from survey designed for admin staff (survey monkey) |

The table above right represents the respondent’s opinion of the usefulness of timetabling functionality which will in turn provide the means for student and tutor participation tracking and reporting. The survey also confirmed that it is felt a student and tutors profiles would be made available which would contain personal information as well as any course participation records. It was suggested some sorted of unique identifier be assigned to users in order to eliminate duplicate eateries and the re-entry of data from archives. The table above left represents the opinion of the respondents when asked if the ability to upload data in bulk using .csv files would be a useful piece of functionality.

The third survey was designed to get a general idea of what other centres were using to carry out their own data storage and handling tasks and to try a general opinion if they felt the functionality proposed in TEAd@S would be useful. Although limited responses were received they served to confirm that most of applications used by other centres are bespoke owing to the fact that there are very few applications on the market which are affordable and provide the functionality required by such centres. Respondents also indicated that they felt a lot of functionality proposed would be useful or are already included in the applications they use.

As mentioned previously the survey didn’t provide all the necessary information needed to aid a final decision on requirements so a meeting was arranged a selection of administration staff and management. During this meeting the results of the different surveys were discussed and the level of support the proposed requirements received. The meeting also served to clear up what data was required by the centre itself and third parties such as funding bodies.

2.4 Conclusion

This chapter started by outlining what is expected of organisations when it comes to handling personal data. It goes on to outline conclusions regarding difficulties faced by small organisations when managing data, resources (both IT and Non-IT) and carrying required administrative duties. Section 2.2 outlines available technologies which are currently available to tackle some of these problems providing a review of each using Nielsons Heuristics. Section 3 provides an analysis of the results acquired from research carried out within the centre being used as a case study as well as those similar in size or services provided. The next chapter outlines the functional and non functional requirements which have been specified as a result of the research carried out in this chapter

# Chapter

# 3 Requirement Analysis

Carrying out research allowed the developer gain a clearer picture of the applications requirements and discount several requirements which had been proposed. From completed surveys student note provision and work collection gained favourable support. Through the meeting discussed previously and through discussions with a project supervisor it was decided that both these pieces of functionality would be left for future work. It was conveyed in the meeting that they wouldn’t be seen as a priority for the centre and that they would probably be underutilised due to the student’s low IT literacy levels. It was also suggested that they would probably form part of a separate application. Another proposed requirement was a reporting engine which would auto-generate the various reports required by the centre. Although it is felt that this would be a useful feature it was decided also to add this to future work or if time permits a final phase. It is felt it is more important to provide the correct data for the reports than concentrating on report design. Some suggestions for features were conveyed through the surveys one of which was an individual learning plan for students. Again it is felt although this would be useful it is not a priority and so has been added to future work and would more than likely be implemented in a separate application. The possibility of allowing students to request access to their student profiles for third parties such as potential employers or school registrars. Although this would probably be a good feature it has also been marked for future work.

## 3.1 Functional Requirements

This project will provide an easy to use all in one application to input, record and utilize data in small education centres and businesses. This application will be designed to be flexible, adjustable and extendable so it can be used for as wide a variety of organisations as possible. Outlined below is some of the functionality that will be developed for adult and community education centres.

The following is what the developed application aims to provide:

* Secure login  
  Entering of user data
* Update, delete or archive student or staff data
* Add, update or delete courses and categories
* Student and tutor profiles presented in a user friendly fashion
* Create timetable
* Generate class schedule for class attendance and performance recording
* Statistic generation
* Perform basic network administration functions

Below is a table of functional requirements decided upon as a result of requirement gathering and their analysis.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Req ID | Name of Req | Description | Priority | | User Contact | | |
| 1 | **Secure Login** | **Login with provided user credentials. Log user visit details** | **High** | | **All Users** | |
| 2 | **Manual insertion of user data** | **Enter data relating to students or tutors using an electronic form. If required accounts and or folders will be created for the user on the files server.** | **High** | | **Administration staff** | |
| 3 | **Upload data from files** | **Bulk upload user data from .csv file. If required accounts and or folders will be created for the user on the files server.** | **Medium** | | **Administration staff** | |
| 4 | **Restore user data from archive** | **Student and tutor data is archived at the end of each semester. This feature will allow the restoration of only required data. If required accounts and or folders will be created/ reactivated for the user on the file server.** | **High** | | **Administration staff** | |
| 6 | **Add categories** | **Add new categories of courses for example ICT, literacy** | **High** | | **Administration staff** | |
| 7 | **Add new course** | **A new course must be added to existing category. One course can belong to different categories** | **High** | | **Administration staff** | |
| 8 | **Update, delete courses and categories** | **Delete or Update course or category data and resources on file server if required** | **Medium** | | **Administration staff** | |
| 9 | **Add tutor to a course** | **Create a new instance of a course with a unique course code, tutor and running date range. Create folder on file server** | **High** | | **Administration staff** | |
| 10 | **Add a student to an instance of a course** | **Add a student to an instance of a tutors course. If necessary create a folder for the student on the file server** | **High** | | **Administration and teaching staff** | |
| 11 | **View user profile and Update, delete or archive user data** | **To present user data including personal and course details in a visually appealing manner allowing fields to be altered or user deleted** or archived | **High** | | **All users** | |
| 12 | **Create timetable** | **Allow the user to define a specific term period including holidays and**  **to Select relevant data in order to create a timetable** | **High** | | **Administration staff** | |
| 13 | **Select class list from schedule and fill in data** | **To allow teaching staff select class lists from their personal daily schedules and enter student attendance and performance data** | **Medium** | | **System/ teaching staff** | |
| 14 | **Generate statistics for reports** | **Allow users select various criteria and generate statistics relating to them** | **Medium** | | **Administration staff** | |
| 15 | **Manage Groups** | **To allow administration staff to manage groups and their membership** | | **Low** | | **Administration staff** | |
| 16 | **Perform backups** | **Allow users perform backups of the database or file server** | | **Low** | | **Administration staff** | |

## 3.2 Non Functional Requirements

## 3.2.1 Security and integrity

As sensitive data will be inputted and stored in the database it is imperative that security criteria are evaluated at every stage of development. Attacks such as SQL Injection, XSS and infernal due to elevated privileges must be secured against as well as unauthorised access to the system. All passwords and other critical will be encrypted to an acceptable standard as will any communications containing such data. Measure will also be included to ensure data is not replicated in the databases and that data cannot be accessed or affected by operations that are not authorized to do so.

## 3.2.2 Usability

Ease of use is one of the most important requirements of this application. As the main purpose of the application is to attempt to encourage the correct use of IT resources plus recording and handling user data safely users must be comfortable using the system and be able to carry out tasks without assistance after minimal training. Nielsen’s guide on heuristics will be followed closely in order to achieve this goal.

## 3.2.3 Availability and recovery

If users are to be encouraged to use this system to its full potential it will need to be available constantly. Also if an error or malfunction occurs the data must be recoverable in a very short space of time. In order to achieve this a fail over server will be utilized as will an automated backup plan. It is foreseen with time to spare users will be able to perform backups of data before carrying out operations to enable them to rollback to before they started in the event of an irreversible error. This will be available due to the automatic backups but the data may be older then desired.

# Chapter

# 4 Relevant Technologies

This chapter presents the findings of research into technologies which were considered suitable for this project. The technologies are split into different sub sections depending on which tier it is anticipated they could be used in. An explanation on the pros and cons from of the developer’s point of view on using each technology is provided in each section as well. The chapter concludes by providing an outline of which technologies were chosen and for what reason.

### 4.1 Languages and Platforms

It has been specified that Teadas is to be developed as a web application. There exists many different languages and platforms which can be used to develop such an application each providing their own pros and cons. A description has been provided in this section of three such technologies which are deemed suitable for providing the functionality outlined in chapter 3.

