

Contents

[Analysis 5](#_Toc479148151)

[Overview 5](#_Toc479148152)

[Problem Identification 5](#_Toc479148153)

[Functional and non-functional requirements 6](#_Toc479148154)

[Non-Functional Requirements 8](#_Toc479148155)

[System Methodology 9](#_Toc479148156)

[Waterfall 9](#_Toc479148157)

[Extreme Programming 10](#_Toc479148158)

[Rapid Application Development 11](#_Toc479148159)

[Scrum 11](#_Toc479148160)

[Dynamic System Development Method 12](#_Toc479148161)

[Conclusion 12](#_Toc479148162)

[Gantt chart 13](#_Toc479148163)

[Design 14](#_Toc479148164)

[Identification of Entities 14](#_Toc479148165)

[ER Diagram 15](#_Toc479148166)

[Class Diagrams 16](#_Toc479148167)

[Entity Standard Notation 19](#_Toc479148168)

[1NF 19](#_Toc479148169)

[2NF 19](#_Toc479148170)

[3NF 19](#_Toc479148171)

[All Standard Notations 3NF 20](#_Toc479148172)

[Data Dictionary 21](#_Toc479148173)

[Student 21](#_Toc479148174)

[Tutor 22](#_Toc479148175)

[ExamEntry 23](#_Toc479148176)

[ExternalExam 23](#_Toc479148177)

[BlockBooking 24](#_Toc479148178)

[Room 24](#_Toc479148179)

[TuitionChoice 25](#_Toc479148180)

[TuitionFee 25](#_Toc479148181)

[ExamResult 25](#_Toc479148182)

[Payment 26](#_Toc479148183)

[BlockBooking 28](#_Toc479148184)

[Report 28](#_Toc479148185)

[TimeTabledLesson 30](#_Toc479148186)

[TutorTakes 30](#_Toc479148187)

[Storyboards 31](#_Toc479148188)

[Student 31](#_Toc479148189)

[Tutor 33](#_Toc479148190)

[Payment History 35](#_Toc479148192)

[Lesson Timetable 36](#_Toc479148193)

[Room 37](#_Toc479148194)

[Exam Entry 38](#_Toc479148195)

[External Exam 40](#_Toc479148196)

[Walkthrough and overview of application 41](#_Toc479148197)

[Menu 41](#_Toc479148198)

[Modify Students form 41](#_Toc479148199)

[Modify Tutor form 42](#_Toc479148200)

[Modify Exam form 43](#_Toc479148201)

[Modify Room form 44](#_Toc479148202)

[Modify Exam Entries form 45](#_Toc479148203)

[Modify Block Booking form 46](#_Toc479148204)

[Past Payments form 47](#_Toc479148205)

[Lesson Timetable form 48](#_Toc479148206)

[Modify Lessons 49](#_Toc479148207)

[Report on Students 50](#_Toc479148208)

[Report on Tutors 50](#_Toc479148209)

[Report on Bookings 50](#_Toc479148210)

[Report on Payments 50](#_Toc479148211)

[Implementation 51](#_Toc479148212)

[Justification of Solution 51](#_Toc479148213)

[Why C# 51](#_Toc479148214)

[Why SQL 53](#_Toc479148215)

[SQL Tables 54](#_Toc479148216)

[Implementation of C# classes 58](#_Toc479148217)

[Testing 97](#_Toc479148218)

[Evaluation 113](#_Toc479148219)

[Evaluation of Technique 113](#_Toc479148220)

[Use of waterfall-DSDM 113](#_Toc479148221)

[Evaluation of Self Performance 113](#_Toc479148222)

[Time Management 113](#_Toc479148223)

[Planning 113](#_Toc479148224)

[Skills developed 114](#_Toc479148225)

[Problems encountered and how they were dealt with 114](#_Toc479148226)

[Backup routine 115](#_Toc479148227)

[Testing 115](#_Toc479148228)

[Future development 116](#_Toc479148229)

[Overall 116](#_Toc479148230)

[User experience improvements 116](#_Toc479148231)

[Efficiency improvements 116](#_Toc479148232)

[Feature improvements 116](#_Toc479148233)

[Additional features 117](#_Toc479148234)

[Strengths and weaknesses of the project 118](#_Toc479148235)

[Strengths 118](#_Toc479148236)

[Weaknesses 118](#_Toc479148237)

[References to resources used within project 119](#_Toc479148238)

[Code 119](#_Toc479148239)

[Post Code regular expression code 119](#_Toc479148240)

[Telephone number regular expression code 119](#_Toc479148241)

[Additional sources 119](#_Toc479148242)

# Analysis

## Overview

The Mitchell School of Music is an educational service provided by Peter Mitchell who runs the business. The school space is a large structure with 12 practice rooms, 4 classroom sized rooms and a hall. Formerly he was running this service from his residence but due to the businesses success he has since moved to this new building due to the significant increase in space over the previous location. The service provided to the students of his school is the opportunity to be taught a wide range of instruments of their choice by a wide range of professionals in the areas of these instruments of a variety of skill levels.

All private lessons are now running at the school and the active hours are now 1-9 on Monday to Thursday and 1-8 on a Friday. There are three significant parts to Peter’s business, the private tuition as previously mentioned, a specialist weekend school and a specialist summer school while also providing a service for instrument hiring along the side, available to any student presently taking classes at the school.

Peter Mitchell has more recently decided that he wishes to computerise more within his business so as to improve efficiency and so he can keep track on all aspects of his business. As it stands in its current state, Peter finds it extremely difficult to manage his business in its current state and believes that doing this would significantly help in keeping all the information relating to his business up to date, he wishes to do this by computerising records of data such as students, tutors and the scheduling of his school.

## Problem Identification

* **Paper-based record keeping**

Keeping paper based records is rather wasteful in terms of upkeep, accuracy and time.

* Once records are made there is no way of looking without going over old records as to if there is a conflict of data with a record made in the past, this process can be automated by computer however making it more efficient in terms of accuracy and speed.
* Paper based records can be easily lost, destroyed or damaged beyond use, while computer records can be stored indefinitely, allowing them to be called upon at any time as well as them being easily copied to ensure they are not lost and so backups can be made.
* Data entry is a long process that could be easily sped up.
* **Manual checking of data**

Checking old records is a highly time consuming process during which mistakes are often made.

* As a result of data needing to be checked by going over old records, mistakes are made and discrepancies and inconsistencies are created causing trouble for students, tutors and the record keepers themselves.
* **Making mistakes**

Mistakes are made much too often which could be severely lessened with the use of computers.

* Mistakes over records can be rather costly to a business and should be limited as much as possible, they damage the image of the business and as a result damages the profit as they are not running as effectively as they could be.
* **Time wasting**

Significant time is wasted on current system.

* Having an entire system computerised allows all data to be stored, collected and sorted however the user would like, improving ease of access, this seems to be an issue with the current system considering the amount of paperwork involved.
* **Lost data**

Some data may be lost as a result of being considered insignificant.

* Data can be recorded as data is parsed through the system, making every piece of data equally as significant as the other.
* Records can be kept for as long as deemed necessary by the business.
* **Poor/difficult management**

Management would be easier through a computerised system.

* Data can be recorded and visually presented in a much more adaptable and understandable way. Allowing organisation of business operations to be much faster.
* **Potential for overbooking**

Bookings can overlap in rooms and students and tutors can be busy on times when classes are booked.

* A computerised system can be designed so as mistakes such as this are impossible and only unique values can be set under certain conditions.

## Functional and non-functional requirements

1. Menu
   1. The main menu must be the initial form to load and appear in the centre of the screen for use.
   2. The layout for the menu must be easily understandable to the average user allowing them to easily navigate between forms from the menu screen.
   3. The menu must provide utilities through which other forms can be accessed and used.
   4. Upon exiting through any means, the form must shutdown the application entirely and leave no background processes running from the program.
2. Student form
   1. The student form must allow for the entering of new students into the table as long as data entered is within acceptable parameters.
   2. The student form must be able to create new students to add to the student table through pressing the create button.
   3. The student form must be able to remove students from the table through pressing the delete button.
   4. The student form must generate a new student number for each new student that is created by the user.
   5. The student form must prevent the user from entering clearly incorrect information through the use of masked textboxes and combo boxes.
   6. The student form must ensure that all data entered is valid within the context of the field it is being entered for, this data must, if invalid, provide the user with feedback specifying what went wrong with the data they entered.
   7. The student form must allow the user to clearly see all students currently stored within the SQL students table.
   8. The student form must allow the user to select a student from the students table in order to allow for that student’s data to be edited, updating the relevant fields within the form for editing.
   9. The student form must allow the user to sort students by any field in ascending or descending order.
   10. The student form must allow for the user to search specific fields for specific students.
3. Tutor form
   1. The tutor form must allow for the entering of new tutors into the table as long as data entered is within acceptable parameters.
   2. The tutor form must be able to create new tutors to add to the tutor table through pressing the create button.
   3. The tutor form must be able to remove tutors from the table through pressing the delete button.
   4. The tutor form must generate a new tutor number for each new tutor that is created by the user.
   5. The tutor form must prevent the user from entering clearly incorrect information through the use of masked textboxes and combo boxes.
   6. The tutor form must ensure that all data entered is valid within the context of the field it is being entered for, this data must, if invalid, provide the user with feedback specifying what went wrong with the data they entered.
   7. The tutor form must allow the user to clearly see all tutors currently stored within the SQL tutors table.
   8. The tutor form must allow the user to select a tutor from the tutors table in order to allow for that tutor’s data to be edited, updating the relevant fields within the form for editing.
   9. The tutor form must allow the user to sort tutors by any field in ascending or descending order.
   10. The tutor form must allow for the user to search specific fields for specific tutors.
4. Exam Entry form
   1. The exam entry form must allow for the entering of new exam entries into the table as long as data entered is within acceptable parameters.
   2. The exam entry form must be able to create new exam entries to add to the exam entry table through pressing the create button.
   3. The exam entry form must be able to remove exam entries from the table through pressing the delete button.
   4. The exam entry form must generate a new exam entry number for each new exam entry that is created by the user.
   5. The exam entry form must prevent the user from entering clearly incorrect information through the use of masked textboxes and combo boxes.
   6. The exam entry form must ensure that all data entered is valid within the context of the field it is being entered for, this data must, if invalid, provide the user with feedback specifying what went wrong with the data they entered.
   7. The exam entry form must allow the user to clearly see all exam entries currently stored within the SQL exam entries table.
   8. The exam entry form must allow the user to select an exam entry from the exam entries table in order to allow for that exam entry’s data to be edited, updating the relevant fields within the form for editing.
   9. The exam entry form must allow the user to sort exam entries by any field in ascending or descending order.
   10. The exam entry form must allow for the user to search specific fields for specific exam entries.
   11. Exam entries must be payable through the use of the payment form system.
5. External Exam form
   1. The external exam form must allow for the entering of new external exam entries into the table as long as data entered is within acceptable parameters.
   2. The external exam form must be able to create new exam entries to add to the external exam table through pressing the create button.
   3. The external exam form must be able to remove exam entries from the table through pressing the delete button.
   4. The external exam form must generate a new external exam number for each new external exam that is created by the user.
   5. The external exam form must prevent the user from entering clearly incorrect information through the use of masked textboxes and combo boxes.
   6. The external exam form must ensure that all data entered is valid within the context of the field it is being entered for, this data must, if invalid, provide the user with feedback specifying what went wrong with the data they entered.
   7. The external exam form must allow the user to clearly see all exam entries currently stored within the SQL exam entries table.
   8. The external exam form must allow the user to select an external exam from the exam entries table in order to allow for that external exam’s data to be edited, updating the relevant fields within the form for editing.
   9. The external exam form must allow the user to sort exam entries by any field in ascending or descending order.
   10. The external exam form must allow for the user to search specific fields for specific exam entries.
6. Block booking form
   1. The block booking form must allow for the entering of new block booking entries into the table as long as data entered is within acceptable parameters.
   2. The block booking form must be able to create new block bookings to add to the block booking table through pressing the create button.
   3. The block booking form must be able to remove block bookings from the table through pressing the delete button.
   4. The block booking form must generate a new block booking number for each new block booking that is created by the user.
   5. The block booking form must prevent the user from entering clearly incorrect information through the use of masked textboxes and combo boxes.
   6. The block booking form must ensure that all data entered is valid within the context of the field it is being entered for, this data must, if invalid, provide the user with feedback specifying what went wrong with the data they entered.
   7. The block booking form must allow the user to clearly see all block bookings currently stored within the SQL block bookings table.
   8. The block booking form must allow the user to select an block booking from the block bookings table in order to allow for that block booking’s data to be edited, updating the relevant fields within the form for editing.
   9. The block booking form must allow the user to sort block bookings by any field in ascending or descending order.
   10. The block booking form must allow for the user to search specific fields for specific block bookings.
   11. Bookings must be able to be paid for through the use of the payment system.
7. Payment form
   1. The payment form must only allow payments when all specified fields have been filled.
   2. The payment form must record all successful payments to an SQL database table.
   3. The payment form must allow for the choice of payment method and record at least some of the necessary details regarding this within the data row saved to the payment table of the database.
   4. The payment form must return the user to the form through which they arrived here once either payment is completed or the user cancels the payment.
   5. The payment form must provide necessary information regarding the nature of the payment to the user so as they can validate that they are paying for the correct thing.
8. Payment History form
   1. The payment history form must display all payment records made within the current system.
   2. The payment history form must allow for the user to search specific fields for specific payments within the SQL database.
   3. The payment history form must allow the user to sort the payments by clicking the column head of the column they wish to sort.
9. Timetabled lessons form
   1. The timetabled lesson form must allow for the entering of new timetabled lessons into the table as long as data entered is within acceptable parameters.
   2. The timetabled lesson form must be able to create new timetabled lessons to add to the timetabled lesson table through pressing the create button.
   3. The timetabled lesson form must be able to remove timetabled lessons from the table through pressing the delete button.
   4. The timetabled lesson form must generate a new timetabled lesson number for each new timetabled lesson that is created by the user.
   5. The timetabled lesson form must prevent the user from entering clearly incorrect information through the use of masked textboxes and combo boxes.
   6. The timetabled lesson form must ensure that all data entered is valid within the context of the field it is being entered for, this data must, if invalid, provide the user with feedback specifying what went wrong with the data they entered.
   7. The timetabled lesson form must allow the user to clearly see all timetabled lessons currently stored within the SQL timetabled lessons table.
   8. The timetabled lesson form must allow the user to select a timetabled lesson from the timetabled lessons table in order to allow for that timetabled lesson’s data to be edited, updating the relevant fields within the form for editing.
   9. The timetabled lesson form must allow the user to sort timetabled lessons by any field in ascending or descending order.
   10. The timetabled lesson form must allow for the user to search specific fields for specific timetabled lessons.
   11. The timetabled lesson form must only allow for the entry of lessons that do not conflict with the requirements of other timetabled lessons.
   12. The timetabled lesson form must provide the user with easily accessible choices through the use of combo boxes to streamline the creation system and where possible limit the choices to those that would be acceptable given other variables regarding the choice of booking.
10. Lesson timetable form
    1. The lesson timetable form must supply the user with an understandable view of lessons that have been booked on various days.
    2. The lesson timetable form must allow the user to select the day for which they wish to search for lessons.
    3. The lesson timetable form must allow the user to hover their mouse over cells within the timetable to receive additional information about the specified lesson or empty cell such as the room, time of day and details regarding the lesson itself if there were one booked there.
    4. The lesson timetable form must allow the user to click on an empty cell to provide a quick method of selecting a day, time and room to book a lesson.
    5. The lesson timetable form must transfer the user with the specified details to the timetabled lessons form so as they can organize a lesson easily.
    6. The lesson timetable form must display cells as different colours to make the table easier to understand.
11. Room form
    1. The room form must allow for the entering of new rooms into the table as long as data entered is within acceptable parameters.
    2. The room form must be able to create new rooms to add to the room table through pressing the create button.
    3. The room form must be able to remove rooms from the table through pressing the delete button.
    4. The room form must generate a new room number for each new room that is created by the user.
    5. The room form must prevent the user from entering clearly incorrect information through the use of masked textboxes and combo boxes.
    6. The room form must ensure that all data entered is valid within the context of the field it is being entered for, this data must, if invalid, provide the user with feedback specifying what went wrong with the data they entered.
    7. The room form must allow the user to clearly see all rooms currently stored within the SQL rooms table.
    8. The room form must allow the user to select a room from the rooms table in order to allow for that room’s data to be edited, updating the relevant fields within the form for editing.
    9. The room form must allow the user to sort rooms by any field in ascending or descending order.
    10. The room form must allow for the user to search specific fields for specific rooms.
12. Tutor takes form
    1. The tutor takes form must be accessible through the tutor form by selecting a tutor and pressing proficiencies.
    2. The tutor takes form must display all proficiencies regarding the specified tutor as selected on the tutor form.
    3. The tutor takes form must allow for the entering of new proficiencies into the table as long as data entered is within acceptable parameters.
    4. The tutor takes form must allow for the user to search specific fields for specific proficiencies.
    5. The tutor takes form must allow the user to sort proficiencies by any field in ascending or descending order.
    6. The tutor takes form must allow the user to select a proficiency from the tutor takes table in order to allow for that proficiency’s data to be edited, updating the relevant fields within the form for editing.
    7. The tutor takes form must be able to remove proficiencies from the table through pressing the delete button.
    8. The tutor takes must generate a new tutor takes number for each new proficiency that is created by the user.
    9. The tutor takes must prevent the user from entering clearly incorrect information through the use of masked textboxes and combo boxes.
13. Student reports
    1. The student reports form must allow the user to view all data rows regarding the student table within the database.
    2. The student reports form must allow the user to query the report for each of the columns as part of that SQL table.
    3. The student reports form must allow the user to clear the query they had previously set.
14. Tutor reports
    1. The tutor reports form must allow the user to view all data rows regarding the tutor table within the database.
    2. The tutor reports form must allow the user to query the report for each of the columns as part of that SQL table.
    3. The tutor reports form must allow the user to clear the query they had previously set.
15. Payment reports
    1. The payment reports form must allow the user to view all data rows regarding the payment table within the database.
    2. The payment reports form must allow the user to query the report for each of the columns as part of that SQL table.
    3. The payment reports form must allow the user to clear the query they had previously set.
16. Block booking reports
    1. The block booking reports form must allow the user to view all data rows regarding the block booking table within the database.
    2. The block booking reports form must allow the user to query the report for each of the columns as part of that SQL table.
    3. The block booking reports form must allow the user to clear the query they had previously set.