### 4.1.1 PHP

PHP is a widely used language for the creation of dynamic web content. It can also be used for command-line scripting and client-side GUI applications. PHP can be used on all major operating systems such as Ubuntu, Debian, Mac OS and windows as well as the leading web servers such as Apache and IIS. PHP supports all of the major relational databases such as MySQL, Oracle, MS-SQL, PostgreSQL and NOSQL databases such as MongoDB. PHP provides a library of code which can be used for common tasks such as error handling and also provides support for many different output types such as PDF, Photograph files and Flash movies [15]. PHP has been noted as proving many advantages for web development including the fact that it is open source, less class structure than other languages allowing for easier interpretation of code, portability it that it works on all major operating systems, web servers and with most major database systems. Overall it is deemed as quiet an easy language to learn and use resulting in an increased speed of development [16], [17]. Some disadvantages of PHP have been described as an apparent lack of security due to its open source nature and also its lack of standardisation [18].

### 4.1.2 Java

Java is an object orientated language for developing client server applications. Java is known to be an extremely portable language. Source programs are compiled into bytecode which can be run on servers or clients using a Java Virtual Machine (JVM) providing Java is installed. As with PHP Java can also be used with all of the major databases types and its help resources are vast. Java is also known as a robust language which is quiet secure. Objects cannot contain references to external data and so addresses of data storage in other systems or operating systems cannot be passed in as an instruction. The JVM performs data integrity checks on each object in order to enforce this [19]. A major attraction of Java for developers is its automatic garbage collection which relieves the developer of the task of allocation and deallocation of memory. Although it is based on C++ as a whole Java is considered an easier language to learn. Some drawbacks of using Java are considered to be its lack of performance at times and lack of support for low level programming. [20], [21]

Many different flavours of Java are available for web application including Java Servlet API, JavaServer Pages Technology, JavaServer Faces Technology and JDBC API to name a few [22].

### 4.1.3 Active Server Pages (ASP)

Asp was developed by Microsoft as a server side scripting language to facilitate the development of dynamic interactive web pages. VBScript is the default scripting language but others can be used such as Jscript. ASP runs on IIS servers which can be run on any Windows machine from NT4 up [23]. Visual studio is also available from Microsoft which provides developers a platform to develop web applications integrating scripting languages with coding languages such as C#, C++, Java, JavaScript and VB. Some advantages of using ASP for web development are considered to be its ease of use, language options, help and documentation availability, development tools and extensibility via COM components. [24]. A substantial drawback of using ASP is the fact if an application is to be hosted locally it must be run on an IIS server which requires a Windows machine to run which will typically add expense to a project.

## 4.2 Databases

This section provides a description of three database technologies deemed suitable for the Teadas application. That is not to say they were the only three that were considered appropriate as there are many different systems which provide the functionality required by Teadas.

## 4.2.1 Oracle

Oracle DB is a relational databases management system. Oracle structure is split logically and physically making it extremely scalable and fault tolerant. Logical elements include table spaces which each create one or more physical datafiles, schema objects which include tables, views, store procedures and indexes to name a few, datablocks, extents and segments [25]. Physical structures comprise of datafiles, redo logfiles and control files [26]. Oracle comes in several different versions including enterprise, standard, express and lite. PL/SQL is an extension to SQL offered by Oracle in order to write triggers, procedures, functions, types and triggers [25]. Oracle is available in different version depending on user needs including Express which is a scaled down version with restrictions on CPU, storage and memory usage that is free for non commercial use [27]. Oracle is compatible with most major operating systems [25].

## 4.2.2 Microsoft SQL Server

According to db-engines.com’s scoring system Microsoft SQL Server currently lies third behind MySQL and oracle in the database ranking table. [28] Like oracle MS SQL Server is a relational database which is accusable by using the SQL language either locally or via applications [29]. T-SQL is similar to Oracles PL/SQL in that it extends SQL to allow developers include procedural programming [30].[tsq l.info] MS SQL Server comes in many different flavours depending on an applications needs. Express is a down version which is free for non commercial use. It contains many features of a full version with certain limitations in place such as processor usage and database sizes. One feature in particular it does not contain is SQL Server agent which provides features such as job scheduling. A Microsoft operating system is required to run all versions of MS SQL Server which makes it a lot less attractive for applications which are required to be portable.

## 4.2.3 MySQL

MySQL is an open source relational database system which is now ranked second behind Oracle in terms of use. MySQL is easily deployable and is compatible with all major operating systems. As with Oracle and MS Server SQL is the language of choice when interacting with MYSQL databases and it supports support procedures, triggers, functions and views. Some advantages of using MySQL have been identified as cost, portability, security, scalability and performance [31]. Some disadvantages which have been identified are the lack of good development and debugging tools, lack of transaction support, lack of check constraints, lack of efficiency when working with large databases and the lack of development on the open source version since taken over by Oracle [31],[32].

## 4.3 User Interface

There are many technologies available for developing interfaces for human interaction. Below is an outline of some standalone technologies and libraries which are available to simplify their use.

## 4.3.1 Javascript

JavaScript is a programming language which makes websites interactive. It can be used to carry out a range of tasks such as but not limited to changing HTML content, attributes, styles and the likes of data validation [33], [34].

## 4.3.2 jQuery

jQuery is a lightweight JavaScript library which wraps JavaScript into methods which can be called by one line of code. It makes JavaScript manipulation much easier as well as Ajax calls. Ajax (Asynchronous JavaScript and XML) is used to make calls behind the scenes to a server in order to update pages without the need for reloading. jQuery is very popular and widely used and there is a vast range of online help available [35].

## 4.3.3 Dojo

Dojo is a toolkit which is similar to jQuery. It was designed to aid the rapid design of cross platform, JavaScript, Ajax based websites. dojo is split into several packages including dojoCore which is the most widely used, dijit, dojox, and util. again dojo is extremely popular and there are plenty of online resources [36].

## 4.4 Chosen Technologies

This section provides a an analysis of the different technologies that have been chosen of the Teadas application. it outlines why each technology has been chosen including why they were deemed more suitable then the alternatives described in the previous sections

## 4.4.1 Language

As a result from the research carried out it has been decided the Java Programming language will be used to develop the business logic layer of this project. Although there was very little to choose from between the different languages it is felt that Java will provide more flexibility and security for this project. Eclipse can be used

Bash scripting will be used to carry out the various file server administration tasks that will be required. Research has shown that such scripts can be run quiet easily from Java methods.

## 4.4.2 Database

MySQL will be used for this project mainly due to its cost, scalability and cross platform support. Although free versions of Oracle and MS SQL Server provide many of the features available in full versions they do come with certain limitations in terms of performance and scalability. MS SQL Server was the first to be discounted as it can only be used with Microsoft products resulting in a lack of portability and added expense. Although Oracle ticked most boxes its lack of scalability would be a worry in the event of a major expansion or deployment of the application.

## 4.4.3 User Interface

Along with HTML and CSS JavaScript and jQuey will be used to develop the user interface. Although there is not a lot to choose between jQuery and other toolkits such as Dojo it is felt there is more support available for the latter..

## 4.4.4 Application Server

Apache Tomcat will be used as an application server mainly due to it cost and ease of use.

# 4.5 Approach and Methodology

## 4.5.1 AUP

Agile Unified Process (AUP) is a simplified version of the Rational Unified Process (RUP) and is seen as lying somewhere between it and extreme programming. It comprises of many of the features of RUP as well as adhering to the principles of Agile development including Test Driven Development, Agile Model Driven Development, Agile Change Management and Database Refactoring. AUP offers flexibility in what tools should be used during design development and testing.

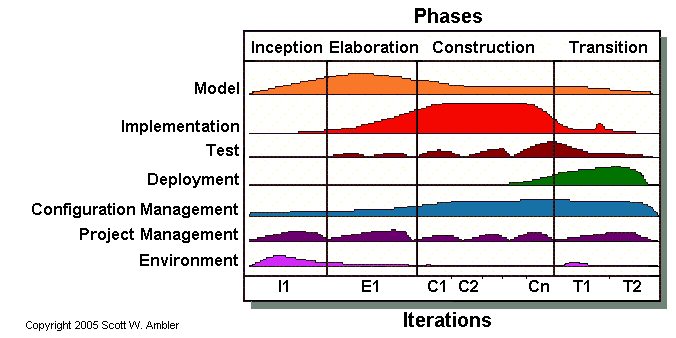


Fig 7 Graphical Representation of AUP (http://www.ambysoft.com/unifiedprocess/agileUP.html)

There are four serial phases of AUP

1. **Inception**

Define initial scope of project and investigate its feasibility and acquire approval. Potential architectures are also investigated

1. **Elaboration**

Further define the architecture and prove its achievable

1. **Construction**

Development of application in stages releasing at regular intervals

1. **Transition**

Integrate application into real world environment

AUP contains seven disciplines which are performed iteratively

1. **Model**

Important part of the process it covers RUP’s Business Modelling, Requirements gathering and analysis and design disciplines.

1. **Implementation**

Transform models into executable code and test

1. **Test**

Perform testing to ensure applications meets users requirements, non functional requirements and contains no defects

1. **Deployment**

Make system or part of available to the user

1. **Configuration management**

Manage access and changes to the different parts of the project

1. **Project management**

Direct different activities of project including specifying tasks and monitoring progress

1. **Environment**

Ensure a proper environment is set up with tools and platforms needed for development and testing

When following AUP software should be released in portions instead of all in one go. Having developed and tested a particular piece of functionality it should be releases in a staging area for QA and User acceptance testing before being integrated with other components for phased deployment into a real world environment [37].

**\*\*Please note\*\* Full research the area has been carried in this area and is to be included in the manual**

# 4.5.2 Summary

AUP will be the Software Development Methodology used as a guide for this project. Although it has been overtaken by other methodologies in recent years in terms of usage it is felt as this project is being undertaken by a single developer it is a good methodology to follow due to the fact it comprises of the best parts of RUP without its complexities and also adheres to Agile design principles. Other methodologies which were researched and discounted for various reasons include Waterfall which was discounted due to its inflexible nature, Spiral due to its complex nature and the fact its more suited to larger projects, Scrum due to the fact it is designed more for team projects, XP again mostly because it is designed for team projects and in particular that it specifies that programming should be carried out in pairs and RUP mainly due to its complexity.

This project has been broken up into phases which are specified in the user requirement template. Each phase contains sub phases which have been assigned in order of importance. As each sub phase is developed it will be release into a staging area for quality assurance (QA) and user acceptance testing (UAT). Once the entire sub phases of a particular phase have been tested sufficiently they will be integrated into an overall testing area for further QA and UAT test as well as integration and performance testing.

Development will be carried out using a bottom up approach. During development each initial use case will be taken and analysed to ascertain if it needs to be altered or expanded and to determine what classes and functions will be needed to achieve what is required. Starting at the data persistent layer the initial ERD and existing database will be analysed to determine if tables, procedures, triggers, views and functions exist which are sufficient for the task required. If not existing elements will be altered or new ones added as required. If any procedural programming and security measures such as views have to be implemented these will be tested before continuing. Next the business logic layer will be analysed to determine if any classes and functions exist to facilitate communication with the data persistent layer and also external systems such as the file server and the user interface. Again existing elements will be updated or new ones added as required. Any new functionality implemented will be tested locally to ensure it performs as required. Finally how the user interface will appear and interact with the business logic layer will be analysed. Tools such as screen shots of mock-ups and diagrams will be used to provide a clear picture of the desired workflow. Once coding is complete the new functionality will be tested as a complete module. If previously written functions or resources have been used to implement the desired functionality the requirement they were developed for originally will be tested to ensure they still perform correctly.

4.6 Conclusion

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Phase | Req ID | Name of Req | Description | Priority | | User Contact | | |
| 1 (a) | **1** | **Secure Login** | **Login with provided user credentials. Log user visit details** | **High** | | **All Users** | |
| 1 (a) | **2** | **Manual insertion of user data** | **Enter data relating to students or tutors using an electronic form. If required accounts and or folders will be created for the user on the files server.** | **High** | | **Administration staff** | |
| 1 (b) | **3** | **Upload data from files** | **Bulk upload user data from .csv file. If required accounts and or folders will be created for the user on the files server.** | **Medium** | | **Administration staff** | |
| 1 (b) | **4** | **Restore user data from archive** | **Student and tutor data is archived at the end of each semester. This feature will allow the restoration of only required data. If required accounts and or folders will be created/ reactivated for the user on the file server.** | **High** | | **Administration staff** | |
| 2 (a) | **6** | **Add categories** | **Add new categories of courses for example ICT, literacy** | **High** | | **Administration staff** | |
| 2 (a) | **7** | **Add new course** | **A new course must be added to existing category. One course can belong to different categories** | **High** | | **Administration staff** | |
| 2 (a) | **8** | **Update, delete courses and categories** | **Delete or Update course or category data and resources on file server if required** | **Medium** | | **Administration staff** | |
| 2 (b) | **9** | **Add tutor to a course** | **Create a new instance of a course with a unique course code, tutor and running date range. Create folder on file server** | **High** | | **Administration staff** | |
| 2 (b) | **10** | **Add a student to an instance of a course** | **Add a student to an instance of a tutors course. If necessary create a folder for the student on the file server** | **High** | | **Administration and teaching staff** | |
| 2 (c) | **11** | **View user profile and Update, delete or archive user data** | **To present user data including personal and course details in a visually appealing manner allowing fields to be altered or user deleted** or archived | **High** | | **All users** | |
| 3 (a) | **12** | **Create timetable** | **Allow the user to define a specific term period including holidays and**  **to Select relevant data in order to create a timetable** | **High** | | **Administration staff** | |
| 3 (b) | **13** | **Select class list from schedule and fill in data** | **To allow teaching staff select class lists from their personal daily schedules and enter student attendance and performance data** | **Medium** | | **System/ teaching staff** | |
| 4 (a) | **14** | **Generate statistics for reports** | **Allow users select various criteria and generate statistics relating to them** | **Medium** | | **Administration staff** | |
| 4 (b) | **15** | **Manage Groups** | **To allow administration staff to manage groups and their membership** | | **Low** | | **Administration staff** | |
| 4 (b) | **16** | **Perform backups** | **Allow users perform backups of the database or file server** | | **Low** | | **Administration staff** | |

## Table 3 User requirements

|  |
| --- |
|  |

|  |
| --- |
| **Fig 8 Flowchart representing overall development flow** |

|  |
| --- |
| fig 9 Flowchart of individual requirement development |

# Design

## Technical architecture diagram:



Fig 10 architectural overview



Fig 11 architectural overview with technologies

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  | | --- | | User interface |   *Application server*   |  | | --- | | *Business logic layer*  Call script for adding a new user providing any  Response handler  Credentials or ack  Request handler  Add user to database |  |  | | --- | | *Data persistent layer*  *Return credential flags for file server or an ack that data has been inserted* |  |  | | --- | | *Create a/c + folders return ack* |   *Linux file server*  *Fig 12. Example of a proposed workflow* |

## Other design documents

## 

|  |
| --- |
|  |

Fig 14 Use Case Diagram

# Initial Use Cases

Below are a set of initial uses cases which have been created for each of the requirements identified through gathering and analysis. These use cases are not finalised and have been created merely as an initial guide of what will be required. During development each use case will be taken to ascertain if it needs to be altered or expanded and to obtain a clear picture of what exactly will be required to implement the requirement.

|  |
| --- |
| Use case number: 1 |
| Use case name: Login User |
| Intent: To allow users to login with unique username and password |
| Precondition: User opens login screen |
| Use case initiation: Username and password entered |
| Dialog (Description) :   1. User: Hits submit button 2. System: Hashes password and checks against the database 3. System: Checks what group user belongs to 4. System: Logins user and presents appropriate screen   Alternate Flows:  3a. System rejects login request and provides error message |
| Use case termination: user logged in |
| Post Condition: Record of login in logs database |

|  |
| --- |
|  |

Fig 14. Initial ERD of logins databse

|  |
| --- |
| Fig 15 Partial Initial ERD for core database |
|  |

Fig 16 rest of initial ERD for core database

# Prototyping and Development

|  |
| --- |
|  |

Fig 17. Screen shot of prototype home screen

As a prototype a home screen has been created as well as a user administration screen where a limited number of user details can be entered and saved to the database. A message with a user id is presented to the user upon completion. A small amount of user data can also be retrieved and displayed in a simple table.