## Non-Functional Requirements

1. General

1.1. An SQL database should be used to store the inputted data for future use and editing.

2. The form design

2.1. The design of the forms within the application should be easily understandable to the average user capable of working an application such as this.

2.2. The forms should be designed in such a way as to limit the user to prevent them from inputting erroneous data that would not be acceptable in those areas. This should be done to as much of an extent as possible as long as the usability of the application is not hindered.

2.3. The design of these forms should be laid out in such a way as for them to be presentable and understandable at a glance, usage of tools such as group boxes should be heavily used as such to enhance the users experience in this way.

## System Methodology

A system methodology is the process by which the task of creating the project is performed, each one structuring the process of the project creation in a different way. I intend to investigate these methodologies and decide for myself which would best be suited for the creation of the database creation project for Mitchell’s School of Music.

### Waterfall

Overview

The waterfall method is designed in such a way as to lead the development process through the process of the applications development one stage at a time, providing a simple yet effective rate of progress through the progress, starting from the start and ending at the end, rarely going over completed areas until it is time for testing or if there is a serious issue in the way it is designed such as a program-breaking issue or a clear issue with the way it is designed such as a design change requested by the user. This methodology is very linear as it progresses through the whole project, rarely looking back, hence the name “Waterfall” in reference to how it simply flows through the entire development process from start to finish.

The waterfall methodology is broken up into seven stages of development which the developer(s) pass through one by one before arriving at the finished product. These stages are: Analysis, Design, Software Development, Testing, Installation, Review and Maintenance. One of the primary issues with this methodology is also one of its most prominent features, that being the linear nature of the development, this linear view leads to some areas of the program to become less integrated into the final product, preventing the developer(s) from going back and integrating new features that may improve future usability, efficiency and ease when developing the rest of the application.

Another significant flaw to this methodology is the lack of flexibility as it does not encourage improving on finished stages of the project as mentioned although a further flaw to this is that it prevents the ability to receive input from the user as any comment would be on already completed areas of the project. As a result of this, the program will likely have been created with minimal to no guidance from the user.

However, these issues may be slightly subsided due to the requirement of each stage needing to be signed off by the project manager to confirm that the relevant stage has been completed to an acceptable standard, performing a sort of quality check on each stage. This is a subject of a greater problem however as it slows the progress of the project, preventing it from moving onto later stages until a standard is reached which may take an incalculable amount of time, making this perhaps among the most time consuming methodologies available for use.

Advantages and disadvantages

+ Consistent

+ Reliable (in most cases)

+ Quality assurance in the form of project manager signoffs on each stage.

- Very time consuming

- Very inflexible

- Accepts minimal user feedback during project creation.

### Extreme Programming

Overview

Extreme programming (or XP for short) is a methodology which relies heavily on its team members to hit their targets at the specified times, performing sprints similar to those is scrum only with slightly lighter targets to be achieved in 1-2 weeks rather than scrums 3-4. This allows the team to work on their specific area to completion, before getting set a new target to be completed and so on until the project is complete. This is one of the more agile methodologies as there are no specific rules as to what area of the project must be completed next, allowing the team members to focus their efforts wherever the targets are set to be and wherever they are needed.

XP has a significant focus on the applications of teamwork in a development environment, making use of paired programming which is a method in which two programmers are sat at the same piece of code and work through it together, solving problems and improving each other in the way in which they write it along the way, allowing one to point out mistakes and improvements while the other works. Although in a workplace, having two people work on one piece of code could be inefficient even if the quality of the code were improved as this would require having to have two people on the payroll for doing one job. This aspect of XP would not be relevant to me in my project however as I would be unable to obtain a consistent programmer to partner up with during the project.

Another large aspect of XP is the testing of the program during the creation of the program, allowing changes to be made on the fly, if it doesn’t work it can be changed at almost any time. Similarly, the user is interacted with as much as possible to ensure that the project is constantly meeting the expectations of the user and is working as fast and effectively as possible.

Advantages and disadvantages

+ Paired programming improves quality and stability of code.

+ Plenty of user feedback.

+ Fast development periods allow for quick builds and target setting.

- Potentially unstable as there are no clear boundaries between the areas of the development

- Two people hired to do one job.

- Paired programming may be inhibited by personality.

### Rapid Application Development

Overview

Rapid application development (or RAD for short) is a methodology which has a large and significant focus on the rapid prototyping of areas of the application, deciding to create many instances of it in quick succession, this could potentially result in some loss of quality however as rushing a product may result in rushing the code rather than patiently improving efficiency over time. One of the greatest advantages of this rapid production of prototypes however is the opportunity for the user to test out the newest prototypes to ensure that it is following the plan they had for it, providing a great opportunity for user feedback. Although, as one of the most significant focuses of this methodology is on speed, this comes at a few significant costs. Time limits set on each part of the overall system are very strict and must be met on time, thus potentially inflicting significant damage on the quality of the application. The methodology also works through allotting some requirements to essential and some to non-essential allowing for some features to be left out if time runs out. These time limits also require some improvements on the method of code design performed by the programmers as it emphasises the reuse of code as much as possible, providing a heavy focus on reusable classes and methods which can be used many times over many projects such as validation classes and classes which access specific data which is usually in the same place.

Advantages and disadvantages

+ Reuse of code.

+ Very good for user feedback.

+ Fast development time.

- Potential quality loss due to time constraints.

- Reuse of code may not be the most efficient method in some cases.

- Rushing code causes mistakes and potentially inefficient code.

### Scrum

Overview

The primary aim of the scrum methodology is to have the desired product completed to the desired standard as soon as possible. The ways in which they achieve this goal is through techniques such as sprints which last around 3-4 weeks, during these sprints teams of 7 plus or minus 2 people will work on an area of the project which must be completed by a specified time, this is knows as a scrum team, each scrum team would consist of its members plus a scrum master who essentially acts as the coach for the team, ensuring everyone remains on track and everything works the way it is supposed to, consistently checking the progress and status of the team so as to ensure it is completed on time. Each day the team will hold a scrum meeting, a fairly brief interaction in which they check up on progress and ensure that they are on track for the completion of their specified area.

The planning stage of a scrum development team sill usually be as short as possible as they will tend to work more under the principal that the projects issues can be corrected as it goes on, any issues with it being raised by the project owner who essentially represents the business in the scrum development team, updating them with what the business would like done so as their expectations can be reached accordingly.

Advantages and disadvantages

+ Will have someone present most of the time to represent the needs of the business.

+ Fast

+ Efficiency is maintained by a dedicated “coach” (the scrum master).

- Less communication between areas of the project.

- High pressure deadlines may cause mistakes.

- Minimal planning can cause mistakes and potentially require going over already completed tasks.

### Dynamic System Development Method

The Dynamic System Development Methodology (DSDM), has a focus on agility in the way the development of the project is approached. DSDM does this based on how important the independent features of the project are, allocating them under the MoSCoW prioritisation, assigning them the status of either must have, should have, could have and won’t have. This allows the developers of the project to focus resources on the areas which are the most in need of development and allowing for the vital features of the project to be created first and using time remaining to work on the less significant features. This feature is combined with Timeboxing to decide which areas of the project make it into the final iteration, removing features from the plan which are not vital if it is believed that they would not be able to complete them on time.

As DSDM is an agile methodology is shares similar features to those of RAD and EP, with a significant focus on creating quick working iterations of the project, used in conjunction with MoSCoW to create the iterations with the most important features first before moving onto the non-essentials. Testing is also performed after the creation of each of these iterations, so as it is clear on which areas of the project require the most future development.

Advantages and disadvantages

+ Helps ensure the most important features of the project are completed first.

+ Faster development time for necessary components.

+ Agile development allows for areas to be improved long after completion.

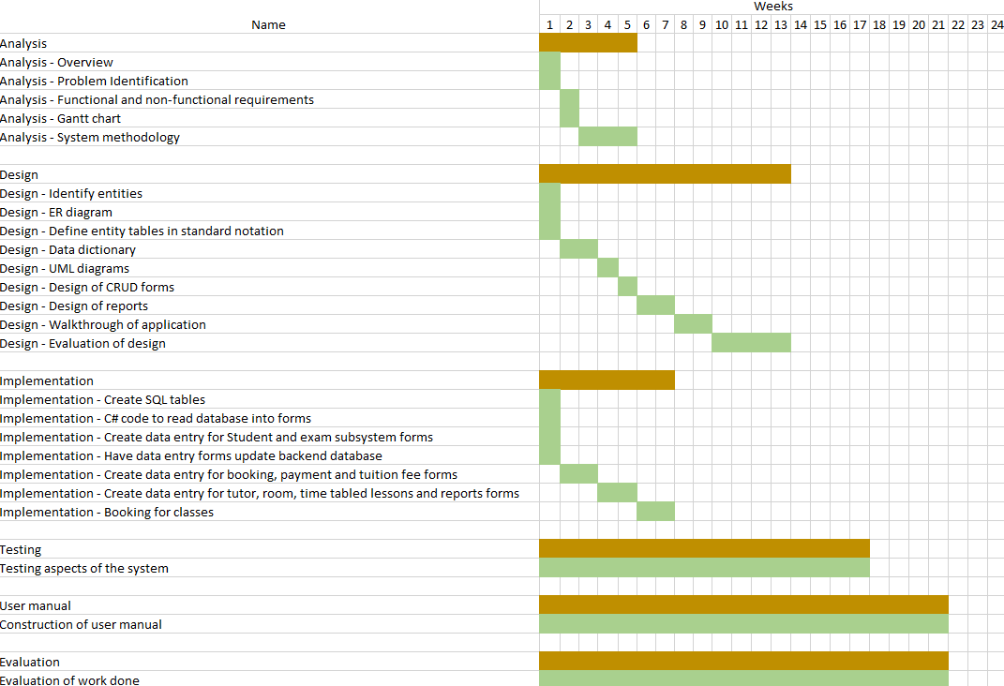
- Functionality takes significant importance while user experience is an afterthought.

### Conclusion

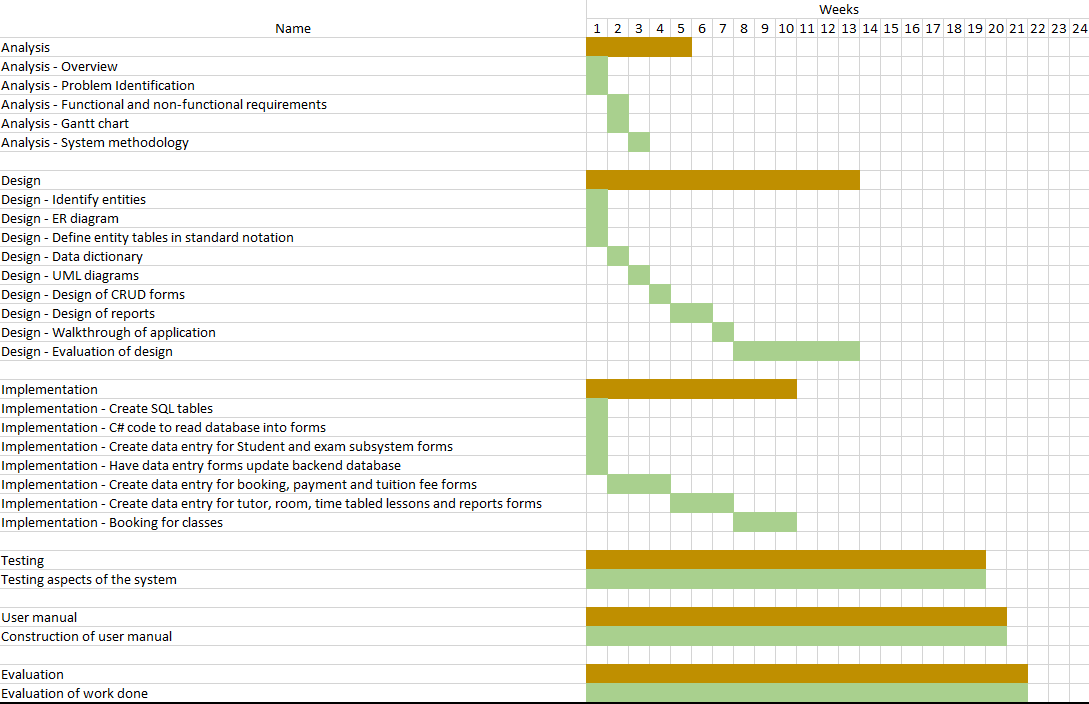
The methodology I will be following closest to would likely be primarily the waterfall method due to significant portions of the other methodologies features being locked out because of time constraints, resources and manpower as I am the sole worker on the project. A mixture of extreme programming for its speed and philosophy on how targets are set and are to be completed quickly, however, while this methodology most suits how I work on my project the other factors (such as paired programming and being able to get feedback from the business owner) would not be possible and Rapid application development for its reuse of code and methodology for creating many working prototypes of the project also play a part in the way in which my project is being developed as they both contribute well without being a detriment to the project as a whole. DSDM makes also has the best aspects of RAD integrated into it while also making use of the MoSCoW methodology of doing the most essential features first and foremost, all of which are important aspects which I have integrated into my development methodology.

## Gantt chart

### Initial plan



### Final timeline



### Analysis of Gantt charts

I kept to the plan for the most part, following the order I had laid out for myself, however as the project progressed I fell behind in a couple areas which then required to be caught up on, areas such as the testing which in turn required the user manual to be pushed back as a result of the late completion of the testing. This also meant I had less time to work on the evaluation but within the week I worked on it, I caught up at this point.

I was also able to keep ahead on my work in certain aspects such as the system methodologies which I completed with time to spare which then went into the development and implementation of the program itself.

# Design

## Identification of Entities

|  |  |  |
| --- | --- | --- |
| Entity | Description | Additional Information |
| Student | Information about the Students attending. |  |
| Instrument | Information on the availability and status of instruments. |  |
| ExternalExam | Information on the details of external exams. |  |
| InstrumentHire | Information on an instruments hiring. |  |
| ExamEntry | Details of a student’s entry into an exam. |  |
| Payment | Information on the student’s payment, a reference will be used to determine what the payment is for. |  |
| ExamResult | Information on a student’s performance in an exam. |  |
| BlockBooking | Information on a student’s booking for classes. |  |
| TuitionFee | Information on the amount a student may need to pay. | Lookup table. |
| TimeTabledLesson | Information regarding the date and time of lessons and where they take place. |  |
| Room | Information on a room within the building. |  |
| TuitionChoice | Information on the options available to students in regards to classes they could take. |  |
| Tutor | Information regarding tutors available for classes in the school. |  |
| TutorTakes | Information on the applicable skills of a tutor. |  |
| Report | Information regarding the classes of a student and their progress. |  |