1. Implementation

Use case number one in the system design plan describes the requirement of allowing users to login to the application using a username and password which has been provided to them after their details had been entered into the system for the first time. As this requires to have user details entered and a hashed password generated and entered into the logins database it was decided to swap this with requirement number two which is the providing the ability to enter user information through a form.

Use Case 1: Insert User

The first step in the implementation of this requirement was to create a user interface which contains a table displaying current users in the system and buttons which when selected will use a Bootstrap Modal which will contain a form for user detail insertion. The HTML template used for this requirement and indeed most of the phase one requirements is userindex.jsp with commonmodals.jsp and head.jsp also included. Any JavaScript, JQuery and Ajax functions being called can be found in AddUser.js.

Once the new user form has been completed and the submit button has been selected the function insertUser() is called. This takes all the fields from the form and submits them along with an instruction id as Json data string though an Ajax call to the InsertData servlet which is which can be found in the controls class UserInsert.java. Having established a database connection through the DBManager.java class and checked the instruction id in the request the servlet then calls the relevant function from the UserInsertManager class which in turn calls the relevant function(s) in the dao class InsertUserData to perform the insert to the database.

A bit has also been sent with the request indicating if the data needs to be checked for duplicates. If the bit indicates the database is to be checked for duplicate user details the function CheckForDups() is called. If it is found similar details already exist in the database the function ReturnPosDups() is called to relay these details back to the user to enable them to abort the operation if it is the case that duplicate data exists. If the user confirms they wish to proceed the function InsertSingleUser() is called which extracts the fields from the request and creates a random number which will be used to create the users unique id in the database. It then calls the stored procedure InsertUser in the database passing the relevant fields into it. This stored procedure enters all details into the TB\_User table. Once the insertion has been complete this returns a newly generated id from the table. Depending on the users role within the organisation two letters are appended to the id which is to serve as the users login name. This is then returned to the InsertUser.java where it is used to create an entry to the tb\_userlogin table. Before inserting into tb\_userlogin the createHash() function from the security manager is called to hash the users date of birth which will serve as the users login password. If the user has been inserted successfully a confirmation message is returned to the user and the table displays the new users details.

Use case 1 a: Generate table of users.

This requirement specified that a table of users should be displayed when the user management screen is rendered. The Adduser.js function getUserData uses an Ajax call to the getUserData servlet in the controls class which calls the UserRetrieveManager function GetUserData which calls the GetUserFunction from the dao RetrieveUserData class. This function retrieves all user data from the database and associates two buttons to each row on for deletion and another to allow editing. A JQuery datatable[] usertable is then generated to display the data

Use Case 2: User Login

Use case number two specifies the requirement of allowing a user to log on with their own username and password which is to be compared to the hash values in the database. if authentication is successful their user id and role id should be returned to be used as a session cookie. The role id will determine which initial screen is to be presented to the user initially and to which areas the user should have access to throughout the application.

Upon loading the application the user is presented with a logon screen which provides a field for a user name and password. Oncethe relevant data has been entered and submitted this are passed to the LoginUser servlet in the controls class LoginUser.java. The LoginUser servlet then calls relevant function from RetrieveManager which in turn calls the relevant function from dao.RetrieveUserData to extract the relevant data from the database. Once this data has been returned the password supplied is hashed and compared to the hashed value returned from the database using the validatePassword() function in the SecurityManager.java class[ ]. If successful the user is directed to the appropriate screen based on their role otherwise they are returned to the login screen.

Use Case 3: Upload User Details

This use case was not developed during phase one development due to a combination of a lack of time and ineffective research. It was feared by proceeding with this requirement others may have been missed in phase two which were deemed more important for the overall functionality of the application. Once time permitted this requirement was researched further and was approached towards the end of phase two.

In order to facilitate the bulk upload of user data from .csv files a new button was added to the user interface. Upon selection of this button a modal appears with a file input. Once a file has been selected and submitted the file name is passed to the FileUploadServlet which handles the uploading and opening of the file as well as reading its contents. Each field except for the header fields are read into an array which is in turn passed to the InsertSingleUser function in the dao class InsertUserData for insertion to the database. This requirement at present is not considered completed an explanation for why is provided in the difficulties section for this phase.

Use Case 4: Delete User from System

This use case outlined the requirement of allowing users to be reactivated in the system having been deleted previously. Again an oversight during the design phase saw the need for the reordering of requirements. In order to restore a user they must first be marked as deleted. Due to this fact it was decided to develop the delete user functionality at this point

Data protection laws specify that data must be kept for a specific period of time and be accessible upon request. These laws also state that data should only be available for the purpose it was intended. To comply with these rules and to allow users who have previously attended education centres to progress to further course user details are marked as archived when deleted and so are kept in the system hidden rather than deleted altogether. In order to facilitate this functionality a delete button was added to each row in the user table. Once a button has been the relevant user id is selected from the table as is the row it belongs to and both are passed to the function complteUpdateDel in AddUser.js which presents are bootbox containing a request for confirmation from the user. Once confirmation has been received the function upDel is called which passes the user id to the InsertData servlet in UserInsert.java which uses the RestoreArchiveUser function in the UserInsertManager to call the dao function RestoreArchiveUser. This function starts by selecting the user’s role from tb\_user to ascertain if the user is a student or tutor. If the user is a student enties in tb\_user and tb\_student\_courses are marked as archived. If the user is a tutor only the relevant entry in tb\_user is marked as archived. Once all updates are complete on the database a success message is returned to upDel which displays it to the user and deletes the relevant entry from the table and redraws it.

Use Case: 5 Restore User

To allow for users to be re-instated on the system a second button was added to the user management button. When selected the current table is replaced by a one that includes details of users which have been marked as deleted in the past by using the same functionality developed for use case 1 a except providing all users which are marked as archived in the database. Each entry in the table now contains a restore button which when selected retrieves the user id from the table and the row on which it resides and passes them to completeUpadateDel in AddUser.js which asks the user to confirm their actions before passing the data to the upDel function which passes the data through an Ajax call to the InsertUser servlet InsertData servlet in UserInsert.java which uses the RestoreArchiveUser function in the UserInsertManager to call the dao function RestoreArchiveUser. Once a user has been restored successfully a confirmation will be presented to the user. The user should no longer be visible in the table of archived users and should be visible in table containing all user details.

An oversight in the design for this requirement was highlighted during user acceptance testing. It was pointed out that there was the lack of an option to restore multiple users at the same time. Time has not permitted this functionality to be added and so is marked for work in the future

Phase 2

Phase two mainly consisted of user requirements which concerned category and course management. The main user interface template for this phase is catcoursindex.jsp and its main js file is CatCourse.js. Each course which is run by the centre must be a member of at least one category. Courses can be assigned to different categories and each course name can be used in multiple instances providing they have a unique identifier.

Use Case 6: Insert Category

Requirement five specifies that users should have the ability to add a new course category to the system. This entails the user specifying a name and description for the new category. Once the relevant button on the user interface is pressed a modal is presented which contains a small form for the new category details. Once the relevant details have been submitted and the user confirms their intensions inserCatCourse() is called from CatCourse.js. This passes the data through an Ajax call to the CategoryCourses servlet which can be found in CategoryCourseInsert.java. This calls the dao function InsertCategory() through the CategoryCourse manager. InsertCategory() checks for any duplicate entries in the database by calling the stored procedure checkfordups() in the database. If a duplicate category name exists the operation is cancelled and a message stating so is returned to the user. Otherwise the data is entered into tb\_Category and a success message is returned to the user. The new category is now visible in the category table.

Use Case 6a: Generate table of categories

This requirement was developed in conjunction with use case 6. Its specification outlined that a table of all categories should be generated when the category management page is initially loaded and in event of any changes to the category table. The first function to be called in order to generate a categories table id getCatData which makes an Ajax callto the CategoryCoursesRetrieve servlet in the controls class CategoryCoursesRetrieve.java. This in turn calls the CategoryCourseRetieveManager function GetCatData which then calls the GetCatData in the dao class CategoryCoursesRetrieveData. This function retrieves all category information and associates three buttons to each row whose functionality are discussed in future uses cases. A JQuery datatable categories is then generated using the data returned through the function drawDataTable which has the tables rows and columns passed into it from the getCallData callback.

Use Case 7: Insert Course

This requirement specifies a user should be able to add a course to a category. For this requirement the user is to specify details such as what category the course should belong to. As stated previously course details can be used for different instances of that particular course. Instances may differ for different reasons such as what tutor is assigned to it or what capacity it will cater for. A button was added to the category and course home page which allows the user to view all course on offer at the centre. Category information is hidden at this point and a button to allow for the addition of a new course is shown. Once selected a modal presents a user form. This form includes selects which provide category name for which the new course can be assigned to. It also provides a list of tutors which can be assigned to the course. If an existing course is not to be used the user can specify a new course name. An existing accreditation body can be selected or if preferred a new one can be added. Other details such as the course level if applicable needs to be selected as does its capacity which must be under the organisations largest room size. Whether or not the course will require computing equipment must also be specified at this time Once the submit button has been selected and the user confirms their actions inserCatCourse() is called from CatCourse.js. This passes the data through an Ajax call to the CategoryCourses servlet which in turn calls the CategoryCourseManager which calls the insertCourse() function from InsertCourseData.java. This function calls the checkForDups stored procedure in the database which determines if there is a course with the same name and level in the database. If this is the case a message is relayed back to the user. If no duplicate exist a separate function InsertToCourses is called which calls the stored procedure addNewCourse in the database. This procedure determines if the course already exists in the database. If it this is not the case the procedure creates a new entry in TB\_Courses which will include the id of the category it belongs to, the course name, its level and accreditation body if applicable and whether or not the course will require equipment. The course id as well as the tutor id and capacity are inserted into tb\_tutorcourses. The course id is either the one sent from the servlet with the request or in the case of a new course being added the one that was generated during the insert. Once ted successfully a message is presented to the user and the new course is visible in the course table.

Use Case 7 a:

This requirement was developed in conjunction with use case 7. This requirement specified that a table of course should be generated when the course management page is initially opened and regenerated if any changes are made to courses within it. The first function called is getCourseDetailsForCourses in CatCourse.js.This function uses the same servlet and java classes as use case 6a just using different functions within these classes to retrieve the relevant rows of data from the database and associate three buttons for each of them. Once the relevant data has been returned a jQuery datatable is generated by the function drawDataTable1 to display the returned information.

Use Case 7 b: View all courses

This functionality was also developed in conjunction with use case 7. In order to initialise and display the course table discussed in use case 7 a new button was added to the button bar in on the category management interface. Once clicked the category table is hidden and the courses table displayed instead. The “Add Category” and “View Courses” are then hidden and two new buttons are displayed “View Categories” and “Add Course”.

Use Case 7 c: View all categories

This functionality was developed to allow a user to return to the categories management screen with the click of a button. Once the “View Categories” button is clicked the course table is hidden and the categories table is shown and the buttons are displayed and hidden opposite to as they were in use case 7 b.

Use Case 8: Insert Course from Category Table

This piece of functionality only became a requirement during development. It was realised that it would be make it easier for a user if they had the option to select a category from the table and add a course directly to that category rather than having to navigate to the course management screen. In order to facilitate this requirement a button is added to each table row as they being generated. Once selected a modal is presented to the user containing a form for the new course details. The category name is pre-populated in the form. If the category currently contains course those names are available for selection by the user. Once the submit button is selected the application carries out the same process as use case six.

Use Case 8: Course count in category table row. Display courses with click of a button

Again this functionality became a requirement during development. It was realised that it would be useful for administration staff to be able to view all courses associated with a category rather than having to search through a full list of courses. to facilitate this piece of functionality a count of courses is selected along with other category information. If this count is greater than zero a clickable button is provided to enable the generation of a courses table with only the courses associated with the particular category.

Use Case 8: Edit Category

This requirement specifies that a user should have the ability to change a categories name and description. To facilitate this functionality an edit button is available on each line of the category table. Once selected the id, category name and description are selected from the category table and are added to a form in a modal to allow editing. Once any details have been edited and the save button has been pressed the function updateCatCourse is called which retireives the data from the table and passes to the CategoryCourses servlet in CategoryCourseInsert.java from where the CategoryCourseManager.java function is used to call the dao function RestoreArchiveCourseCat to carry out the update. If successful a message is returned to the user and the category table is updated to display the new information.

Use Case 9: Delete a Category

This use case specifies that a user should be able to delete a category. This requirement was originally developed allowing any category to be deleted. Testing of this requirement posed highlighted that no consideration had been given to categories and what happens to these when a category is deleted. As a result this requirement was redeveloped and only categories that had no active courses associated with them are made available for deletion by rendering the delete button inactive on ones that have. When the delete button is selected the category of that to be deleted is selected from the category table and passed into ComplteUpDel in CourseCat.js which when called prompts the user to confirm their actions. Once confirmed the function upDel and the same steps as Use Case 8 are used to flag the relevant categories as archived in tb\_category the only difference being the prepared statement in RestoreArchiveCourseData.

Use Case 10: Edit Course Information

This use case specified that a user should have the ability to edit course information. An oversight during the design phase resulted in a lack of definition of what exactly should be editable by the user. Initially this was requirement was developed similar to that of the edit category requirement in that the courses name and capacity could be edited. On reviewing this requirement is was proposed that allowing the courses tutor to be changed as well as its capacity would be more logical. During testing it was realised that by allowing courses that have been timetabled in active timetables could lead to room and tutor availability issues. As time was limited the simpler option of only allowing courses not included in an active timetable was used to protect against such problems.

Once the edit course button is clicked the tutor id for that course which has been set as an attribute on the button is retrieved and passed into the getTutorsSels function which will retrieve all available tutor names and populate the tutor select with them ensuring the current tutor is the selected one. It also retrieves the courses current capacity and display it in an input field to allow editing. A modal with containing a form of editable information is the displayed to the user Once the courseUpdateButton has been clicked the function updateCourse is called which passes the instruction id, course id, tutor id and capacity to the CategoryCourse servlet which determines by the instruction id that the CategoryCourseManager function UpdateCatCourse should be used to call the UpdateCourse function from the dao class UpdateDeleteCatCourseData to perform the update.

Use case 11: Delete Course

This use case specifies that a course can be marked as deleted by an administrator. Again testing provided the realisation that courses currently running should not be removable and so delete buttons for these have been disabled. Once the delete button on the appropriate row has been clicked the course id and row are passed into ComplteUpDel which asks the user to confirm they wish to proceed with the deletion before calling upDel which uses the same servlet and Java function calls as use cases nine and ten to carry out the deletion.

Use case 12: Add student to instance of a course

This requirement specifies that a member of administration staff should have the ability to select a course and add registered students to that course. To facilitate this functionality a new button was added to the course table. Once clicked the getUserForCourses function uses the GetUserData servlet to retrieve details of students not already registered on that course. These details are displayed in a new table each row having an associated tick box. Once a box is ticked the users id is added to an array. Once all students have been selected the “Add Selected Students” button needs to be clicked to add the students to relevant course. Once clicked the function is called which passes the array to the StudentData servlet contained in the controls java class StudentDataInsert which calls the StudentDataManager function InsertStudenttoCourse which calls InsertStuData from the dao class InsertStudentData. This takes the array and builds a comma separated string which is passed along with the size of the array into the stored procedure InsertStudentData in the database. This stored procedure starts by taking the first element of the element in the string which is the course id. It then parses the string taking each student id and adding it along with the course id to tb\_student\_courses. The length of the array which was passed into the stored procedure is used to ensure all students are added successfully by comparing it to a count of insertions. This figure is sent back to the program as an acknowledgement. If the figures match up a success message is provided to the user otherwise the procedure is rolled back and an error message is provided to the user.