## ER Diagram

Student

Instrument Hire

Payment

ExamEntry

ExternalExam

Instrument

Booking

TuitionFee

Tutor

TutorTakes

TuitionChoice

TimeTabledLessons

Report

## Class Diagrams

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | **Student** | | **-StudentNo int**  **-FirstName Varchar(30)**  **-Surname Varchar(30)**  **-Address Varchar(30)**  **-Town Varchar(30)**  **-PostCode Varchar(9)**  **-Email Varchar(50)**  **-DOB Date**  **-Gender Char(1)**  **-Disability bit**  **-ContactTelNo Varchar(13)**  **-ParentName Varchar(60)**  **-DateJoined Date**  **-DateLeft Date**  **-CurrentStudent bit** | | **+Create()**  **+Delete()**  **+Update()**  **+Search()**  **+Sort()** | | |  | | --- | | **Tutor** | | **-TutorNo int**  **-FirstName Varchar(30)**  **-Surname Varchar(30)**  **-Address Varchar(30)**  **-Town Varchar(30)**  **-PostCode Varchar(9)**  **-Email Varchar(50)**  **-DOB Date**  **-Gender Char(1)**  **-Disability bit**  **-ContactTelNo Varchar(13)**  **-NextOfKinName Varchar(60)**  **-DateJoined Date**  **-DateLeft Date**  **-CurrentTutor bit** | | **+Create()**  **+Delete()**  **+Update()**  **+Search()**  **+Sort()** | | |  | | --- | | **Room** | | **-RoomNo int**  **-RoomType varchar(30)**  **-Capacity int**  **-InUse bit** | | **+Create()**  **+Delete()**  **+Update()**  **+Search()**  **+Sort()** | |
| |  | | --- | | **ExternalExam** | | **-ExamNo int**  **-ExamBoard varchar(30)**  **-TitleExam varchar(30)**  **-Grade char(1)**  **-DateExam Date**  **-TimeExam Time**  **-ExamFee Decimal(7,2)**  **-AcceptingEntries bit** | | **+Create()**  **+Delete()**  **+Update()**  **+Search()**  **+Sort()** | | |  | | --- | | **TuitionChoice** | | **-TuitionChoice varchar(50)**  **-CurrentlyOffered bit** | | **+Create()**  **+Delete()**  **+Update()**  **+Search()**  **+Sort()** | | |  | | --- | | **TuitionFee** | | **-FeeCode varchar(9)**  **-LessonRate decimal(7,2)** | | **+Create()**  **+Delete()**  **+Update()**  **+Search()**  **+Sort()** | |
| |  | | --- | | **ExamEntry** | | **-ExamEntryNo int**  **-ExamNo int**  **-StudentNo int**  **-DateEntry date**  **-Fee decimal(7,2)**  **-Paid bit** | | **+Create()**  **+Delete()**  **+Update()**  **+Search()**  **+Sort()** | | |  | | --- | | **ExamResult** | | **-ExamResultNo int**  **-ExamNo int**  **-StudentNo int**  **-Result char(1)**  **-DateCertificateRecieved date**  **-DateCertificateIssued date**  **-CollectedBy varchar(60)**  **-Collected bit** | | **+Create()**  **+Delete()**  **+Update()**  **+Search()**  **+Sort()** | | |  | | --- | | **Payment** | | **-PaymentNo int**  **-RefNo int**  **-StudentNo int**  **-Description varchar(50)**  **-DatePaid date**  **-AmountPaid decimal(7,2)**  **-PaymentMethod varchar(50)**  **-ResponsibleEmployee varchar(60)**  **-Sponsor varchar(60)**  **-SponsorOwner varchar(60)**  **-Paid bit** | | **+Create()**  **+Delete()**  **+Update()**  **+Search()**  **+Sort()** | |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | **BlockBooking** | | **-BookingNo int**  **-StudentNo int**  **-TuitionChoice varchar(50)**  **-DateBooking date**  **-AbilityLevel varchar(1)**  **-NoLessons int**  **-DiscountRate int**  **-LessonRate decimal(7,2)**  **-TotalDue decimal(7,2)**  **-Paid bit**  **-WaitingList bit** | | **+Create()**  **+Delete()**  **+Update()**  **+Search()**  **+Sort()** | | |  | | --- | | **Report** | | **-ReportNo int**  **-StudentNo int**  **-TutorNo int**  **-AcadYear varchar(10)**  **-Term int**  **-ReportComments varchar(1000)** | | **+Create()**  **+Delete()**  **+Update()**  **+Search()**  **+Sort()** | | |  | | --- | | **TimeTabledLesson** | | **-TLNo int**  **-TutorNo int**  **-StudentNo int**  **-RoomNo int**  **-BookingNo int**  **-DateLesson date**  **-TimeLesson smallint**  **-StudentPresent bit**  **-Cancellation bit**  **-CancellationReason varchar(1000)** | | **+Create()**  **+Delete()**  **+Update()**  **+Search()**  **+Sort()** | |
| |  | | --- | | **TutorTakes** | | **-TutorTakesNo int**  **-TutorNo int**  **-TuitionChoice varchar(50)**  **-TeachUpToGrade varchar(1)** | | **+Create()**  **+Delete()**  **+Update()**  **+Search()**  **+Sort()** | |  |  |

## Entity Standard Notation

### 1NF

#### Exam Entry

ExamEntry(**ExamEntryNo**, *ExamNo*, StudentNo, FirstName, Surname, Address, Town, PostCode, Email, DOB, Gender, Disability, ContactTelNo, ParentName, AbilityLevel, DateJoined, CurrentStudent, DateLeft, ExamResult, DateCertificateIssued, DateCertificateRecieved, CollectedBy, Collected)

EXTERNALEXAM(**ExamNo**, ExamBoard, TitleExam, Grade, DateExam, TimeExam, ExamFee)

#### Booking

#### Payment

### 2NF

#### Exam Entry

ExamEntry(**ExamEntryNo**, *ExamNo*, StudentNo, FirstName, Surname, Address, Town, PostCode, Email, DOB, Gender, Disability, ContactTelNo, ParentName, AbilityLevel, DateJoined, CurrentStudent, DateLeft)

EXTERNALEXAM(**ExamNo**, ExamBoard, TitleExam, Grade, DateExam, TimeExam, ExamFee)

EXAMRESULT(**ExamResultNo**, *ExamEntryNo,* Result, DateCertificateRecieved, DateCertificateIssued, CollectedBy, Collected)

#### Booking

#### Payment

### 3NF

#### Exam Entry

ExamEntry(**ExamEntryNo**, *ExamNo*, *StudentNo*, StudentName, Address, TownPostCode)

EXTERNALEXAM(**ExamNo**, ExamBoard, TitleExam, Grade, DateExam, TimeExam, ExamFee)

EXAMRESULT(**ExamResultNo**, *ExamNo, StudentNo,* Result, DateCertificateRecieved, DateCertificateIssued, CollectedBy, Collected)

STUDENT(**StudentNo**, FirstName, Surname, Address, Town, PostCode, Email, DOB, Gender, Disability, ContactTelNo, ParentName, AbilityLevel, DateJoined, CurrentStudent, DateLeft)

#### Booking

BLOCKBOOKING(**BookingNo**, *StudentNo*, DateBooking, TuitionChoice, AbilityLevel, NoLessons, DiscountRate, HourlyRate, Paid, WaitingList)

STUDENT(**StudentNo**, FirstName, Surname, Address, Town, PostCode, Email, DOB, Gender, Disability, ContactTelNo, ParentName, AbilityLevel, DateJoined, CurrentStudent, DateLeft)

TUITIONCHOICE(**TuitionChoice**, CurrentlyOffered)

TIMETABLEDLESSON(**TLNo**, *StudentNo*, *TutorNo*, *RoomNo*, *BookingNo*, DateLesson, TimeLessons, StudentPresent, Cancellation, CancellationReason)

#### Payment

PAYMENT(**PaymentNo**, *RefNo*, Description, PaymentMethod, Fee, ResponsibleEmployee, Sponsor, SponsorOwner, Paid)

TUTOR(**TutorNo**, FirstName, Surname, Address, Town, PostCode, Email, DOB, Gender, Disability, ContactTelNo, NextOfKin, TutorLevel, DateJoined, CurrentTutor, DateLeft)

TUTORTAKES(**TutorTakesNo**, *TutorNo*, TuitionChoice, TeachUpToGrade)

### All Standard Notations 3NF

STUDENT(**StudentNo**, FirstName, Surname, Address, Town, PostCode, Email, DOB, Gender, Disability, ContactTelNo, ParentName, AbilityLevel, DateJoined, CurrentStudent, DateLeft)

INSTRUMENT(**InstrumentNo**, Description, MonthlyHireFee, Condition, Hired)

INSTRUMENTHIRE(**InstrumentHireNo**, *InstrumentNo*, *StudentNo*, DateHire, DateReturn, NoMonthsHire, HireCost, ExcessCharges)

EXTERNALEXAM(**ExamNo**, ExamBoard, TitleExam, Grade, DateExam, TimeExam, ExamFee)

EXAMENTRY(**ExamEntryNo**, *ExamNo*, *StudentNo*, DateEntry, Fee, Paid, DatePaid)

EXAMRESULT(**ExamResultNo**, *ExamNo, StudentNo*, Result, DateCertificateRecieved, DateCertificateIssued, CollectedBy, Collected)

BLOCKBOOKING(**BookingNo**, *StudentNo*, DateBooking, TuitionChoice, AbilityLevel, NoLessons, DiscountRate, HourlyRate, Paid, WaitingList)

TIMETABLEDLESSON(**TLNo**, *StudentNo*, *TutorNo*, *RoomNo*, *BookingNo*, DateLesson, TimeLessons, StudentPresent, Cancellation, CancellationReason)

ROOM(**RoomNo**, RoomType, Capacity, InUse)

TUITIONCHOICE(**TuitionChoice**, CurrentlyOffered)

TUTOR(**TutorNo**, FirstName, Surname, Address, Town, PostCode, Email, DOB, Gender, Disability, ContactTelNo, NextOfKin, TutorLevel, DateJoined, CurrentTutor, DateLeft)

TUTORTAKES(**TutorTakesNo**, *TutorNo*, TuitionChoice, TeachUpToGrade)

PAYMENT(**PaymentNo**, *RefNo*, Description, PaymentMethod, Fee, ResponsibleEmployee, Sponsor, SponsorOwner, Paid)

## Data Dictionary

### Student

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Attribute name | Data type | Length / size (bytes) | Default value | Validation | Description / comments |
| StudentNo | int | 4 | Auto-generated starting at 1 | Auto-generated starting at 1 | Primary key value. +1 for each new value. |
| FirstName | Varchar/string | 30 | Null | Valid varchar/string. | Not null |
| Surname | Varchar/string | 30 | Null | Valid varchar/string. | Not null |
| Address | Varchar/string | 30 | Null | Valid varchar/string.  6-30 chars | Not null |
| Town | Varchar/string | 30 | Null | Valid varchar/string. | Not null |
| PostCode | Varchar/string | 9 | Null | Valid varchar/string.  6-9 chars | Not null |
| Email | Varchar/string | 50 | Null | Valid varchar/string.  5-50 chars | Not null |
| DOB | Date | 3 | Null |  | Not null |
| Gender | Bit/bool | Bit | 0 |  | Not null |
| Disability | Bit/bool | Bit | 0 |  | Not null |
| ContactTelNo | Varchar/string | 13 | Null | Valid varchar/string. | Not null |
| ParentName | Varchar/string | 60 | Null | Valid varchar/string. | Not null |
| DateJoined | Date | 3 | Null |  | Not null |
| DateLeft | Date | 3 | Null |  |  |
| CurrentStudent | Bit/bool | Bit | 0 |  | Not null |

### Tutor

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Attribute name | Data type | Length / size (bytes) | Default value | Validation | Description / comments |
| TutorNo | int | 4 | Auto-generated starting at 1 | Auto-generated starting at 1 | Primary key value. +1 for each new value. |
| FirstName | Varchar/string | 30 | Null | Valid varchar/string. | Not null |
| Surname | Varchar/string | 30 | Null | Valid varchar/string. | Not null |
| Address | Varchar/string | 30 | Null | Valid varchar/string.  6-30 chars | Not null |
| Town | Varchar/string | 30 | Null | Valid varchar/string. | Not null |
| PostCode | Varchar/string | 9 | Null | Valid varchar/string.  6-9 chars | Not null |
| Email | Varchar/string | 50 | Null | Valid varchar/string.  5-50 chars | Not null |
| DOB | Date | 3 | Null | Valid Date. | Not null |
| Gender | Bit/bool | Bit | Null |  | Not null |
| Disability | Bit/bool | Bit | Null |  | Not null |
| ContactTelNo | Varchar/string | 13 | Null | Valid varchar/string. | Not null |
| NextOfKinName | Varchar/string | 60 | Null | Valid varchar/string. | Not null |
| DateJoined | Date | 3 | Null | Valid Date. | Not null |
| DateLeft | Date | 3 | Null | Valid Date. |  |
| CurrentTutor | Bit/bool | Bit | Null |  | Not null |

### ExamEntry

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Attribute name | Data type | Length / size (bytes) | Default value | Validation | Description / comments |
| ExamEntryNo | int | 4 | Auto-generated starting at 1 | Auto-generated starting at 1 | Primary key value. +1 for each new value. |
| ExamNo | Int | 4 | Null | Valid int.  Valid reference to specified table. | Not null  Foreign key reference to the primary key of the ExternalExam table. |
| StudentNo | Int | 4 | Null | Valid int.  Valid reference to specified table. | Foreign key reference to the primary key of the Student table. |
| DateEntry | Date | 3 | Null | Valid date | Not null |
| Fee | Decimal | 5 | Null | 7 digits max including decimal places. | Not null |
| Paid | Bit/Bool | bit | Null |  | Not null |
| DatePaid | date | 3 | Null |  |  |

### ExternalExam

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Attribute name | Data type | Length / size (bytes) | Default value | Validation | Description / comments |
| ExternalExamNo | int | 4 | Auto-generated starting at 1 | Auto-generated starting at 1 | Primary key value. +1 for each new value. |
| ExamBoard | Varchar/string | 30 | Null | 1-30 chars | Not null |
| ExamTitle | Varchar/string | 30 | Null | 1-30 chars | Not null |
| Grade | char | 1 | Null | 1 char | Not null |
| ExamDate | Date | 3 | Null | Valid date | Not null |
| ExamTime | Time | 3 | Null | Valid time | Not null |
| ExamFee | Decimal/float | 5 | Null | 7 digits max including decimal places. | Not null |
| AcceptingEntries | Bit/Bool | bit | 1/true |  | Not null |

### BlockBooking

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Attribute name | Data type | Length / size (bytes) | Default value | Validation | Description / comments |
| BookingNo | int | 4 | Auto-generated starting at 1 | Auto-generated starting at 1 | Primary key value. +1 for each new value. |
| StudentNo | int | 4 | Null | Valid int.  Valid reference to specified table. | Not null.  Foreign key reference to the student tables StudentNo. |
| DateBooking | Date |  | Null | Valid date | Not null |
| TuitionChoice | Varchar/string | 50 | Null | Valid string.  Valid reference to specified table. | Not null.  Foreign key reference to the TuitionChoice tables TuitionChoice. |
| Ability level | Varchar | 1 | Null | 1 char | Not null |
| NoLessons | Int | 4 | Null | Valid int | Not null |
| DiscountRate | Int | 4 | Null | Valid int | Not null |
| LessonRate | decimal | 5 | Null | 7 digits max including decimal places. | Not null |
| TotalDue | decimal | 5 | Null | 7 digits max including decimal places. | Not null  A calculation of the lesson rate and the number of lessons. |
| Paid | Bit/bool | Bit | Null |  | Not null |
| WaitingList | Bit/bool | Bit | Null |  | Not null |

### Room

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Attribute name | Data type | Length / size (bytes) | Default value | Validation | Description / comments |
| RoomNo | Int | 4 | Auto-generated starting at 1 | Auto-generated starting at 1 | Primary key value. +1 for each new value. |
| RoomType | Varchar | 30 | Null | 30 chars | Not null |
| Capacity | Int |  | Null | Valid int | Not null |
| InUse | Bit/bool | Bit | Null |  | Not null |

### TuitionChoice

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Attribute name | Data type | Length / size (bytes) | Default value | Validation | Description / comments |
| TuitionChoice | Varchar | 50 | Null | Valid varchar/string | Primary key value. |
| CurrentlyOffered | Bit/Bool | Bit | Null |  | Not null |

### TuitionFee

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Attribute name | Data type | Length / size (bytes) | Default value | Validation | Description / comments |
| FeeCode | Varchar | 9 | Null | Valid varchar/string | Primary key value. |
| LessonRate | Decimal | 5 | Null | 2 decimal places.  Valid Decimal. | Not null |

### ExamResult

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Attribute name | Data type | Length / size (bytes) | Default value | Validation | Description / comments |
| ExamResultNo | Int | 4 | Auto-generated starting at 1. | Auto-generated starting at 1 | Primary key value. +1 for each new value. |
| ExamNo | int | 4 | Null | Valid int.  Valid reference to specified table. | Not null.  Foreign key reference to the ExternalExam tables ExamNo. |
| StudentNo | int | 4 | Null | Valid int.  Valid reference to specified table. | Not null.  Foreign key reference to the student tables StudentNo. |
| Result | varchar | 1 | Null | Valid Varchar/string | Not null.  Should only be either ▬ |
| DateCertificateRecieved | Date | 3 | Null | Valid Date. | Not null. |
| DateCertificateIssued | Date | 3 | Null | Valid Date. | Not null. |
| CollectedBy | Varchar | 60 | Null | Valid Varchar/string | Not null. |
| Collected | Bit/Bool | Bit | Null |  | Not null. |

### Payment

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Attribute name | Data type | Length / size (bytes) | Default value | Validation | Description / comments |
| PaymentNo | Int | 4 | Auto-generated starting at 1. | Auto-generated starting at 1 | Primary key value. +1 for each new value. |
| RefNo | Int | 4 | Null | Valid int.  Valid reference to specified table. | Not null.  Foreign key reference to the appropriate table to which the payment is due, decided based on the description. |
| StudentNo | int | 4 | Null | Valid int.  Valid reference to specified table. | Not null.  Foreign key reference to the student tables StudentNo. |
| Description | Varchar | 50 | Null | Valid Varchar/string | Not null.  Should only be a reference to the form from which the payment originated such as the booking from or exam entry form for example. |
| DatePaid | Date | 3 | Null | Valid Date. | Not null. |
| AmountPaid | Decimal | 5 | Null | Valid Date. | Not null.  2 decimal places. |
| PaymentMethod | Varchar | 50 | Null | Valid varchar/string. | Not null. |
| ResponsibleEmployee | Varchar | 60 | Null | Not white space. | Not null.  The employee responsible for overseeing the payment. |
| Sponsor | Varchar | 60 | Null | Not white space. | The name of the company that is sponsoring the student. |
| SponsorOwner | Varchar | 60 | Null | Not white space. | The name of the person issuing the sponsor. |
| Paid | Bit/Bool | Bit | Null |  | Not null. |

### BlockBooking

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Attribute name | Data type | Length / size (bytes) | Default value | Validation | Description / comments |
| BookingNo | Int | 4 | Auto-generated starting at 1. | Auto-generated starting at 1 | Primary key value. +1 for each new value. |
| StudentNo | Int | 4 | Null | Valid int.  Valid reference to specified table. | Not null.  Foreign key reference to the student tables StudentNo. |
| TuitionChoice | Varchar | 50 | Null | Valid string.  Valid reference to specified table. | Not null.  Foreign key reference to the TuitionChoice tables TuitionChoice. |
| DateBooking | Date | 3 | Null | Valid date. | Not null. |
| AbilityLevel | Varchar | 1 | Null | Valid Varchar/string | Not null. |
| NoLessons | Int | 4 | Null | Valid int. | Not Null. |
| DiscountRate | Int | 4 | Null | Valid int. | Not Null. |
| LessonRate | Decimal | 5 | Null | Valid Decimal.  2 decimal places. | Not Null. |
| TotalDue | Decimal | 5 | Null | Valid Decimal.  2 decimal places. | Not Null. |
| Paid | Bit/Bool | Bit | Null |  | Not Null. |
| WaitingList | Bit/Bool | Bit | Null |  | Not Null. |