Use case 13: View students from courses table and add/edit their grade

This requirement specifies that a member of administration staff should have the ability to view all students in a particular course and be able to add or edit their final grade for that course. In order to facilitate this functionality count students registered to a particular course is selected along with other relevant course information. If this count is over zero an active button is also included on the table row which when clicked calls selects the course id from the table before paasing it into getUserForCourses to retrieve students who are registered on that course. As well as the students name and id each table row contains their current grade in an imput field and an edit button which when selected selects the course id, student id and grade and passes them into updateStudentData. The data is then snetsent to the StudentData servlet in the controls class StudentDataInsert.java. From the instruction id sent with the request the servlet determines that StudentDataManager function UpdaterStudentRes should be called which in turn calls UpdateStuData in the dao class InsertStudentData to perform the update on the database which returns a message stating whether the operation was successful or not.

Use case 14: View/ edit student or tutor information from the users table

This requirement specifies that a member of administration staff should have the ability to select a tutor or student from the users table and view their personal details and performance records. They should also have the option of editing certain details in the case of student edit their grade.

To provide this functionality a button is provided on each row in the users table. Once clicked the information from the row in question is placed in a form and presented to the user using a modal. The name fields have been disabled but all other information is editable. There is also a button on the modal which when clicked calls getUserRecs in AddUser.js. This function determines an instruction id based on the whether a student or tutor has been selected.= and passes this along with the selected id to the GetUserData servlet in UserRetrieve.java. The servlet uses the GetRecs function in the UserRetrieveManger class to call the GetRecs function in the dao class RetriveUserData.java. the select statements run against the database depend upon whether a tutor or student had been selected. In the case of a student GetRecs returns all of the courses they have participated in complete with start and end dates as well as their grade in an editable input. This input also has a button associated with it in order to allow the grade to be edited. If data is being retrieved for a tutor details regarding all the courses they have taught are returned as well as an average mark achieved by students in each.

Grades are edited independently and only change if the individual edit button is selected. Once this button is selected the function updateStudentGrade is called which uses the an Ajax call to pass the updated data to StudentData which carries out the update using the same process as described in use case 13.

The rest of the data is updated when the “Update” button is clicked. The button click calls updateUser which sends the data to the InsertData servlet in the controls class UserInsert.java uses the UserInsertManager to call the UpdateUser function from the dao class UpdateDeleteUserData.java class which performs the update and returns a message to whether it was successful or not.

Phase 3

The requirements in the phase were intended to provided functionality to allow timetables and schedules to be created to allowing student attendance to be stored and tracked. The main view page for this phase is timetablecreator.jsp and the main js file is

Use case :

The first requirement in this phase specified that a member of administration staff should have the ability to create a new timetable and to save it as a draught or as an active timetable. The first step in the development process for this step was to create a user interface which would allow the user select required fields and add them to a work in progress timetable. Four selects where added to the page one for the required day, one for the required time another for the required course and the final one for the required room. Initially only the day select is active and has been populated through a call to the function getDataForSelectors. This function makes call to the TimeTableRetrieveServlet which is in the controls class TimeTableRetrieve.java. This uses the TimeTableRetriveManager to call the GetTT function from the dao class RetriveTimeTableData.java. This process is used in order to populate each select when called individually. The selects statements used differ in relation to what selects are being populated and take into account what has been selected previously through different where statements that perform checks using different formulae.

Once a day has been selected the day select is disabled and the times select activated having been populated with available timeslots from the database. Again the same process occurs when a time has been selected in that the select is disabled and the course select enabled having been populated by with courses which don’t already occur at that particular timeslot and who’s tutor doesn’t already have a class at that time. The total teaching hours for that particular course are also taken into account during the select. Once a course has been selected that select is disabled and the rooms select enabled having been populated by available rooms for that particular time period and that can provided the computer equipment if the courses requires it.

Once all selects have been populated the new entry can be added by click the add button. Once this button is clicked the selects are refreshed as is the table and addNewCellObject is called. This function takes the ids from the selected options and uses an Ajax call to the TimeTableInsert servlet including them in the request. The servlet then uses the function TimetableManager function InserTTData to interact with the InsertTTData function in the dao class InsertTimeTableData to insert the data into the tb\_timetabletemp table. This table is used to store work in progress timetables.

If an insertion is successful a message code of 1 is sent back to the servlet. The TimeTableManager function GetTimeTable is then used to call the RetriveTimeTableData,java function RetireveTimeTable. This function starts by declaring a multidimensional array called slots which will be used to store sells of data regarding timetabled course assigned to a certain slot. An array of ints slotcnt is also declare which will be used to track the amount of entries for each timeslot.

This function operates by selecting data from the database one day at a time populate the relevant time slots after each select. Once all the data for a particular day has been processed a loop is used to go through the slots multidimensional array by selecting each row and taking each column in that row to create cell objects to be added to a timetable object. This will enable the objects to be displayed as rows in a table. Once all the data in tb\_timetabletemp has been processed the time table object (TimeTableObjects ttdata) is passed back to the servlet from where it is passed back to the calling js function which returns defines the timetable headers and returns those and the returned data as callback which is used by the function drawtimetable to generate the timetable using a JQuery datatable. A day id is included in the table but is hidden as it is only used for orsdering purposes. Other table features such as search are also hidden.

At this point the new entries in tb\_timetabletemp are not marked as saved. They will only be saved and committed when the user chooses the “Save as Draft” button. When this button is clicked the function saveTT is called. This makes an Ajax call to the servlet TimeTableInsert which call the TimeTableMnager function SaveTTasWIP to mark the new entries as saved in the database.

A second button is available on this interface which gives the option to the user to save the timetable as the new current timetable. Once pressed a modal appears with a select question asking if the user would like to assign the timetable to an existing semester or to define a new one. If yes is selected the function getmesterDetails is called to retrieve through a call to the TimeTableRetrive servlet details of any existing semesters such as its start and end date. This datra is used to populate a table on the modal with a checkbox on each row to allow the user to make a selection.

If the user chooses no a small form with is provided to allow the user to provide details such as which semester the timetable is for and it’s start and end date. If the “Save New Semester” button is clicked InsertSem is called to make a call to TimeTableInsert to add the new semester details to the database. Another select now appears on the modal asking the user to choose if they would like to add holiday periods for the recently saved semester. If the user chooses yes two date fields are shown to allow the user enter start and and dates for the holiday periods. Any amount of holiday periods can be added taking into account the different circumstances which may result in classes not being held.

# Testing

Whitebox

(Testing carried out by a tester with indebt knowledge of code how it is supposed to perform and expected results.)[38]

### Unit testing

As stated previously the AUP model for software development will be used as a guide for this development process. As per the project design plan development will be carried out in phases which each phase containing sub phases which contain individual components for development. Once developed the new component will be tested by the developer to ensure it performs as expected. Before any development commences on a particular requirement clear guidelines on what it should and shouldn’t deliver will be developed to aid testing. Testing will not only be carried out to ensure the particular feature works but also to ensure it cannot be broken.