### Report

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Attribute name | Data type | Length / size (bytes) | Default value | Validation | Description / comments |
| ReportNo | Int | 4 | Auto-generated starting at 1. | Auto-generated starting at 1 | Primary key value. +1 for each new value. |
| StudentNo | Int | 4 | Null | Valid int.  Valid reference to specified table. | Not null.  Foreign key reference to the student tables StudentNo. |
| TutorNo | Int | 4 | Null | Valid int.  Valid reference to specified table. | Not null.  Foreign key reference to the tutor tables TutorNo. |
| AcadYear | Varchar | 10 | Null | Valid varchar/string. | Not null. |
| Term | Int | 4 | Null | Valid int. | Not null. |
| ReportComments | Varchar | 1000 | Null | Valid varchar/string. | Not Null. |

### TimeTabledLesson

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Attribute name | Data type | Length / size (bytes) | Default value | Validation | Description / comments |
| TLNo | Int | 4 | Auto-generated starting at 1. | Auto-generated starting at 1 | Primary key value. +1 for each new value. |
| TutorNo | Int | 4 | Null | Valid int.  Valid reference to specified table. | Not null.  Foreign key reference to the tutor tables TutorNo. |
| StudentNo | Int | 4 | Null | Valid int.  Valid reference to specified table. | Not null.  Foreign key reference to the student tables StudentNo. |
| RoomNo | Int | 4 | Null | Valid int.  Valid reference to specified table. | Not null.  Foreign key reference to the Room tables RoomNo. |
| BookingNo | Int | 4 | Null | Valid int.  Valid reference to specified table. | Not null.  Foreign key reference to the Booking tables BookingNo. |
| DateLesson | Date | 3 | Null | Valid Date. | Not Null. |
| TimeLesson | Smallint | 2 | Null | Valid Int. | Not Null. |
| StudentPresent | Bit/bool | Bit | Null |  | Not Null. |
| Cancellation | Bit/bool | Bit | Null |  | Not Null. |
| CancellationReason | Varchar | 1000 | Null | Valid varchar/string. | Not Null. |

### TutorTakes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Attribute name | Data type | Length / size (bytes) | Default value | Validation | Description / comments |
| TutorTakesNo | Int | 4 | Auto-generated starting at 1. | Auto-generated starting at 1 | Primary key value. +1 for each new value. |
| TutorNo | Int | 4 | Null | Valid int.  Valid reference to specified table. | Not null.  Foreign key reference to the tutor tables TutorNo. |
| TuitionChoice | Varchar | 50 | Null | Valid string.  Valid reference to specified table. | Not null.  Foreign key reference to the TuitionChoice tables TuitionChoice. |
| TeachUpToGrade | Varchar | 1 | Null | Valid varchar/string. | Not null. |

## Storyboards

Note

* Dotted/dashed boxes imply programmatically generated objects

### Student

1

2

3

6

5

4

7

13

12

9

8

11

10

14

40

39

25

38

24

23

22

21

20

19

18

17

16

15

35

34

33

32

31

30

29

28

27

37

36

26

|  |  |  |
| --- | --- | --- |
| Number | Name | Object type |
| 1 | frmStudent | Windows Form |
| 2 | dgvStudent | Data Grid View |
| 3 | gbxTools | Group Box |
| 4 | btnCreate | Button |
| 5 | btnUpdate | Button |
| 6 | btnDelete | Button |
| 7 | gbxSearch | Group Box |
| 8 | lblSearch | Label |
| 9 | lblIn | Label |
| 10 | txtSearch | Text Box |
| 11 | cboCollumnTitles | Combo Box |
| 12 | btnReset | Button |
| 13 | btnSearch | Button |
| 14 | gbxProperties | Group Box |
| 15 | lblFirstName | Label |
| 16 | lblSurname | Label |
| 17 | lblAddress | Label |
| 18 | lblTown | Label |
| 19 | lblPostCode | Label |
| 20 | lblEmail | Label |
| 21 | lblDOB | Label |
| 22 | lblTelNo | Label |
| 23 | lblGender | Label |
| 24 | lblDateJoined | Label |
| 25 | lblDateLeft | Label |
| 26 | lblParentName | Label |
| 27 | txtFirstName | Text Box |
| 28 | txtSurname | Text Box |
| 29 | txtAddress | Text Box |
| 30 | txtTown | Text Box |
| 31 | mtbPostCode | Masked Text Box |
| 32 | txtEmail | Text Box |
| 33 | mtbDOB | Masked Text Box |
| 34 | mtbTelNo | Masked Text Box |
| 35 | cboGender | Combo Box |
| 36 | cbxDisability | Check Box |
| 37 | cbxCurrentStudent | Check Box |
| 38 | mtbDateJoined | Masked Text Box |
| 39 | mtbDateLeft | Masked Text Box |
| 40 | txtParentName | Text Box |

### Tutor

1

2

3

6

5

4

7

13

12

9

8

11

10

14

40

39

25

38

24

23

22

21

20

19

18

17

16

15

35

34

33

32

31

30

29

28

27

37

36

26

|  |  |  |  |
| --- | --- | --- | --- |
| Number | Name |  | Object type |
| 1 | frmTutor |  | Windows Form |
| 2 | dgvTutor |  | Data Grid View |
| 3 | gbxTools |  | Group Box |
| 4 | btnCreate |  | Button |
| 5 | btnUpdate |  | Button |
| 6 | btnDelete |  | Button |
| 7 | gbxSearch |  | Group Box |
| 8 | lblSearch |  | Label |
| 9 | lblIn |  | Label |
| 10 | txtSearch |  | Text Box |
| 11 | cboCollumnTitles |  | Combo Box |
| 12 | btnReset |  | Button |
| 13 | btnSearch |  | Button |
| 14 | gbxProperties |  | Group Box |
| 15 | lblFirstName |  | Label |
| 16 | lblSurname |  | Label |
| 17 | lblAddress |  | Label |
| 18 | lblTown |  | Label |
| 19 | lblPostCode |  | Label |
| 20 | lblEmail |  | Label |
| 21 | lblDOB |  | Label |
| 22 | lblTelNo |  | Label |
| 23 | lblGender |  | Label |
| 24 | lblDateJoined |  | Label |
| 25 | lblDateLeft |  | Label |
| 26 | lblNextOfKinName |  | Label |
| 27 | txtFirstName |  | Text Box |
| 28 | txtSurname |  | Text Box |
| 29 | txtAddress |  | Text Box |
| 30 | txtTown |  | Text Box |
| 31 | mtbPostCode |  | Masked Text Box |
| 32 | txtEmail |  | Text Box |
| 33 | mtbDOB |  | Masked Text Box |
| 34 | mtbTelNo |  | Masked Text Box |
| 35 | cboGender |  | Combo Box |
| 36 | cbxDisability |  | Check Box |
| 37 | cbxCurrentTutor |  | Check Box |
| 38 | mtbDateJoined |  | Masked Text Box |
| 39 | mtbDateLeft |  | Masked Text Box |
| 40 | txtNextOfKinName |  | Text Box |

### Payment History

4

8

7

6

5

10

9

1

2

3

|  |  |  |
| --- | --- | --- |
| Number | Name | Object type |
| 1 | frmPaymentHistory | Windows Form |
| 2 | dgvPayment | Data Grid View |
| 3 | btnClose | Button |
| 4 | gbxSearch | Group Box |
| 5 | lblSearch | Label |
| 6 | lblIn | Label |
| 7 | btnReset | Button |
| 8 | btnSearch | Button |
| 9 | txtSearch | Text Box |
| 10 | cboCollumnTitles | Combo Box |

### Lesson Timetable

1

2

3

4

5

|  |  |  |
| --- | --- | --- |
| Number | Name | Object type |
| 1 | frmLessonTimetable | Windows Form |
| 2 | dtpSearch | Date Time Picker |
| 3 | lblX[] | Label [] |
| 4 | lblY[] | Label [] |
| 5 | pbxCell[,] | Picture Box [,] |

### Room

1

2

3

6

5

4

7

13

12

9

8

11

10

16

15

18

17

19

14

|  |  |  |
| --- | --- | --- |
| Number | Name | Object type |
| 1 | frmRoom | Windows Form |
| 2 | dgvRoom | Data Grid View |
| 3 | gbxTools | Group Box |
| 4 | btnCreate | Button |
| 5 | btnUpdate | Button |
| 6 | btnDelete | Button |
| 7 | gbxSearch | Group Box |
| 8 | lblSearch | Label |
| 9 | lblIn | Label |
| 10 | txtSearch | Text Box |
| 11 | cboCollumnTitles | Combo Box |
| 12 | btnReset | Button |
| 13 | btnSearch | Button |
| 14 | gbxProperties | Group Box |
| 15 | lblRoomType | Label |
| 16 | lblCapacity | Label |
| 17 | txtRoomType | Text Box |
| 18 | nudCapacity | Numeric Up-Down |
| 19 | cbxInUse | Check Box |

### Exam Entry

1

2

3

6

5

4

14

13

12

9

8

11

10

28

27

23

25

22

15

17

26

7

16

18

24

21

19

20

|  |  |  |
| --- | --- | --- |
| Number | Name | Object type |
| 1 | frmExamEntry | Windows Form |
| 2 | dgvExamEntry | Data Grid View |
| 3 | gbxTools | Group Box |
| 4 | btnCreate | Button |
| 5 | btnUpdate | Button |
| 6 | btnDelete | Button |
| 7 | btnPay | Button |
| 8 | lblSearch | Label |
| 9 | lblIn | Label |
| 10 | txtSearch | Text Box |
| 11 | cboCollumnTitles | Combo Box |
| 12 | btnReset | Button |
| 13 | btnSearch | Button |
| 14 | gbxSearch | Group Box |
| 15 | lblExamNo | Label |
| 16 | lblStudentNo | Label |
| 17 | cboExamNo | Combo Box |
| 18 | cboStudentNo | Combo Box |
| 19 | txtExamDetails | Text Box |
| 20 | txtStudentDetails | Text Box |
| 21 | lblEntryDate | Label |
| 22 | lblFee | Label |
| 23 | lblDatePaid | Label |
| 24 | mtbDateEntry | Masked Text Box |
| 25 | mtbFee | Masked Text Box |
| 26 | cbxPaid | Check Box |
| 27 | mtbDatePaid | Masked Text Box |
| 28 | gbxProperties | Group Box |

### External Exam

1

2

3

6

5

4

7

13

12

9

8

11

10

14

17

16

15

23

22

21

27

18

24

19

25

20

26

|  |  |  |
| --- | --- | --- |
| 1 | frmExternalExam | Windows Form |
| 2 | dgvExternalExam | Data Grid View |
| 3 | gbxTools | Group Box |
| 4 | btnCreate | Button |
| 5 | btnUpdate | Button |
| 6 | btnDelete | Button |
| 7 | gbxSearch | Group Box |
| 8 | lblSearch | Label |
| 9 | lblIn | Label |
| 10 | txtSearch | Text Box |
| 11 | cboCollumnTitles | Combo Box |
| 12 | btnReset | Button |
| 13 | btnSearch | Button |
| 14 | gbxProperties | Group Box |
| 15 | lblExamBoard | Label |
| 16 | lblExamTitle | Label |
| 17 | lblGrade | Label |
| 18 | lblExamDate | Label |
| 19 | lblExamTime | Label |
| 20 | lblExamFee | Label |
| 21 | txtExamBoard | Text Box |
| 22 | txtExamTitle | Text Box |
| 23 | cboGrade | Combo Box |
| 24 | mtbExamDate | Masked Text Box |
| 25 | mtbExamTime | Masked Text Box |
| 26 | mtbExamFee | Masked Text Box |
| 27 | cbxAcceptingEntry | Check Box |

## Walkthrough and overview of application

### Menu

Once the application is started, the first thing that should happen is the main menu should appear, presenting a menu strip alongside a visually appealing logo for the Mitchells School of Music business. Clicking on one of the areas within the menu strip triggers a drop down selection, allowing the choice of forms available within the selected category. For example, the reports drop down will have all reports, the bookings will have everything regarding the arranging and timetabling of lessons and the exams drop-down will have everything in regards to the exams available to students.

Clicking on any of these selections within the drop-downs will trigger the transfer to its respective form, with the exception of the exit button which will close the application in its entirety. Clicking either the red x at the top right of the form, or the exit button under the menu drop-down will close the entire application, including any other forms that may also be open at the time.

### Modify Students form

#### Adding students

1. Fill in all necessary fields.
2. Once verified and valid, the entered data will be entered into a class, after which, all information will be ported into a data row of its respective database and finally added to the database, updating the database itself with the new details. If the data entered is invalid, an error provider will appear next to the incorrect fields, instructing the user as to where they went wrong.

#### Deleting students

1. Select a data row to delete in the data grid view.
2. Press the delete button in the tools group box.
3. A prompt will appear, asking if you are sure you wish to delete the selected row.
4. Pressing yes will complete deletion of the selected row.
5. Pressing no will cancel the deletion process.

#### Updating students

1. Select a data row to update from the data grid view.
2. Selecting a data row will auto-fill all fields within the properties section of the form with details from the data row.
3. Modifications can be made to this data as needed.
4. Pressing the update data button in the tools group box will update the selected row with the data entered once it has confirmed it to be valid and added to a class.
5. If invalid, the error provider will appear next to the invalid fields and inform the user where they went wrong.

#### Sorting students

1. Clicking the column by which to sort within the data grid view will sort the database by that field.

#### Searching students

1. The user can type the word, number, date or true/false value for which they wish to search into the search text box.
2. The user should then select the column by which they wish to search using the combo box below the search field which contains each of the columns available to be searched.
3. The user should then press the search button to commence search.
4. The data grid view will then be filtered to match the search criteria.

### Modify Tutor form

#### Adding tutors

1. Fill in all necessary fields.
2. Once verified and valid, the entered data will be entered into a class, after which, all information will be ported into a data row of its respective database and finally added to the database, updating the database itself with the new details. If the data entered is invalid, an error provider will appear next to the incorrect fields, instructing the user as to where they went wrong.

#### Deleting tutors

1. Select a data row to delete in the data grid view.
2. Press the delete button in the tools group box.
3. A prompt will appear, asking if you are sure you wish to delete the selected row.
4. Pressing yes will complete deletion of the selected row.
5. Pressing no will cancel the deletion process.

#### Updating tutors

1. Select a data row to update from the data grid view.
2. Selecting a data row will auto-fill all fields within the properties section of the form with details from the data row.
3. Modifications can be made to this data as needed.
4. Pressing the update data button in the tools group box will update the selected row with the data entered once it has confirmed it to be valid and added to a class.
5. If invalid, the error provider will appear next to the invalid fields and inform the user where they went wrong.

#### Sorting tutors

1. Clicking the column by which to sort within the data grid view will sort the database by that field.

#### Searching tutors

1. The user can type the word, number, date or true/false value for which they wish to search into the search text box.
2. The user should then select the column by which they wish to search using the combo box below the search field which contains each of the columns available to be searched.
3. The user should then press the search button to commence search.
4. The data grid view will then be filtered to match the search criteria.

### Modify Exam form

#### Adding exams

1. Fill in all necessary fields.
2. Once verified and valid, the entered data will be entered into a class, after which, all information will be ported into a data row of its respective database and finally added to the database, updating the database itself with the new details. If the data entered is invalid, an error provider will appear next to the incorrect fields, instructing the user as to where they went wrong.

#### Deleting exams

1. Select a data row to delete in the data grid view.
2. Press the delete button in the tools group box.
3. A prompt will appear, asking if you are sure you wish to delete the selected row.
4. Pressing yes will complete deletion of the selected row.
5. Pressing no will cancel the deletion process.

#### Updating exams

1. Select a data row to update from the data grid view.
2. Selecting a data row will auto-fill all fields within the properties section of the form with details from the data row.
3. Modifications can be made to this data as needed.
4. Pressing the update data button in the tools group box will update the selected row with the data entered once it has confirmed it to be valid and added to a class.
5. If invalid, the error provider will appear next to the invalid fields and inform the user where they went wrong.

#### Sorting exams

1. Clicking the column by which to sort within the data grid view will sort the database by that field.

#### Searching exams

1. The user can type the word, number, date or true/false value for which they wish to search into the search text box.
2. The user should then select the column by which they wish to search using the combo box below the search field which contains each of the columns available to be searched.
3. The user should then press the search button to commence search.
4. The data grid view will then be filtered to match the search criteria.

### Modify Room form

#### Adding rooms

1. Fill in all necessary fields.
2. Once verified and valid, the entered data will be entered into a class, after which, all information will be ported into a data row of its respective database and finally added to the database, updating the database itself with the new details. If the data entered is invalid, an error provider will appear next to the incorrect fields, instructing the user as to where they went wrong.

#### Deleting rooms

1. Select a data row to delete in the data grid view.
2. Press the delete button in the tools group box.
3. A prompt will appear, asking if you are sure you wish to delete the selected row.
4. Pressing yes will complete deletion of the selected row.
5. Pressing no will cancel the deletion process.

#### Updating rooms

1. Select a data row to update from the data grid view.
2. Selecting a data row will auto-fill all fields within the properties section of the form with details from the data row.
3. Modifications can be made to this data as needed.
4. Pressing the update data button in the tools group box will update the selected row with the data entered once it has confirmed it to be valid and added to a class.
5. If invalid, the error provider will appear next to the invalid fields and inform the user where they went wrong.

#### Sorting rooms

1. Clicking the column by which to sort within the data grid view will sort the database by that field.

#### Searching rooms

1. The user can type the word, number, date or true/false value for which they wish to search into the search text box.
2. The user should then select the column by which they wish to search using the combo box below the search field which contains each of the columns available to be searched.
3. The user should then press the search button to commence search.
4. The data grid view will then be filtered to match the search criteria.

### Modify Exam Entries form

#### Adding exam entries

1. Fill in all necessary fields.
2. Once verified and valid, the entered data will be entered into a class, after which, all information will be ported into a data row of its respective database and finally added to the database, updating the database itself with the new details. If the data entered is invalid, an error provider will appear next to the incorrect fields, instructing the user as to where they went wrong.

#### Deleting exam entries

1. Select a data row to delete in the data grid view.
2. Press the delete button in the tools group box.
3. A prompt will appear, asking if you are sure you wish to delete the selected row.
4. Pressing yes will complete deletion of the selected row.
5. Pressing no will cancel the deletion process.

#### Updating exam entries

1. Select a data row to update from the data grid view.
2. Selecting a data row will auto-fill all fields within the properties section of the form with details from the data row.
3. Modifications can be made to this data as needed.
4. Pressing the update data button in the tools group box will update the selected row with the data entered once it has confirmed it to be valid and added to a class.
5. If invalid, the error provider will appear next to the invalid fields and inform the user where they went wrong.

#### Sorting exam entries

1. Clicking the column by which to sort within the data grid view will sort the database by that field.

#### Searching exam entries

1. The user can type the word, number, date or true/false value for which they wish to search into the search text box.
2. The user should then select the column by which they wish to search using the combo box below the search field which contains each of the columns available to be searched.
3. The user should then press the search button to commence search.
4. The data grid view will then be filtered to match the search criteria.

#### Paying for exam entries

1. Select a created data row that has not yet been paid for.
2. Click the pay button in the tools group box.
3. A payment window will open.
4. Select the payment method with the combo box on the right. Different payment methods will require the entry of different fields.
5. Enter required data into fields.
6. The make payment button should now be clickable, clicking on it reveals a message box informing the user that the payment was successful if all data was entered correctly. A payment record is saved to the payment table and clicking okay on the message box will send the user back to the relevant form from which the payment was requested.

### Modify Block Booking form

#### Adding Block Booking

1. Fill in all necessary fields.
2. Once verified and valid, the entered data will be entered into a class, after which, all information will be ported into a data row of its respective database and finally added to the database, updating the database itself with the new details. If the data entered is invalid, an error provider will appear next to the incorrect fields, instructing the user as to where they went wrong.

#### Deleting Block Booking

1. Select a data row to delete in the data grid view.
2. Press the delete button in the tools group box.
3. A prompt will appear, asking if you are sure you wish to delete the selected row.
4. Pressing yes will complete deletion of the selected row.
5. Pressing no will cancel the deletion process.

#### Updating Block Booking

1. Select a data row to update from the data grid view.
2. Selecting a data row will auto-fill all fields within the properties section of the form with details from the data row.
3. Modifications can be made to this data as needed.
4. Pressing the update data button in the tools group box will update the selected row with the data entered once it has confirmed it to be valid and added to a class.
5. If invalid, the error provider will appear next to the invalid fields and inform the user where they went wrong.

#### Sorting Block Booking

1. Clicking the column by which to sort within the data grid view will sort the database by that field.

#### Searching Block Booking

1. The user can type the word, number, date or true/false value for which they wish to search into the search text box.
2. The user should then select the column by which they wish to search using the combo box below the search field which contains each of the columns available to be searched.
3. The user should then press the search button to commence search.
4. The data grid view will then be filtered to match the search criteria.

#### Paying for Block Booking

1. Select a created data row that has not yet been paid for.
2. Click the pay button in the tools group box.
3. A payment window will open.
4. Select the payment method with the combo box on the right. Different payment methods will require the entry of different fields.
5. Enter required data into fields.
6. The make payment button should now be clickable, clicking on it reveals a message box informing the user that the payment was successful if all data was entered correctly. A payment record is saved to the payment table and clicking okay on the message box will send the user back to the relevant form from which the payment was requested.

### Past Payments form

#### Searching past payments

1. The user can type the word, number, date or true/false value for which they wish to search into the search text box.
2. The user should then select the column by which they wish to search using the combo box below the search field which contains each of the columns available to be searched.
3. The user should then press the search button to commence search.
4. The data grid view will then be filtered to match the search criteria.

#### Sorting past payments

1. Clicking the column by which to sort within the data grid view will sort the database by that field.

### Lesson Timetable form

#### Searching for lessons on a day

1. Select the date time picker.
2. Select a date from the date time picker.
3. The grid beneath the date time picker will now be rearranged based on the selected day.
4. Nothing will happen if the selected day was a weekend as the school is not available on those days.
5. Only half the usual periods will appear if a Friday is selected as that day is a half day for the school.

#### Finding information about a day

1. If a cell is yellow, a lesson is planned for that day.
2. If a cell is red, that room is not available for classes for whatever reason.
3. If a cell is white, that room at that time of day is open for booking.
4. Hovering the mouse cursor over a cell will reveal additional details about it, such as the room and period of the day. As well as any lessons that may be taking place within that cell.

#### Booking lessons for a day

1. Clicking on a white cell will offer the user a transfer to the timetabling for lessons form.
2. A message box will appear offering if the user would like to transfer to the form where lessons can be booked.
3. If the user says no, the message box will disappear and the form will return to normal.
4. If the user says yes, they will be transferred to the form where lessons can be booked with the day and period of day pre-selected for them. (See Modify Lessons).

### Modify Lessons

#### Adding Lessons

1. Selecting the booking will auto filter and select some other options as somethings would not be applicable to certain block bookings.
2. Fill in all necessary fields.
3. Once verified and valid, the entered data will be entered into a class, after which, all information will be ported into a data row of its respective database and finally added to the database, updating the database itself with the new details. If the data entered is invalid, an error provider will appear next to the incorrect fields, instructing the user as to where they went wrong.
4. The database will be searched and checked to ensure that the lessons generated do not conflict with other lessons before lessons are written to the database.
5. The number of lessons created is equal to the number of lessons requested within the block booking.

#### Deleting Lessons

1. Select a data row to delete in the data grid view.
2. Press the delete button in the tools group box.
3. A prompt will appear, asking if you are sure you wish to delete the selected row.
4. Pressing yes will complete deletion of the selected row.
5. Pressing no will cancel the deletion process.

#### Updating Lessons

1. Select a data row to update from the data grid view.
2. Selecting a data row will auto-fill all fields within the properties section of the form with details from the data row.
3. Modifications can be made to this data as needed.
4. Pressing the update data button in the tools group box will update the selected row with the data entered once it has confirmed it to be valid and added to a class.
5. If invalid, the error provider will appear next to the invalid fields and inform the user where they went wrong.

#### Sorting Lessons

1. Clicking the column by which to sort within the data grid view will sort the database by that field.

#### Searching Lessons

1. The user can type the word, number, date or true/false value for which they wish to search into the search text box.
2. The user should then select the column by which they wish to search using the combo box below the search field which contains each of the columns available to be searched.
3. The user should then press the search button to commence search.
4. The data grid view will then be filtered to match the search criteria.

### Report on Students

#### Querying the student report

1. Click the new query button.
2. Select the column to query.
3. Select the potential queries from the combo box below.
4. Click the add query button.

#### Clearing the query

1. Click the clear query button.

### Report on Tutors

#### Querying the Tutor report

1. Click the new query button.
2. Select the column to query.
3. Select the potential queries from the combo box below.
4. Click the add query button.

#### Clearing the query

1. Click the clear query button.

### Report on Bookings

#### Querying the Booking report

1. Click the new query button.
2. Select the column to query.
3. Select the potential queries from the combo box below.
4. Click the add query button.

#### Clearing the query

1. Click the clear query button.

### Report on Payments

#### Querying the Payment report

1. Click the new query button.
2. Select the column to query.
3. Select the potential queries from the combo box below.
4. Click the add query button.

#### Clearing the query

1. Click the clear query button.

# Implementation

## Justification of Solution

### Why C#

C# is one of the newest languages to work under the .net framework, it is also one of the few of which that is object oriented, allowing for it to produce results quickly and efficiently when compared to its counterparts such as C++ which, while powerful and efficient, does not allow for the usability of the C# language, although it does support object oriented methodologies, C++ is limited in its complexity, similarly with most other languages in the .net framework, which hinders the aspect of rapid development from the languages. This is not an issue with C#, it being designed to provide an easy, fast, simple, stable and adaptable environment from which a program can be written. Its ease of use comes from its object oriented nature, that being one which allows for pre-constructed objects to be used and have values assigned and changed, allowing for the programs code to be treated as though it were a real life object with properties and information which has been assigned to it, allowing it to be easily understood and changed by the programmer, saving a lot of time in the process, the also leads into the speed at which it can be written as its simplicity allows for less room to make mistakes, less time fixing errors and thanks to its built-in garbage collector, does not require the manual typing of the end of the lives of variables as it is all performed automatically by the compiler, saving even more time that would have been spent in a language such as C++ writing the destruction of variables. Another significant advantages C# would have over many of the other languages available would be the added features supplied to it by the visual studio programming environment, supplying the writing of the code with corrective IntelliSense, debugging tools such as break points and very controllable object management, the only other viable languages which supply these features being C++ and visual basic, C++ being ruled out due to complexity and time constraints applied by its nature as a less object oriented language and it requiring additional manual support to function such as the manual garbage collection that must be done, ruling out C++ as a contender. Visual basic was a viable contender, however, lacks some of the more useful features of C# which would allow it to better connect to the SQL database. C# also has a great support for exception handling, which is another easy to use feature which makes the code written even more stable than the language would normally allow. C# does however lose points in the areas of efficiency and general features, efficiency as the language is a Frankenstein’s monster-like composite of many predecessor languages, put together to make it as easy to use and unbreakable as possible, however, this results in fairly long compile times and generally makes the code less efficient (very slightly) than if it were written in the languages from which C# was composed. C# also lacks a few features from C++ and other languages although this I largely due to the relative newness of the language in comparison to its predecessors.

#### Advantages

* Easily typed
* Object oriented
* Good and easily used exception handling
* Automatic garbage collection
* Much lower chance of memory leaks
* Heavily supports the SQL scripting language
* Visual Studio support
* IntelliSense functionality
* Debugging tools such as break points
* Good, fast and easy resource management

#### Disadvantages

* Less efficient at run-time than many of the alternatives
* Longer compile times increase testing times
* Losses some features supplied by predecessors due to the relative newness of the language

### Why SQL

SQL, or structured Query Language is a scripting language used to create databases containing tables and other useful data storage and parsing tools. When it comes to types of databases, there is not as much contention as there is with programming languages, SQL is a fairly reliable scripting language which is capable of supplying the features and usability required to make this project work and meet the standards and requirements I expect to accomplish.

#### Advantages

* Very compatible with C#
* Very efficient

#### Disadvantages

* Troubles may arise when using two separate languages to create one project
* Data type inconsistencies
* Parsing data to SQL from C#

## SQL Tables

|  |  |  |
| --- | --- | --- |
| Table Name | Table Purpose | SQL Script |
| Student | To store data on each individual student within the music school. | CREATE TABLE Student  (  StudentNo int IDENTITY (1,1) PRIMARY KEY not null,  FirstName varchar(30)not null,  Surname varchar(30)not null,  Address varchar(30)not null,  Town varchar(30)not null,  PostCode varchar(9)not null,  Email varchar(50)not null,  DOB Date not null,  Gender char(1) not null,  Disability bit not null,  ContactTelNo varchar(13)not null,  ParentName varchar(60)not null,  DateJoined Date not null,  DateLeft Date,  CurrentStudent bit not null  ); |
| Tutor | To store data on each individual tutor within the music school. | create table Tutor  (  TutorNo int IDENTITY (1,1) PRIMARY KEY,  FirstName varchar(30)not null,  Surname varchar(30)not null,  Address varchar(30)not null,  Town varchar(30)not null,  PostCode varchar(9)not null,  Email varchar(50)not null,  DOB Date not null,  Gender char not null,  Disability bit not null,  ContactTelNo varchar(13)not null,  NextOfKin varchar(60)not null,  DateJoined Date not null,  DateLeft Date,  CurrentTutor bit not null  ); |
| ExternalExam | To store data on each individual exam within the music school. | create table ExternalExam  (  ExamNo int IDENTITY(1,1) PRIMARY KEY,  ExamBoard varchar(30) not null,  TitleExam varchar(30) not null,  Grade char not null,  DateExam Date not null,  TimeExam Time not null,  ExamFee decimal(7,2) not null,  AcceptingEntries bit not null  ); |
| ExamEntry | To store data on each individual student entry for an exam within the music school. | CREATE TABLE ExamEntry  (  ExamEntryNo int IDENTITY (1,1) PRIMARY KEY,  ExamNo int FOREIGN KEY references ExternalExam(ExamNo) not null,  StudentNo int FOREIGN KEY references Student(StudentNo) not null,  DateEntry Date not null,  Fee decimal(7,2) not null,  Paid bit not null,  DatePaid Date  ); |
| ExamResult | To store data on each individual student exam result within the music school. | create table ExamResult  (  ExamResultNo int IDENTITY (1,1) PRIMARY KEY,  ExamNo int FOREIGN KEY references ExternalExam(ExamNo) not null,  StudentNo int FOREIGN KEY references Student(StudentNo) not null,  Result char(1) not null,  DateCertificateRecieved Date not null,  DateCertificateIssued Date not null,  CollectedBy varchar(60) not null,  Collected bit not null  ); |
| TuitionChoice | To store data on each individual tuition choice available to the students within the music school. | create table TuitionChoice  (  TuitionChoice varchar(50) PRIMARY KEY,  CurrentlyOffered bit not null  ); |
| Room | To store data on each individual room within the music school. | CREATE TABLE Room  (  RoomNo int IDENTITY(1,1) PRIMARY KEY,  RoomType varchar(30) not null,  Capacity int not null,  InUse bit not null  ); |
| Payment | To store data on each individual payment made to the music school. | create table Payment  (  PaymentNo int IDENTITY (1,1) PRIMARY KEY,  RefNo int not null,  StudentNo int FOREIGN KEY references Student(StudentNo) not null,  Description varchar(50) not null,  DatePaid Date not null,  AmountPaid decimal(7,2) not null,  PaymentMethod varchar(50) not null,  ResponsibleEmployee varchar(60) not null,  Sponsor varchar(60),  SponsorOwner varchar(60),  Paid bit not null  ); |
| BlockBooking | To store data on each individual booking made by a student within the music school. | create table BlockBooking  (  BookingNo int IDENTITY (1,1) PRIMARY KEY,  StudentNo int FOREIGN KEY references Student(StudentNo) not null,  TuitionChoice varchar(50) FOREIGN KEY references TuitionChoice(TuitionChoice) not null,  DateBooking Date not null,  AbilityLevel varchar(1) not null,  NoLessons int not null,  DiscountRate int not null,  LessonRate decimal(7,2) not null,  TotalDue decimal(7,2) not null,  Paid bit not null,  WaitingList bit not null  ); |
| Report | TODO | create table Report  (  ReportNo int IDENTITY (1,1) PRIMARY KEY,  StudentNo int FOREIGN KEY references Student(StudentNo) not null,  TutorNo int FOREIGN KEY references Tutor(TutorNo) not null,  AcadYear varchar(10) not null,  Term int not null,  ReportComments varchar(1000)  ); |
| TimeTabledLesson | To store data on each individual lesson scheduled into the timetable within the music school. | create table TimetabledLesson  (  TLNo int IDENTITY (1,1) PRIMARY KEY,  TutorNo int FOREIGN KEY references Tutor(TutorNo),  StudentNo int FOREIGN KEY references Student(StudentNo),  RoomNo int FOREIGN KEY references Room(RoomNo),  BookingNo int FOREIGN KEY references BlockBooking(BookingNo),  DateLesson date not null,  TimeLesson smallint not null,  StudentPresent bit not null,  Cancellation bit not null,  CancellationReason varchar(1000)  ); |
| TutorTakes | To store data on what instrument each tutor within the music school can teach. | create table TutorTakes  (  TutorTakesNo int IDENTITY (1,1) PRIMARY KEY,  TutorNo int FOREIGN KEY references Tutor(TutorNo),  TuitionChoice varchar(50) FOREIGN KEY references TuitionChoice(TuitionChoice),  TeachUpToGrade varchar(1) not null  ); |