Performed Unit Testing

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Phase | Req ID | Test Case # | Test | Round | Result |
| 1 a | 1 | 1.0.0 | Insert User | 1 | Fail |
| 1 a | 1 | 1.0.1 | Insert User | 2 | Fail |
| 1 a | 1 | 1.0.2 | Insert User | 3 | Pass |
| 1 a | 1 | 1.1.0 | Insert Duplicate User (proceed) | 1 | Pass |
| 1 a | 1 | 1.1.1 | Insert Duplicate User (abort) | 1 | Pass |
| 1 a | 1 | 1.2.0 | Constraints Check | 1 | Fail |
| 1 a | 1 | 1.2.1 | Constraints Check | 2 | Pass |
| 1 a | 2 | 2.0.1 | Secure User Login | 1 | Pass |
| 1 a | 2 | 2.0.2 | Secure User Login | 2 | Fail |
| 1 a | 2 | 2.0.3 | Secure User Login | 3 | Pass |
| 1 a | 4 | 4.0.0 | Delete User from System | 1 | Fail |
| 1 a | 4 | 4.0.0 | Delete User from System | 1 | Fail |
| 1 b | 5 | 5.0.0 | Restore User | 1 | Fail |
| 1 b | 5 | 5.0.1 | Restore User | 2 | Pass |
| 2 a | 5 | 5.0.0 | Insert category | 1 | Pass |
| 2 a | 6 | 5 a | Insert course | 1 | Fail |
| 2 a | 6 | 5 b | Insert course | 2 | Pass |
| 2 a | 6 a | 6 | Insert course from on category | 1 | Fail |
| 2 a | 6 a | 6 b | Insert course from on category | 2 | Fail |
| 2 a |  |  |  |  |  |

Testing was carried out continuously during development. Many issues where encountered during development mostly due to a lack of knowledge regarding JQuery, Ajax and JavaScript. Initial research had been carried out into these technologies but as with most new skills the best way to test new skills is to use them.

The table above includes testing results from unit test cases carried out by the developer after each sub phase had been completed. These tests were carried out on the newly developed piece of functionality in isolation. Details of all integration testing are outlined in the next section.

Phase 1 Unit Test Analysis

The first piece of development carried out was on the requirement which specifies a user should have the ability to add a user through a digital form. As the table shows the first round of testing resulted in failure. This was due to the fact that no date of birth was visible in the database. After some quick research it was discovered that the database expects the date to be added in the year month day format due to its default configuration. Upon further testing it was realised that the temporary external ids which were generated using a random numbers where not being updated with the newly generated database id’s which resulted in users having the same external id which caused an error during selection as more than one result was being returned. The first test also failed due to the fact the user table did not reflect the changes carried out even though they were present in the database. A substantial amount of research was required to find the most effective and efficient way to carry this out.

The next set of unit testing was checking the systems response when a duplicate first name and surname were entered in the form. Testing was carried out to ensure the user could proceed and insert the data or abort and close the form. Both tests passed. Tests were also carried out to ensure data integrity constraints on the user form were performing as expected. the initial test carried out failed as the constraints were too sever in some cases and didn’t work at all on the email address question.

The nest requirement to be tested was the ability to login with a secure password and username. The first round of testing carried out did not included the Security Manager functionality which hashes passwords when entering them into the database and un-hashes them when retrieving. This first round of tests passed. When the hashing functionality was added the next test failed. This was due to the password comparison function which contained a mistake in how the password and username was retrieved from the database for comparison. Once this issue was rectified the third test passed.

Testing Use Case 4

The ability to delete user information from the system was the next piece of functionality developed and tested. The first round of testing failed as user course data was not flagged as archived in the tb\_student\_courses table it was a case the user was a student and the table on the user interface did not refresh. Upon investigation it was realised that the SQL statement written to flag the the entries in tb\_student\_courses was referencing the wrong table.

Testing Case 5

The next requirement to be developed was allowing a user which had previously been deleted from the system to be restored. The initial tests on this piece of functionality failed due to the user interface table not being updated, the solution for this problem was the same as for other requirements.

Test Case 9: Edit Course Details

As described in the implementation for this requirement testing brought the realisation that changing course details could cause possible knock on effects in the system with regards to timetabling. It was realised that if a courses capacity was changed the room allocated in the timetable may not be big enough to cater for the class. In the case that a different tutor is assigned to the course this could cause a clash in the schedule in that the tutor may be assigned two classes at the same time.

Testing on the updated functionality initially failed on this piece of functionality due to the fact the tutor select wouldn’t populate with the current tutors name as its selected option. Having carried out extensive research no solution performed as desired and so it was decided to perform the action when the tutor selects initially by passing the tutor id into getTutorsForSels and adding an option with the corresponding tutor name once all the options have been added.

Test Case 10: Delete Course

The first round of testing of this functionality failed as although the table was being refreshed correctly when the application was refreshed or restarted the deleted entry was once again visible. Upon investigation it was found the select statement for the courses table didn’t exclude courses marked as archived

User acceptance- when submitting grades insure when finished

Add students – array made null rather than new array

View students in course when looking at specific coursers – should return back to courses not all courses

|  |
| --- |
| **Test Case Number:** 1 |
| **Test Case Name:** Edit Course Details |
| **Purpose:** To ensure course details can be edited successfully |
| **Procedure Steps:**   1. Select add user in the users screen 2. Fill out typical user information 3. Submit 4. View message |
| **Expected Results:**   1. **Success Message received** 2. **All user details in TB\_User** 3. **Same user in tb\_Logins** 4. **New user visible on screen** 5. **Ability to logon with user name and password** |
| **Results**  **1: Pass**  **2: Pass**  **3: Pass**  **4: Pass**  **5: Fail** |

### Integration testing

Once unit testing has been completed successfully on a particular component it will be integrated into the existing system and tested to evaluate its interaction with existing hardware and software. As well as testing Interoperability this testing will focus on the affect on performance and security after integration.

## Blackbox

(Testing carried out by a tester with no knowledge of the code and how it is supposed to perform)[39]

Once a sub-phase has been completed it will be released to users for testing. These users will have no knowledge of the code. Test-cases will be developed to cover different scenarios.

Round 1.

Integration Testing

|  |
| --- |
| **Test Case Number:** 1 |
| **Test Case Name:** Insert user data |
| **Purpose:** To ensure data for a new user can be added to the database through the application |
| **Procedure Steps:**   1. Select add user in the users screen 2. Fill out typical user information 3. Submit 4. View message |
| **Expected Results:**   1. **Success Message received** 2. **All user details in TB\_User** 3. **Same user in tb\_Logins** 4. **New user visible on screen** 5. **Ability to logon with user name and password** |
| **Results**  **1: Pass**  **2: Fail – no dob**  **3: Fail**  **4: Fail**  **5: Fail** |

Round 2

|  |
| --- |
| **Test Case Number:** 1 |
| **Test Case Name:** Insert user data |
| **Purpose:** To ensure data for a new user can be added to the database through the application |
| **Procedure Steps:**   1. Select add user in the users screen 2. Fill out typical user information 3. Submit 4. View message |
| **Expected Results:**   1. **Success Message received** 2. **All user details in TB\_User** 3. **Same user in tb\_Logins** 4. **New user visible on screen** 5. **Ability to logon with user name and password** |
| **Results**  **1: Pass**  **2: Pass**  **3: Pass**  **4: Pass**  **5: Fail** |

|  |
| --- |
| **Test Case Number:** 1 |
| **Test Case Name:** Insert user data |
| **Purpose:** To ensure data for a new user can be added to the database through the application |
| **Procedure Steps:**   1. Select add user in the users screen 2. Fill out typical user information 3. Submit 4. View message |
| **Expected Results:**   1. **Success Message received** 2. **All user details in TB\_User** 3. **Same user in tb\_Logins** 4. **New user visible on screen** 5. **Ability to logon with user name and password** |
| **Results**  **1: Pass**  **2: Pass**  **3: Pass**  **4: Pass**  **5: Pass** |

|  |
| --- |
| **Test Case Number:** 1a |
| **Test Case Name:** Insert duplicate user data |
| **Purpose:** To ensure data for a new user can be added to the database through the application |
| **Procedure Steps:**   1. Select add user in the users screen 2. Fill out form with typical user information except using same name as previously 3. Submit 4. View message |
| **Expected Results:**   1. Message provided to the user that there is a possibility of a duplicate entry in the system 2. User able to submit data |
| **Actual Result:**   1. **Pass** 2. **Pass** |

|  |
| --- |
| **Test Case Number:** 1b |
| **Test Case Name:** Ensure data integrity checks |
| **Purpose:** To ensure user cannot enter incorrect data type into fields |
| **Procedure Steps:**   1. Select add user in the users screen 2. Enter number into name fields 3. Skip address 1 filed 4. Enter single word into email field |
| **Expected Results:** user shouldn’t be able to proceed without correcting mistakes. Clear error message provided. |
| **Actual Result:** |