## Implementation of C# classes

|  |  |  |
| --- | --- | --- |
| Class name | Purpose | C# Code |
| Student |  | using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Mitchell\_School\_of\_Music  {  class Student  {  //fields  string firstName;  string surname;  string address;  string town;  string postCode;  string email;  DateTime dOB;  bool gender;  bool disability;  string contactTelNo;  string parentName;  DateTime dateJoined;  DateTime dateLeft;  bool currentStudent;    //properties  public string FirstName  {  get { return firstName; }  set  {  //check and set if valid  if (Utilities.ValidString(value, 1, 30))  {  firstName = value;  }  else  {  throw new InvalidDataException("Firstname must be between 1 and 30 characters long.");  }  }  }  public string Surname  {  get { return surname; }  set  {  //check and set if valid  if (Utilities.ValidString(value, 1, 30))  {  surname = value;  }  else  {  throw new InvalidDataException("Surname must be between 1 and 30 characters long.");  }  }  }  public string Address  {  get { return address; }  set  {  //check and set if valid  if (Utilities.ValidString(value, 6, 30))  {  address = value;  }  else  {  throw new InvalidDataException("Address must be between 6 and 30 characters long.");  }  }  }  public string Town  {  get { return town; }  set  {  //check and set if valid  if (Utilities.ValidString(value, 1, 30))  {  town = value;  }  else  {  throw new InvalidDataException("Town must be between 1 and 30 characters long.");  }  }  }  public string PostCode  {  get { return postCode; }  set  {  //check and set if valid  if (Utilities.ValidString(value, 6, 9))  {  if (Utilities.ValidPostcode(value))  {  postCode = value;  }  else  {  throw new InvalidDataException("Postcode must be a valid postcode.");  }  }  else  {  throw new InvalidDataException("Postcode must be between 6 and 9 characters long.");  }  }  }  public string Email  {  get { return email; }  set  {  //check and set if valid  if (Utilities.ValidString(value, 5, 50))  {  email = value;  }  else  {  throw new InvalidDataException("Email must be between 5 and 50 characters long.");  }  }  }  public string DOB  {  get { return dOB.ToString(); }  set  {  //check and set if valid  if (Utilities.ValidDate(value, DateTime.Now, DateTime.Now.AddYears(-100)))  {  dOB = DateTime.Parse(value);  }  else  {  throw new InvalidDataException("Date of birth must be a valid date between the years " + DateTime.Now.Year + " and " + DateTime.Now.AddYears(-100) + ".");  }  }  }  public bool Gender  {  get { return gender; }  set  {  //check and set if valid  if (Utilities.ValidBool(value))  {  gender = value;  }  else  {  throw new InvalidDataException("Please enter a value for gender.");  }  }  }  public bool Disability  {  get { return disability; }  set  {  //check and set if valid  if (Utilities.ValidBool(value))  {  disability = value;  }  else  {  throw new InvalidDataException("Please enter a value for disability.");  }  }  }  public string ContactTelNo  {  get { return contactTelNo; }  set  {  //check and set if valid  if (Utilities.ValidString(value, 13, 13))  {  if (Utilities.ValidTelNo(value))  {  contactTelNo = value;  }  else  {  throw new InvalidDataException("Telephone number must be a valid telephone number.");  }  }  else  {  throw new InvalidDataException("Telephone number must be a valid number between 8 and 12 characters long.");  }  }  }  public string ParentName  {  get { return parentName; }  set  {  //check and set if valid  if (Utilities.ValidString(value, 3, 60))  {  parentName = value;  }  else  {  throw new InvalidDataException("Parent name must be between 3 and 60 characters long.");  }  }  }  public DateTime DateJoined  {  get { return dateJoined; }  set  {  //check and set if valid  if (Utilities.ValidDate(value, DateTime.Now.Date, DateTime.Now.Date))  {  dateJoined = value;  }  else  {  throw new InvalidDataException("Date joined is in an incorrect format.");  }  }  }  public string DateLeft  {  get { return dateLeft.ToString(); }  set  {  //check and set if valid  if (Utilities.ValidDate(value, DateTime.Now.Date, dateJoined.Date))  {  dateLeft = DateTime.Parse(value);  }  else  {  throw new InvalidDataException("Date left must be between either " + DateTime.Now.Date.ToString("dd/MM/yyyy)") + " and the date the person joined (" + dateJoined + ").");  }  }  }  public bool CurrentStudent  {  get { return currentStudent; }  set  {  //check and set if valid  if (Utilities.ValidBool(value))  {  currentStudent = value;  }  else  {  throw new InvalidDataException("Input a value for whether this is a current student.");  }  }  }  }  } |
| Tutor |  | using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Mitchell\_School\_of\_Music  {  class Tutor  {  //fields  string firstName;  string surname;  string address;  string town;  string postCode;  string email;  DateTime dOB;  bool gender;  bool disability;  string contactTelNo;  string nextOfKin;  DateTime dateJoined;  DateTime dateLeft;  bool currentTutor;  //properties  public string FirstName  {  get { return firstName; }  set  {  //check and set if valid  if (Utilities.ValidString(value, 1, 30))  {  firstName = value;  }  else  {  throw new InvalidDataException("Firstname must be between 1 and 30 characters long.");  }  }  }  public string Surname  {  get { return surname; }  set  {  //check and set if valid  if (Utilities.ValidString(value, 1, 30))  {  surname = value;  }  else  {  throw new InvalidDataException("Surname must be between 1 and 30 characters long.");  }  }  }  public string Address  {  get { return address; }  set  {  //check and set if valid  if (Utilities.ValidString(value, 6, 30))  {  address = value;  }  else  {  throw new InvalidDataException("Address must be between 6 and 30 characters long.");  }  }  }  public string Town  {  get { return town; }  set  {  //check and set if valid  if (Utilities.ValidString(value, 1, 30))  {  town = value;  }  else  {  throw new InvalidDataException("Town must be between 1 and 30 characters long.");  }  }  }  public string PostCode  {  get { return postCode; }  set  {  //check and set if valid  if (Utilities.ValidString(value, 6, 9))  {  //check and set if valid  if (Utilities.ValidPostcode(value))  {  postCode = value;  }  else  {  throw new InvalidDataException("Postcode must be a valid postcode.");  }  }  else  {  throw new InvalidDataException("Postcode must be between 6 and 9 characters long.");  }  }  }  public string Email  {  get { return email; }  set  {  //check and set if valid  if (Utilities.ValidString(value, 5, 50))  {  email = value;  }  else  {  throw new InvalidDataException("Email must be between 5 and 50 characters long.");  }  }  }  public string DOB  {  get { return dOB.ToString(); }  set  {  //check and set if valid  if (Utilities.ValidDate(value, DateTime.Now, DateTime.Now.AddYears(-100)))  {  dOB = DateTime.Parse(value);  }  else  {  throw new InvalidDataException("Date of birth must be a valid date between the years " + DateTime.Now.Year + " and " + DateTime.Now.AddYears(-100) + ".");  }  }  }  public bool Gender  {  get { return gender; }  set  {  //check and set if valid  if (Utilities.ValidBool(value))  {  gender = value;  }  else  {  throw new InvalidDataException("Please enter a value for gender.");  }  }  }  public bool Disability  {  get { return disability; }  set  {  //check and set if valid  if (Utilities.ValidBool(value))  {  disability = value;  }  else  {  throw new InvalidDataException("Please enter a value for disability.");  }  }  }  public string ContactTelNo  {  get { return contactTelNo; }  set  {  //check and set if valid  if (Utilities.ValidString(value, 13, 13))  {  if (Utilities.ValidTelNo(value))  {  contactTelNo = value;  }  else  {  throw new InvalidDataException("Telephone number must be a valid telephone number.");  }  }  else  {  throw new InvalidDataException("Telephone number must be a valid number between 8 and 12 characters long.");  }  }  }  public DateTime DateJoined  {  get { return dateJoined; }  set  {  //check and set if valid  if (Utilities.ValidDate(value, DateTime.Now, DateTime.Now))  {  dateJoined = value;  }  else  {  throw new InvalidDataException("Date joined is in an incorrect format.");  }  }  }  public string DateLeft  {  get { return dateLeft.ToString(); }  set  {  //check and set if valid  if (Utilities.ValidDate(value, DateTime.Now, dateJoined))  {  dateLeft = DateTime.Parse(value);  }  else  {  throw new InvalidDataException("Date left must be between either " + DateTime.Now + " and the date the person joined (" + dateJoined + ").");  }  }  }  public bool CurrentTutor  {  get { return currentTutor; }  set  {  //check and set if valid  if (Utilities.ValidBool(value))  {  currentTutor = value;  }  else  {  throw new InvalidDataException("Input a value for wheather this is a current tutor.");  }  }  }  public string NextOfKin  {  get { return nextOfKin; }  set  {  //check and set if valid  if (Utilities.ValidString(value, 3, 60))  {  nextOfKin = value;  }  else  {  throw new InvalidDataException("Next of kin name must be between 3 and 60 characters long.");  }  }  }  }  } |
| ExamEntry |  | using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Mitchell\_School\_of\_Music  {  class ExamEntry  {  //fields  int examNo;  int studentNo;  DateTime dateEntry;  float fee;  bool paid;  DateTime datePaid;  //properties  public int ExamNo  {  get { return examNo; }  set  {  //check and set if valid  if (Utilities.ValidNumber(value, int.MaxValue, 1))  {  examNo = value;  }  else  {  throw new InvalidDataException("Exam number must be between 1 and " + int.MaxValue + " characters long.");  }  }  }  public int StudentNo  {  get { return studentNo; }  set  {  //check and set if valid  if (Utilities.ValidNumber(value, int.MaxValue, 1))  {  studentNo = value;  }  else  {  throw new InvalidDataException("Student number must be between 1 and " + int.MaxValue + " characters long.");  }  }  }  public DateTime DateEntry  {  get { return dateEntry; }  set  {  //check and set if valid  if (Utilities.ValidDate(value, DateTime.Now.AddYears(5), DateTime.Now.Date))  {  dateEntry = value;  }  else  {  throw new InvalidDataException("The entry date for the exam must be within 5 years of now (" + DateTime.Now.Date + " - " + DateTime.Now.AddYears(5) + ").");  }  }  }  public float Fee  {  get { return fee; }  set  {  //check and set if valid  if (Utilities.ValidNumber(value, 1000.00, 0.00))  {  fee = value;  }  else  {  throw new InvalidDataException("Fee must be no more than £1000.00 and no less than £0.00.");  }  }  }  public bool Paid  {  get { return paid; }  set  {  //check and set if valid  if (Utilities.ValidBool(value))  {  paid = value;  }  else  {  throw new InvalidDataException("Payment state can only be true or false");  }  }  }  public string DatePaid  {  get { return datePaid.ToString(); }  set  {  //check and set if valid  if (Utilities.ValidDate(value, DateTime.Now, DateTime.Now.AddYears(-3)))  {  datePaid = DateTime.Parse(value);  }  else  {  throw new InvalidDataException("The payment must have been recieved before ths can be created, payment must have been made within the last 3 years. (" + DateTime.Now + " - " + DateTime.Now.AddYears(-3) + ").");  }  }  }  }  } |
| ExternalExam |  | using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Mitchell\_School\_of\_Music  {  class ExternalExam  {  //fields  string examBoard;  string titleExam;  char grade;  DateTime examDateTime;  float examFee;  bool acceptingEntries;    //properties  public string ExamBoard  {  get { return examBoard; }  set  {  //check and set if valid  if (Utilities.ValidString(value, 1, 30))  {  examBoard = value;  }  else  {  throw new InvalidDataException("Examination board name must be between 1 and 30 characters long.");  }  }  }  public string TitleExam  {  get { return titleExam; }  set  {  //check and set if valid  if (Utilities.ValidString(value, 1, 30))  {  titleExam = value;  }  else  {  throw new InvalidDataException("Title of exam must be between 1 and 30 characters long.");  }  }  }  public char Grade  {  get { return grade; }  set  {  //check and set if valid  if (Utilities.ValidChar(value))  {  grade = value;  }  else  {  throw new InvalidDataException("Grade entered is not a valid grade expected for this field.");  }  }  }  public string ExamDateStringInOut  {  get { return examDateTime.Date.ToString(); }  set  {  //check and set if valid  if (Utilities.ValidDate(value, DateTime.Now.AddYears(5), DateTime.Now))  {  examDateTime = DateTime.Parse(value);  }  else  {  throw new InvalidDataException("The date and time of the exam must be after " + DateTime.Now + " and within 5 years advance of today (" + DateTime.Now.AddYears(5) + ").");  }  }  }  public string ExamTimeStringInOut  {  get { return examDateTime.TimeOfDay.ToString(); }  set  {  //check and set if valid  if (Utilities.ValidTime(value))  {  examDateTime = examDateTime.AddHours(double.Parse(value.Substring(0, 2)));  examDateTime = examDateTime.AddMinutes(double.Parse(value.Substring(3, 2)));  }  else  {  throw new InvalidDataException("The entered time must be in a valid 24hour format.");  }  }  }  public DateTime ExamDateTimeInOut  {  get { return examDateTime; }  set  {  //check and set if valid  if (Utilities.ValidDate(value, DateTime.Now.AddYears(5), DateTime.Now))  {  examDateTime = value;  }  else  {  throw new InvalidDataException("The date and time of the exam must be after " + DateTime.Now + " and within 5 years advance of today (" + DateTime.Now.AddYears(5) + ").");  }  }  }  public float ExamFee  {  get  { return examFee; }  set  {  //check and set if valid  if (Utilities.ValidNumber(value, 1000.00, 0.00))  {  examFee = value;  }  else  {  throw new InvalidDataException("The price cannot be beyond the £1000 payment point and can also not be below £0.");  }  }  }  public bool AcceptingEntries  {  get { return acceptingEntries; }  set  {  //check and set if valid  if (Utilities.ValidBool(value))  {  acceptingEntries = value;  }  else  {  throw new InvalidDataException("The value entered for accepting entries is not a valid true or false.");  }  }  }  }  } |
| BlockBooking |  | using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Mitchell\_School\_of\_Music  {  class BlockBooking  {  //fields  private int studentNo;  private DateTime dateBooking;  private string tuitionChoice;  private char abilityLevel;  private Int16 noLessons;  private byte discountRate;  private float lessonRate;  private float totalDue;  private bool paid;  private bool waitingList;  //properties  //StudentNo property  public int StudentNo  {  get { return studentNo; }  set  {  //check and set if valid  if (Utilities.ValidNumber(value, int.MaxValue, 1))  {  studentNo = value;  }  else  {  throw new InvalidDataException("Entered number is not a valid number.");  }  }  }  //Date property  public DateTime DateBooking  {  get { return dateBooking; }  set  {  //check and set if valid  if (Utilities.ValidDate(value, DateTime.Now, DateTime.MinValue.Date))  {  dateBooking = value;  }  else  {  throw new InvalidDataException("The date specified is not valid.");  }  }  }  public string TuitionChoice  {  get { return tuitionChoice; }  set  {  //check and set if valid  if (Utilities.ValidString(value, 1, 50))  {  tuitionChoice = value;  }  else  {  throw new InvalidDataException("The entered set of characters must be a valid set within the range of 1 - 50 characters.");  }  }  }  public char AbilityLevel  {  get { return abilityLevel; }  set  {  //check and set if valid  if (Utilities.ValidChar(value))  {  abilityLevel = value;  }  else  {  throw new InvalidDataException("Entered ability level is not one of the ability levels available. Please enter an appropriate ability level.");  }  }  }  public Int16 NoLessons  {  get { return noLessons; }  set  {  //check and set if valid  if (Utilities.ValidNumber(value, Int16.MaxValue, 3))  {  noLessons = value;  }  else  {  throw new InvalidDataException("The number of lessons planned must be between 3 and " + Int16.MaxValue + ".");  }  }  }  public byte DiscountRate  {  get { return discountRate; }  set  {  //check and set if valid  if (Utilities.ValidNumber(value, 100, 0))  {  discountRate = value;  }  else  {  throw new InvalidDataException("The rate of discount goes beyond the percentage range of a discount (0 - 100).");  }  }  }  public float LessonRate  {  get { return lessonRate; }  set  {  //check and set if valid  if (Utilities.ValidNumber(value, float.MaxValue, 0))  {  lessonRate = value;  }  else  {  throw new InvalidDataException("The Lesson rate specified goes beyond the allowed range (" + float.MaxValue + " - " + 0 + ").");  }  }  }  public float TotalDue  {  get { return totalDue; }  set  {  //check and set if valid  if (Utilities.ValidNumber(value, float.MaxValue, 0))  {  totalDue = value;  }  else  {  throw new InvalidDataException("The total due value is not an acceptable value within the range of " + float.MaxValue + " - 0.");  }  }  }  public bool Paid  {  get { return paid; }  set  {  //check and set if valid  if (Utilities.ValidBool(value))  {  paid = value;  }  else  {  throw new InvalidDataException("Error occured with the state of the payment.");  }  }  }  public bool WaitingList  {  get { return waitingList; }  set  {  //check and set if valid  if (Utilities.ValidBool(value))  {  waitingList = value;  }  else  {  throw new InvalidDataException("Error occured with the state of the payment.");  }  }  }  }  } |
| Payment |  | using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Mitchell\_School\_of\_Music  {  public class Payment  {  int refNo;  int studentNo;  string description;  DateTime datePaid;  float amountPaid;  string paymentMethod;  string responsibleEmplyee;  string sponsor;  string sponsorOwner;  bool paid;  public int RefNo  {  get { return refNo; }  set  {  //check and set if valid  if (Utilities.ValidNumber(value, int.MaxValue, 1))  {  refNo = value;  }  else  {  throw new InvalidDataException("Reference number entered is not a valid reference number.");  }  }  }  public int StudentNo  {  get { return studentNo; }  set  {  //check and set if valid  if (Utilities.ValidNumber(value, int.MaxValue, 1))  {  studentNo = value;  }  else  {  throw new InvalidDataException("Student number used is not a valid student number.");  }  }  }  public string Description  {  get { return description; }  set  {  //check and set if valid  if (Utilities.ValidString(value, 1, 50))  {  description = value;  }  else  {  throw new InvalidDataException("The description of the payment must be a valid description, explaining the reason for payment.");  }  }  }  public DateTime DatePaid  {  get { return datePaid; }  set  {  //check and set if valid  if (Utilities.ValidDate(value, DateTime.Now.Date, DateTime.Now.Date))  {  datePaid = value.Date;  }  else  {  throw new InvalidDataException("The date of the payment is not a valid date.");  }  }  }  public float AmountPaid  {  get { return amountPaid; }  set  {  //check and set if valid  if (Utilities.ValidNumber(value, 1000, 0))  {  amountPaid = value;  }  else  {  throw new InvalidDataException("The amount paid is not a valid amount of currency.");  }  }  }  public string PaymentMethod  {  get { return paymentMethod; }  set  {  //check and set if valid  if (Utilities.ValidString(value, 1, 50))  {  paymentMethod = value;  }  else  {  throw new InvalidDataException("The entered method of payment is not a valid method of payment.");  }  }  }  public string ResponsibleEmployee  {  get { return responsibleEmplyee; }  set  {  //check and set if valid  if (Utilities.ValidString(value, 1, 60))  {  responsibleEmplyee = value;  }  else  {  throw new InvalidDataException("The responsible employees name can be at most 60 characters, initials or recognisable signature names may be used.");  }  }  }  public string Sponsor  {  get { return sponsor; }  set  {  //check and set if valid  if (Utilities.ValidString(value, 1, 60))  {  sponsor = value;  }  else  {  throw new InvalidDataException("The name of a sponsor must be at most 60 characters long, a shortened name may be used if necessary.");  }  }  }  public string SponsorOwner  {  get { return sponsorOwner; }  set  {  //check and set if valid  if (Utilities.ValidString(value, 1, 60))  {  sponsorOwner = value;  }  else  {  throw new InvalidDataException("The owner of the sponsor, must be at most 60 characters long, a shortened name may be used if necessary.");  }  }  }  public bool Paid  {  get { return paid; }  set  {  //check and set if valid  if (Utilities.ValidBool(value))  {  paid = value;  }  else  {  throw new InvalidDataException("The payments success must be indicated with the appropriate variable.");  }  }  }  }  } |
| Exceptions |  | using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Mitchell\_School\_of\_Music  {  public class InvalidDataException : Exception  {  public InvalidDataException()  {  }  public InvalidDataException(string Message) : base(Message)  {  }  }  class SQLFailureException : Exception  {  public SQLFailureException()  {  }  public SQLFailureException(string Message) : base(Message)  {  }  }  } |
| Room |  | using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Mitchell\_School\_of\_Music  {  class Room  {  //fields  string roomType;  ushort capacity;  bool inUse;    //properties  public string RoomType  {  get { return roomType; }  set  {  //check and set if valid  if (Utilities.ValidString(value, 1, 20))  {  roomType = value;  }  else  {  throw new InvalidDataException("The room type must not be longer than 20 characters, a shortened word or phrase may be used instead");  }  }  }  public ushort Capacity  {  get { return capacity; }  set  {  //check and set if valid  if (Utilities.ValidNumber(value, 1000, 1))  {  capacity = value;  }  else  {  throw new InvalidDataException("The capacity entered is not a valid capacity. The maximum capacity allowed is 1000.");  }  }  }  public bool InUse  {  get { return inUse; }  set  {  //check and set if valid  if (Utilities.ValidBool(value))  {  inUse = value;  }  else  {  throw new InvalidDataException("");  }  }  }  }  } |
| TutorTakes |  | using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Mitchell\_School\_of\_Music  {  class TutorTakes  {  private int tutorNo;  private string tuitionChoice;  private char teachUpTo;  public int TutorNo  {  get { return tutorNo; }  set  {  //check and set if valid  if (Utilities.ValidNumber(value, int.MaxValue, 1))  {  tutorNo = value;  }  else  {  throw new InvalidDataException(value + " was not a valid number as specified.");  }  }  }  public string TuitionChoice  {  get { return tuitionChoice; }  set  {  //check and set if valid  if (Utilities.ValidString(value, 1, 50))  {  tuitionChoice = value;  }  else  {  throw new InvalidDataException(value + " was not a valid set of characters as specified.");  }  }  }  public char TeachUpTo  {  get { return teachUpTo; }  set  {  //check and set if valid  if (Utilities.ValidChar(value))  {  teachUpTo = value;  }  else  {  throw new InvalidDataException(value + " was not a valid set of characters as specified.");  }  }  }  }  } |
| TimetabledLesson |  | using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Mitchell\_School\_of\_Music  {  class TimetabledLesson  {  //fields  private int tutorNo;  private int studentNo;  private int roomNo;  private int bookingNo;  private DateTime dateTimeLesson;  private byte periodLesson;  private bool studentPresent;  private bool cancellation;  private string cancellationReason;  //properties  public int TutorNo  {  get { return tutorNo;}  set  {  //check and set if valid  if (Utilities.ValidNumber(value, int.MaxValue, 1))  {  tutorNo = value;  }  else  {  throw new InvalidDataException("Entered number is not a valid number.");  }  }  }  public int StudentNo  {  get { return studentNo;}  set  {  //check and set if valid  if (Utilities.ValidNumber(value, int.MaxValue, 1))  {  studentNo = value;  }  else  {  throw new InvalidDataException("Entered number is not a valid number.");  }  }  }  public int RoomNo  {  get { return roomNo;}  set  {  //check and set if valid  if (Utilities.ValidNumber(value, int.MaxValue, 1))  {  roomNo = value;  }  else  {  throw new InvalidDataException("Entered number is not a valid number.");  }  }  }  public int BookingNo  {  get { return bookingNo;}  set  {  //check and set if valid  if (Utilities.ValidNumber(value, int.MaxValue, 1))  {  bookingNo = value;  }  else  {  throw new InvalidDataException("Entered number is not a valid number.");  }  }  }  public DateTime DateTimeLesson  {  get { return dateTimeLesson;}  set  {  //check and set if valid  if (Utilities.ValidDate(value, DateTime.Now.AddYears(5), DateTime.Parse("01/01/2017")))  {  dateTimeLesson = value;  }  else  {  throw new InvalidDataException("Your first lesson must be within 5 years from today.");  }  }  }  public byte PeriodLesson  {  get { return periodLesson;}  set  {  //check and set if valid  if (Utilities.ValidNumber(value, byte.MaxValue, 1))  {  periodLesson = value;  }  else  {  throw new InvalidDataException("There is a limit of " + byte.MaxValue + " and a minimum of 1.");  }  }  }  public bool StudentPresent  {  get { return studentPresent;}  set  {  //check and set if valid  if (Utilities.ValidBool(value))  {  studentPresent = value;  }  else  {  throw new InvalidDataException("Student present value was not recieved or was invalid, please try again. (value recieved = " + value + ").");  }  }  }  public bool Cancellation  {  get { return cancellation;}  set  {  //check and set if valid  if (Utilities.ValidBool(value))  {  cancellation = value;  }  else  {  throw new InvalidDataException("Cancellation value was not recieved or was invalid, please try again. (value recieved = " + value + ").");  }  }  }  public string CancellationReason  {  get { return cancellationReason;}  set  {  //check and set if valid  if (Utilities.ValidString(value, 0, 1000))  {  cancellationReason = value;  }  else  {  throw new InvalidDataException("The reason for cancelling must be at most 1000 characters.");  }  }  }  }  } |
| Utilities |  | using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Text.RegularExpressions;  using System.Data;  using System.Threading.Tasks;  using System.Windows.Forms;  namespace Mitchell\_School\_of\_Music  {  public static class Utilities  {  //Ensures a string is valid in relation to the defined parameters.  public static bool ValidString(string value, int min, int max)  {  try  {  //Checks to ensure passed string is within the defined parameters for length. Doing so will also check if the string entered is null, in either case, this will ensure the string is valid.  if (value.Length >= min && value.Length <= max && !string.IsNullOrWhiteSpace(value))  {  return true;  }  else  {  return false;  }  }  catch  {  return false;  }  }  //Checks if entered value is a valid character (not null)  public static bool ValidChar(char? value)  {  try  {  if (value != null)  {  return true;  }  else  {  return false;  }  }  catch  {  return false;  }  }  //checks if string entered is a single character  public static bool ValidChar(string value)  {  try  {  char temp;  return (char.TryParse(value, out temp));  }  catch  {  return false;  }  }  //Ensures the integer entered is a valid number by ensuring it is within the defined parameters.  public static bool ValidNumber(int? value, int maxValue, int minValue)  {  try  {  if (value <= maxValue && value >= minValue)  {  return true;  }  else  {  return false;  }  }  catch  {  return false;  }  }  //Overload of the valid integer so as double variables can also be checked to be valid in terms of the determined paramenters.  public static bool ValidNumber(double? value, double maxValue, double minValue)  {  try  {  if (value <= maxValue && value >= minValue)  {  return true;  }  else  {  return false;  }  }  catch  {  return false;  }  }  //Overload of the valid integer so as float variables can also be checked to be valid in terms of the determined paramenters.  public static bool ValidNumber(float? value, double maxValue, double minValue)  {  try  {  if (value <= maxValue && value >= minValue)  {  return true;  }  else  {  return false;  }  }  catch  {  return false;  }  }  //Ensures the date entered is valid by ensuring it can be parsed to a DateTime format as well as falling between the min and max parameters.  public static bool ValidDate(string value, DateTime maxValue, DateTime minValue)  {  try  {  DateTime temp = DateTime.Parse(value);  if (temp <= maxValue && temp >= minValue)  {  return true;  }  else  {  return false;  }  }  catch  {  return false;  }  }  //checks if string entered is a valid date  public static bool ValidDate(string value)  {  try  {  DateTime.Parse(value);  return true;  }  catch  {  return false;  }  }  //Overload of the validate the date method (ValidDate) so as to validate the DateTime format rather than just the string to DateTime format.  public static bool ValidDate(DateTime? value, DateTime maxValue, DateTime minvalue)  {  try  {  if (value.HasValue && value <= maxValue && value >= minvalue)  return true;  else  return false;  }  catch  {  return false;  }  }  //checks if entered date is null  public static bool ValidDate(DateTime? value)  {  try  {  DateTime temp;  return DateTime.TryParse(value.ToString(), out temp);  }  catch  {  return false;  }  }  //checks if entered time is a valid time  public static bool ValidTime(string Time)  {  try  {  if (int.Parse(Time.Substring(0,2)) < 24 && int.Parse(Time.Substring(0,2)) >= 0 && int.Parse(Time.Substring(3,2)) < 60 && int.Parse(Time.Substring(3, 2)) >= 0)  return true;  else  return false;  }  catch  {  return false;  }  }  //ensures that a bool is not null.  public static bool ValidBool(bool? value)  {  try  {  if (value != null)  {  return true;  }  else  {  return false;  }  }  catch  {  return false;  }  }  //Checks if the string entered is a valid bool value  public static bool ValidBool(string value)  {  try  {  bool.Parse(value);  return true;  }  catch  {  return false;  }  }  //Filters a string array so as only unique values are included  public static string[] UniqueArrayData(string[] Values)  {  try  {  string[] UniqueArray = new string[Values.Length];  UniqueArray = Values.Distinct().ToArray();  return UniqueArray;  }  catch (Exception ex)  {  throw new Exception("Failed to convert array to distinct values.\r\n\r\nAdditional information:\r\n" + ex.Message);  }  }  // performs a regex check to ensure the entered telno is a valid british telno.  public static bool ValidTelNo(string value)  {  return Regex.IsMatch(value, @"^(((\+44\s?\d{4}|\(?0\d{4}\)?)\s?\d{3}\s?\d{3})|((\+44\s?\d{3}|\(?0\d{3}\)?)\s?\d{3}\s?\d{4})|((\+44\s?\d{2}|\(?0\d{2}\)?)\s?\d{4}\s?\d{4}))(\s?\#(\d{4}|\d{3}))?$");  }  // performs a regex check to ensure the entered postcode is a valid british postcode.  public static bool ValidPostcode(string value)  {  return  (  Regex.IsMatch(value, "(^[A-PR-UWYZa-pr-uwyz][0-9][ ]\*[0-9][ABD-HJLNP-UW-Zabd-hjlnp-uw-z]{2}$)") ||  Regex.IsMatch(value, "(^[A-PR-UWYZa-pr-uwyz][0-9][0-9][ ]\*[0-9][ABD-HJLNP-UW-Zabd-hjlnp-uw-z]{2}$)") ||  Regex.IsMatch(value, "(^[A-PR-UWYZa-pr-uwyz][A-HK-Ya-hk-y][0-9][ ]\*[0-9][ABD-HJLNP-UW-Zabd-hjlnp-uw-z]{2}$)") ||  Regex.IsMatch(value, "(^[A-PR-UWYZa-pr-uwyz][A-HK-Ya-hk-y][0-9][0-9][ ]\*[0-9][ABD-HJLNP-UW-Zabd-hjlnp-uw-z]{2}$)") ||  Regex.IsMatch(value, "(^[A-PR-UWYZa-pr-uwyz][0-9][A-HJKS-UWa-hjks-uw][ ]\*[0-9][ABD-HJLNP-UW-Zabd-hjlnp-uw-z]{2}$)") ||  Regex.IsMatch(value, "(^[A-PR-UWYZa-pr-uwyz][A-HK-Ya-hk-y][0-9][A-Za-z][ ]\*[0-9][ABD-HJLNP-UW-Zabd-hjlnp-uw-z]{2}$)") ||  Regex.IsMatch(value, "(^[Gg][Ii][Rr][]\*0[Aa][Aa]$)")  );  }  //provides a data view with a query of the entered data.  public static DataView DataTableFilter(string table, string Column, string search, string priSort)  {  try  {  DataView dv;  //checks if the requested column is a string datatype  if (DataAccess.ds.Tables[table].Columns[Column].DataType == typeof(string))  {  dv = new DataView(DataAccess.ds.Tables[table], Column + " Like '" + search + "%' ", priSort + " ASC", DataViewRowState.CurrentRows);  return dv;  }  //checks if the requested column is a int datatype  else if (DataAccess.ds.Tables[table].Columns[Column].DataType == typeof(int))  {  dv = new DataView(DataAccess.ds.Tables[table], Column + " = " + int.Parse(search), priSort + " ASC", DataViewRowState.CurrentRows);  return dv;  }  //checks if the requested column is a decimal datatype  else if (DataAccess.ds.Tables[table].Columns[Column].DataType == typeof(decimal))  {  dv = new DataView(DataAccess.ds.Tables[table], Column + " = " + decimal.Parse(search), priSort + " ASC", DataViewRowState.CurrentRows);  return dv;  }  //checks if the requested column is a bool datatype  else if (DataAccess.ds.Tables[table].Columns[Column].DataType == typeof(bool))  {  if (Utilities.ValidBool(search))  {  dv = new DataView(DataAccess.ds.Tables[table], Column + " = '" + bool.Parse(search) + "'", priSort + " ASC", DataViewRowState.CurrentRows);  return dv;  }  else  {  throw new Exception("Value entered is not a valid true or false value.");  }  }  //checks if the requested column is a DateTime datatype  else if (DataAccess.ds.Tables[table].Columns[Column].DataType == typeof(DateTime))  {  if (Utilities.ValidDate(search))  {  dv = new DataView(DataAccess.ds.Tables[table], Column + " = '" + DateTime.Parse(search) + "'", priSort + " ASC", DataViewRowState.CurrentRows);  return dv;  }  else  {  throw new Exception("Date entered is not a valid date.");  }  }  else  {  throw new Exception("No type was matched, failed to filter.");  }  }  catch (Exception ex)  {  throw new Exception("Error filtering data table. " + ex.Message);  }  }  //Returns a string array of all items from a data column  public static string[] RetrieveItemsFromDataCollumn(DataTable dt, string CollumnName)  {  string[] values = new string[dt.Rows.Count];  int temp = 0;  foreach (DataRow dr in dt.Rows)  {  values[temp] = dr[CollumnName].ToString();  temp++;  }  return values;  }  //method to quickly open a new form with the option to close the current one.  public static void OpenNewForm(Form CurrentFormName, Form NewFormName, bool CloseCurrent)  {  NewFormName.Show();  NewFormName.Location = CurrentFormName.Location;  if (CloseCurrent)  CurrentFormName.Close();  }  //Shutdown the application.  public static void ShutDownApplication()  {  Application.Exit();  }  }  } |
| DataAccess |  | using System;  using System.Collections.Generic;  using System.Data;  using System.Data.SqlClient;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Mitchell\_School\_of\_Music  {  static class DataAccess  {  //Create data adapters  public static SqlDataAdapter daStudent;  public static SqlDataAdapter daTutor;  public static SqlDataAdapter daRoom;  public static SqlDataAdapter daExamEntry;  public static SqlDataAdapter daExternalExam;  public static SqlDataAdapter daPayment;  public static SqlDataAdapter daBlockBooking;  public static SqlDataAdapter daTuitionChoice;  public static SqlDataAdapter daTimetabledLesson;  public static SqlDataAdapter daTutorTakes;  //Create sql command builders  public static SqlCommandBuilder cmbStudent;  public static SqlCommandBuilder cmbTutor;  public static SqlCommandBuilder cmbRoom;  public static SqlCommandBuilder cmbExamEntry;  public static SqlCommandBuilder cmbExternalExam;  public static SqlCommandBuilder cmbPayment;  public static SqlCommandBuilder cmbBlockBooking;  public static SqlCommandBuilder cmbTuitionChoice;  public static SqlCommandBuilder cmbTimetabledLesson;  public static SqlCommandBuilder cmbTutorTakes;  //Create dataset  public static DataSet ds = new DataSet();  //Create data tables  public static DataTable dtStudent = new DataTable();  public static DataTable dtTutor = new DataTable();  public static DataTable dtRoom = new DataTable();  public static DataTable dtExamEntry = new DataTable();  public static DataTable dtExternalExam = new DataTable();  public static DataTable dtPayment = new DataTable();  public static DataTable dtBlockBooking = new DataTable();  public static DataTable dtTuitionChoice = new DataTable();  public static DataTable dtTimetabledLesson = new DataTable();  public static DataTable dtTutorTakes = new DataTable();  //Connection string to database  public static string connectionString = @"Data Source = LOCALHOST; Initial Catalog=Music1; Integrated Security= true";  //Load database student data  public static void LoadDatabaseStudentData()  {  try  {  string sqlQuery = "select \* from Student";  daStudent = new SqlDataAdapter(sqlQuery, connectionString);  cmbStudent = new SqlCommandBuilder(daStudent);  daStudent.FillSchema(ds, SchemaType.Source, "Student");  daStudent.Fill(ds, "Student");  dtStudent = ds.Tables["Student"];  }  catch (Exception ex)  {  throw new SQLFailureException("There was a critical failure while attempting to load data from the student table: " + ex.Message);  }  }  //Load database Tutor data  public static void LoadDatabaseTutorData()  {  try  {  string sqlQuery = "select \* from Tutor";  daTutor = new SqlDataAdapter(sqlQuery, connectionString);  cmbTutor = new SqlCommandBuilder(daTutor);  daTutor.FillSchema(ds, SchemaType.Source, "Tutor");  daTutor.Fill(ds, "Tutor");  dtTutor = ds.Tables["Tutor"];  }  catch (Exception ex)  {  throw new SQLFailureException("There was a critical failure while attempting to load data from the Tutor table: " + ex.Message);  }  }  //Load database Room data  public static void LoadDatabaseRoomData()  {  try  {  string sqlQuery = "select \* from Room";  daRoom = new SqlDataAdapter(sqlQuery, connectionString);  cmbRoom = new SqlCommandBuilder(daRoom);  daRoom.FillSchema(ds, SchemaType.Source, "Room");  daRoom.Fill(ds, "Room");  dtRoom = ds.Tables["Room"];  }  catch (Exception ex)  {  throw new SQLFailureException("There was a critical failure while attempting to load data from the Room table: " + ex.Message);  }  }  //Load database exam entry data  public static void LoadDatabaseExamEntryData()  {  try  {  string sqlQuery = "select \* from ExamEntry";  daExamEntry = new SqlDataAdapter(sqlQuery, connectionString);  cmbExamEntry = new SqlCommandBuilder(daExamEntry);  daExamEntry.FillSchema(ds, SchemaType.Source, "ExamEntry");  daExamEntry.Fill(ds, "ExamEntry");  dtExamEntry = ds.Tables["ExamEntry"];  }  catch (Exception ex)  {  throw new SQLFailureException("There was a critical failure while attempting to load data from the Exam Entry table: " + ex.Message);  }  }  //Load database external exam data  public static void LoadDatabaseExternalExamData()  {  try  {  string sqlQuery = "select \* from ExternalExam";  daExternalExam = new SqlDataAdapter(sqlQuery, connectionString);  cmbExternalExam = new SqlCommandBuilder(daExternalExam);  daExternalExam.FillSchema(ds, SchemaType.Source, "ExternalExam");  daExternalExam.Fill(ds, "ExternalExam");  dtExternalExam = ds.Tables["ExternalExam"];  }  catch (Exception ex)  {  throw new SQLFailureException("There was a critical failure while attempting to load data from the External Exams table: " + ex.Message);  }  }  //Load database payment data  public static void LoadDatabasePaymentData()  {  try  {  string sqlQuery = "select \* from Payment";  daPayment = new SqlDataAdapter(sqlQuery, connectionString);  cmbPayment = new SqlCommandBuilder(daPayment);  daPayment.FillSchema(ds, SchemaType.Source, "Payment");  daPayment.Fill(ds, "Payment");  dtPayment = ds.Tables["Payment"];  }  catch (Exception ex)  {  throw new SQLFailureException("There was a critical failure while attempting to load data from the Payments table: " + ex.Message);  }  }  //Load database Block booking data  public static void LoadDatabaseBlockBookingData()  {  try  {  string sqlQuery = "select \* from BlockBooking";  daBlockBooking = new SqlDataAdapter(sqlQuery, connectionString);  cmbBlockBooking = new SqlCommandBuilder(daBlockBooking);  daBlockBooking.FillSchema(ds, SchemaType.Source, "BlockBooking");  daBlockBooking.Fill(ds, "BlockBooking");  dtBlockBooking = ds.Tables["BlockBooking"];  }  catch (Exception ex)  {  throw new SQLFailureException("There was a critical failure while attempting to load data from the Block Booking table: " + ex.Message);  }  }  //Load database Tuition choice data  public static void LoadDatabaseTuitionChoiceData()  {  try  {  string sqlQuery = "select \* from TuitionChoice";  daTuitionChoice = new SqlDataAdapter(sqlQuery, connectionString);  cmbTuitionChoice = new SqlCommandBuilder(daTuitionChoice);  daTuitionChoice.FillSchema(ds, SchemaType.Source, "TuitionChoice");  daTuitionChoice.Fill(ds, "TuitionChoice");  dtTuitionChoice = ds.Tables["TuitionChoice"];  }  catch (Exception ex)  {  throw new SQLFailureException("There was a critical failure while attempting to load data from the tuition choice table: " + ex.Message);  }  }  //Lod database timetabled data  public static void LoadDatabaseTimetabledLessonData()  {  try  {  string sqlQuery = "select \* from TimetabledLesson";  daTimetabledLesson = new SqlDataAdapter(sqlQuery, connectionString);  cmbTimetabledLesson = new SqlCommandBuilder(daTimetabledLesson);  daTimetabledLesson.FillSchema(ds, SchemaType.Source, "TimetabledLesson");  daTimetabledLesson.Fill(ds, "TimetabledLesson");  dtTimetabledLesson = ds.Tables["TimetabledLesson"];  }  catch (Exception ex)  {  throw new SQLFailureException("There was a critical failure while attempting to load data from the timetabled lessons table: " + ex.Message);  }  }  //Load database tutor takes data  public static void LoadDatabaseTutorTakesData()  {  try  {  string sqlQuery = "select \* from TutorTakes";  daTutorTakes = new SqlDataAdapter(sqlQuery, connectionString);  cmbTutorTakes = new SqlCommandBuilder(daTutorTakes);  daTutorTakes.FillSchema(ds, SchemaType.Source, "TutorTakes");  daTutorTakes.Fill(ds, "TutorTakes");  dtTutorTakes = ds.Tables["TutorTakes"];  }  catch (Exception ex)  {  throw new SQLFailureException("There was a critical failure while attempting to load data from the tutor takes table: " + ex.Message);  }  }  }  } |
| CellData |  | using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows.Forms;  namespace Mitchell\_School\_of\_Music  {  class CellData  {  private int roomNo;  private byte periodNo;  private DataRow associatedDataRow;  public CellData(int RoomNo, byte PeriodNo)  {  roomNo = RoomNo;  periodNo = PeriodNo;  }  public int RoomNo  {  get { return roomNo; }  }  public byte PeriodNo  {  get { return periodNo; }  }  public DataRow AssociatedDataRow  {  get { return associatedDataRow; }  }  public bool FindAssociatedDataRow(DateTime Date)  {  DataRow[] FoundDataRows = DataAccess.dtTimetabledLesson.Select("RoomNo = " + roomNo + " AND TimeLesson = " + periodNo + " AND DateLesson = '" + Date.Date.ToString() + "'");  if (FoundDataRows.Count() == 1)  {  associatedDataRow = FoundDataRows[0];  return true;  }  else  {  return false;  }  }    public override string ToString()  {  if (associatedDataRow == null)  {  return "RoomNo: " + roomNo + ",\r\nPeriodNo: " + periodNo + ",\r\nThis session is not booked, click here to book a session for this day and time.";  }  else if (((DateTime)associatedDataRow["DateLesson"]).Date == DateTime.Now.Date)  {  string ReturnString = string.Empty;  for (int CollumnNo = 0; CollumnNo < associatedDataRow.ItemArray.Count(); CollumnNo++)  {  ReturnString += associatedDataRow.Table.Columns[CollumnNo].ColumnName + ": " + associatedDataRow[CollumnNo].ToString() + "\r\n";  }  return ReturnString;  }  else  {  string ReturnString = string.Empty;  for (int CollumnNo = 0; CollumnNo < associatedDataRow.ItemArray.Count()-3; CollumnNo++)  {  ReturnString += associatedDataRow.Table.Columns[CollumnNo].ColumnName + ": " + associatedDataRow[CollumnNo].ToString() + "\r\n";  }  return ReturnString;  }  }  }  } |