|  |
| --- |
| **Test Case Number:** |
| **Test Case Name:** Add new Category |
| **Purpose:** To Ensure New Categories can be entered |
| **Procedure Steps:**   1. Select add category from category home screen 2. Fill in name and description of category 3. Submit 4. View message |
| **Expected Results:**   1. **Success Message received** 2. **New category in tb\_category** 3. **New category visible on screen** |
| **Results**  **1. Pass**  **2. Pass**  **3. Fail** |
| **Notes:**  **Table not refreshed** |

|  |
| --- |
| **Test Case Number:** |
| **Test Case Name:** Add new Category |
| **Purpose:** To Ensure New Categories can be entered |
| **Procedure Steps:**   1. Select add category from category home screen 2. Fill in name and description of category 3. Submit 4. View message |
| **Expected Results:**   1. **Success Message received** 2. **New category in tb\_category** 3. **New category visible on screen** |
| **Results**  **1. Pass**  **2. Pass**  **3. Pass** |
| **Notes:** |

|  |
| --- |
| **Test Case Number:** |
| **Test Case Name:** Add new Course from Category Row |
| **Purpose:** To Ensure New Categories to a category when selected from a table |
| **Procedure Steps:**   1. Click on plus sign in desired row 2. Fill in/select details 3. Submit form 4. View message |
| **Expected Results:**   1. **Success message** 2. **Course in TB\_courses if new** 3. **New entry in tb\_tutor\_courses** |
| **Results**   1. **Fail** 2. **Fail** 3. **Fail** |
| **Notes:**  Select for tutor not populated |

This test failed during the first round of testing due to the fact that the tutor and course name selects did not get populated. This issue was rectified and the test was carried out again.

Above are examples of test cases which will be provided for testers. Testers will be made up of members of staff from an education centre, plus another who works in administrative role for an assurance company and has vast experience using data gathering applications, a software engineer as well as one tester who has little experience of software applications or IT in general. The main aim of this type of testing is to ensure the application meets the user requirements and is adhering to the usability criteria guidelines set out in Nielson’s Heuristics. Tests that result in unwanted results will are included to ensure firstly error are caught and also that users are informed with clear easy to understand error messages. Extremely short surveys will be used to analyze the testers overall thoughts of the application.

## Quality Assurance and User Acceptance Testing

Once the application nears completion it will be released to a set of users for the equivalent of user acceptance testing. Testers will be asked to use the application as if they were performing everyday work duties. Once testing has concluded surveys were sent out the tester will be asked to complete a survey which again will be used to analyze the applications overall performance.

# Issues and risks

An overall analysis of the developer’s academic performance over the past few years would highlight a weakness when it comes to development. For one reason or another developer has struggled in certain aspects of this discipline mostly in user interface design and development. A large attraction in developing a web application rather than another type of application was the opportunity to improve these development skills. Adding to the challenge is the choice of using JavaScript and JQuery to develop the user interface. The developer has no previous experience in these technologies but is keen to utilise them as they are widely used within industry today. In order to overcome these issues the developer has been actively practising implementing user interfaces with these technologies and plans on further enhancing theses skills over the coming weeks.

Although no one part of this project should be overly complex there are many pieces of functionality which need to be integrated which may all contain complexities and present issues when integrating which could present unforeseen time issues. As shown below a comprehensive plan has been developed which has split the work into phases and sub-phases each containing units of development. These phases have been planned so that the most important are developed first working down to least important. In phase “” future work has been included in the off chance there is some spare time available. it will be extremely important to manage time and stick to this plan as rigorously as possible to ensure the application contains some substantial functionality.

# Plan and future work

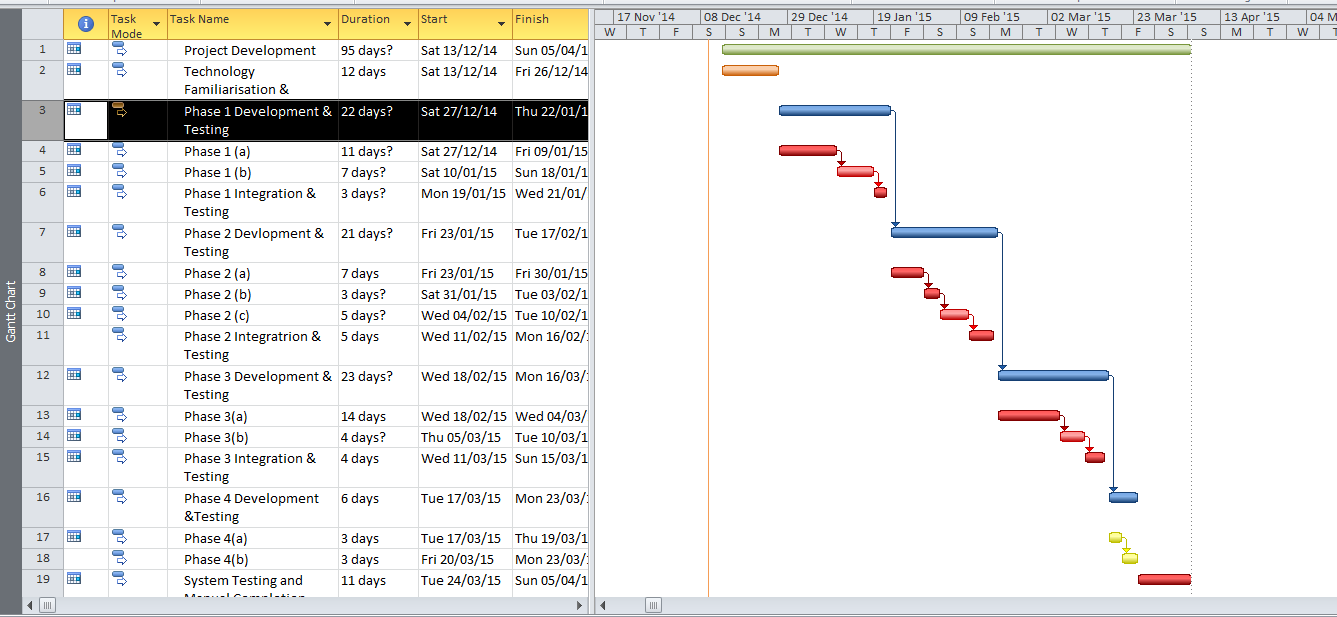
****

Fig 18. Gant chart displaying schedule of future work

# Conclusions

At the beginning of this project the main aim was to try and design and develop an application which would be useful to others and make their lives a little easier. A conscious effort has been made through requirements gathering and analysis to only include features that will be utilized and add worth to the application. Through personal observations and thoughts an idea of requirements had been formulated. In order to achieve my initial aim brainstorming sessions and interviews were set up and surveys were sent out to perspective users. It is felt that through these methods and research into existing systems that a relevant and achievable set of features have been outlined.

A personal goal in completing this questionnaire was to be left with a very clear idea of what was required and what would be needed to achieve these requirements. A good lesson has been learnt from filling out this form in that time should not be taken for granted and even the simplest of task can take a lot longer than expected. Even though a lot of work has been put into completing this form there are certain aspects that could be improved greatly. Going forward it is realised when researching notes should be recorded properly as should references as it will make things a lot easier in the future.

It is felt through the design of a comprehensive design and test plan that all crucial requirements are achievable within the project time frame. It is clear that discipline as well as organisational and time management skills will be will be crucial in order to get the project over the line but if these skills are used and a plan is stuck to success should be attainable.

Implementation

Phase 1(a)

Secure Login

Phase 2(b)

Manual Insertion of User Data

Further referneces

[37] <http://www.ambysoft.com/unifiedprocess/agileUP.html>.

[38] <http://www.webopedia.com/TERM/W/White_Box_Testing.html>

[39] <http://www.webopedia.com/TERM/W/Black_Box_Testing.html>