## Testing

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test plan for: Menu | | | Tester: Developer | | | Test date: 27/02/2017 | |
| TestNo: | Test description: | Test data: | Expected outcome: | Pass/Fail  (P/F) | Corrective action: | Screen shot No: | Functional Objective: |
| 1 | Checking if the “Exit” button works. | Click | Application closes without Error code. | P | N/A | 1 | 1.4 |
| 2 | Checking if the “Students” button works. | Click | Open Students form. | P | N/A | 2 | 1.3 |
| 3 | Checking if the “Tutors” button works. | Click | Open Tutors form. | P | N/A | 3 | 1.3 |
| 4 | Checking if the “Rooms” button works. | Click | Open Rooms form. | P | N/A | 4 | 1.3 |
| 5 | Checking if the “Exam Entries” button works. | Click | Open Exam Entries form. | P | N/A | 5 | 1.3 |
| 6 | Checking if the “Exams” button works. | Click | Open Exams form. | P | N/A | 6 | 1.3 |
| 7 | Checking if the “Past payments” button works. | Click | Open Past payments form. | P | N/A | 7 | 1.3 |
| 8 | Checking if the “Bookings” button works. | Click | Open Bookings form. | P | N/A | 8 | 1.3 |
| 9 | Checking if the “Timetabled Lessons” button works. | Click | Open Time Tabled Lessons form. | P | N/A | 9 | 1.3 |
| 10 | Checking if the “Lesson Timetable” button works. | Click | Open Lesson Timetable form. | P | N/A | 10 | 1.3 |
| 11 | Checking if the “Students Report” button works. | Click | Open Students Report form. | P | N/A | 11 | 1.3 |
| 12 | Checking if the “Tutors Report” button works. | Click | Open Tutors report form. | P | N/A | 12 | 1.3 |

|  |  |
| --- | --- |
| Image No | Image |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |
| 11 |  |
| 12 |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test plan for: Students | | | Tester: Developer | | | Test date: 27/02/2017 | |
| TestNo: | Test description: | Test data: | Expected outcome: | Pass/Fail  (P/F) | Corrective action: | Screen shot No: | Functional Objective: |
| 1 | Retrieving data row information into the form. | Click | Data is loaded from data grid into form fields. | P | N/A | 1 | 2.8 |
| 2 | Testing first name with correct specifications | “Geoff” | Data is added as data row in database. | P | N/A | 2 | 2.1 |
| 3 | Testing surname with correct specifications | “Testler” | Data is added as data row in database. | P | N/A | 2 | 2.1 |
| 4 | Testing address with correct specifications | “42, FBar Road” | Data is added as data row in database. | P | N/A | 2 | 2.1 |
| 5 | Testing town with correct specifications | “Testville” | Data is added as data row in database. | P | N/A | 2 | 2.1 |
| 6 | Testing postcode with correct specifications | “BT307DJ” | Data is added as data row in database. | P | N/A | 2 | 2.1 |
| 7 | Testing email with correct specifications | “geoff\_testler@  hotmail.co.uk” | Data is added as data row in database. | P | N/A | 2 | 2.1 |
| 8 | Testing date of birth with correct specifications | “01/01/1998” | Data is added as data row in database. | P | N/A | 2 | 2.1 |
| 9 | Testing gender with correct specifications | “Male” | Data is added as data row in database. | P | N/A | 2 | 2.1 |
| 10 | Testing disability with correct specifications | Unchecked | Data is added as data row in database. | P | N/A | 2 | 2.1 |
| 11 | Testing telephone, no with correct specifications | “+440284459123” | Data is added as data row in database. | P | N/A | 2 | 2.1 |
| 12 | Testing parent name with correct specifications | “Greg Testler” | Data is added as data row in database. | P | N/A | 2 | 2.1 |
| 13 | Testing current student with correct specifications | Unchecked | Data is added as data row in database. | P | N/A | 2 | 2.1 |
| 14 | Testing first name with incorrect specifications | “txtxtxtxtxtxtxtxtxtxtxtxtxtxtx” (31 characters) | Data is added as data row in database. | P | N/A | 3 | 2.1 |
| 15 | Testing surname with incorrect specifications | “txtxtxtxtxtxtxtxtxtxtxtxtxtxtx” (31 characters) | Data is added as data row in database. | P | N/A | 4 | 2.1 |
| 16 | Testing address with incorrect specifications | “txtxtxtxtxtxtxtxtxtxtxtxtxtxtx” (31 characters) | Data is added as data row in database. | P | N/A | 5 | 2.1 |
| 17 | Testing town with incorrect specifications | “txtxtxtxtxtxtxtxtxtxtxtxtxtxtx” (31 characters) | Data is added as data row in database. | P | N/A | 6 | 2.1 |
| 18 | Testing postcode with incorrect specifications | “BT30000” | Data is added as data row in database. | P | N/A | 7 | 2.1 |
| 19 | Testing email with incorrect specifications | “txtxtxtxtxtxtxtxtxtxtxtxtxtxtx txtxtxtxtxtxtxtxtxtxt” (51 characters) | Data is added as data row in database. | P | N/A | 8 | 2.1 |
| 20 | Testing date of birth with incorrect specifications | “01/04/2017”  (Next month) | Data is added as data row in database. | P | N/A | 9 | 2.1 |
| 21 | Testing parent name with incorrect specifications | “txtxtxtxtxtxtxtxtxtxtxtxtxtxtx txtxtxtxtxtxtxtxtxtxtxtxtxtxtxt” (61 characters) | Data is added as data row in database. | P | N/A | 10 | 2.1 |
| 22 | Testing surname with unexpected specifications | “txtxtxtxtxtxtxtxtxtxtxtxtxtxtx” (30 characters) | Error provider message should appear and data should not be added to data row. | P | N/A | 11 | 2.1 |
| 23 | Testing postcode with unexpected specifications | “-------” | Error provider message should appear and data should not be added to data row. | P | N/A | 12 | 2.1 |
| 24 | Testing date of birth with unexpected specifications | “01/04/2017”  (Next month) | Error provider message should appear and data should not be added to data row. | P | N/A | 13 | 2.1 |
| 25 | Pressing the update button to change a single value within a data row | Adding an ‘a’ to the end of a first name. | Data row should be changed to reflect the new first name. | P | N/A | 14 | 2.8 |
| 26 | Pressing delete with a row selected. | Create then delete duplicate row. | Data row should be deleted | P | N/A | 15 | 2.3 |

|  |  |
| --- | --- |
| Image No | Image |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |
| 11 |  |
| 12 |  |
| 13 |  |
| 14 |  |
| 15 |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test plan for: Tutor | | | Tester: Developer | | | Test date: 27/02/2017 | |
| TestNo: | Test description: | Test data: | Expected outcome: | Pass/Fail  (P/F) | Corrective action: | Screen shot No: | Functional Objective: |
| 1 | Retrieving data row information into the form. | Click | Data is loaded from data grid into form fields. | P | N/A | 1 | 2.8 |
| 2 | Testing first name with correct specifications | “Geoff” | Data is added as data row in database. | P | N/A | 2 | 2.1 |
| 3 | Testing surname with correct specifications | “Testler” | Data is added as data row in database. | P | N/A | 2 | 2.1 |
| 4 | Testing address with correct specifications | “42, FBar Road” | Data is added as data row in database. | P | N/A | 2 | 2.1 |
| 5 | Testing town with correct specifications | “Testville” | Data is added as data row in database. | P | N/A | 2 | 2.1 |
| 6 | Testing postcode with correct specifications | “BT307DJ” | Data is added as data row in database. | P | N/A | 2 | 2.1 |
| 7 | Testing email with correct specifications | “geoff\_testler@  hotmail.co.uk” | Data is added as data row in database. | P | N/A | 2 | 2.1 |
| 8 | Testing date of birth with correct specifications | “01/01/1998” | Data is added as data row in database. | P | N/A | 2 | 2.1 |
| 9 | Testing gender with correct specifications | “Male” | Data is added as data row in database. | P | N/A | 2 | 3.1 |
| 10 | Testing disability with correct specifications | Unchecked | Data is added as data row in database. | P | N/A | 2 | 3.1 |
| 11 | Testing telephone, no with correct specifications | “+440284459123” | Data is added as data row in database. | P | N/A | 2 | 3.1 |
| 12 | Testing next of kin name with correct specifications | “Greg Testler” | Data is added as data row in database. | P | N/A | 2 | 3.1 |
| 13 | Testing current student with correct specifications | Unchecked | Data is added as data row in database. | P | N/A | 2 | 3.1 |
| 14 | Testing first name with incorrect specifications | “txtxtxtxtxtxtxtxtxtxtxtxtxtxtx” (31 characters) | Error provider message should appear and data should not be added to data row. | P | N/A | 3 | 3.1 |
| 15 | Testing surname with incorrect specifications | “txtxtxtxtxtxtxtxtxtxtxtxtxtxtx” (31 characters) | Error provider message should appear and data should not be added to data row. | P | N/A | 4 | 3.1 |
| 16 | Testing address with incorrect specifications | “txtxtxtxtxtxtxtxtxtxtxtxtxtxtx” (31 characters) | Error provider message should appear and data should not be added to data row. | P | N/A | 5 | 3.1 |
| 17 | Testing town with incorrect specifications | “txtxtxtxtxtxtxtxtxtxtxtxtxtxtx” (31 characters) | Error provider message should appear and data should not be added to data row. | P | N/A | 6 | 3.1 |
| 18 | Testing postcode with incorrect specifications | “BT30000” | Error provider message should appear and data should not be added to data row. | P | N/A | 7 | 3.1 |
| 19 | Testing email with incorrect specifications | “txtxtxtxtxtxtxtxtxtxtxtxtxtxtx txtxtxtxtxtxtxtxtxtxt” (51 characters) | Error provider message should appear and data should not be added to data row. | P | N/A | 8 | 3.1 |
| 20 | Testing date of birth with incorrect specifications | “01/04/2017”  (Next month) | Error provider message should appear and data should not be added to data row. | P | N/A | 9 | 3.1 |
| 21 | Testing next of kin name with incorrect specifications | “txtxtxtxtxtxtxtxtxtxtxtxtxtxtx txtxtxtxtxtxtxtxtxtxtxtxtxtxtxt” (61 characters) | Error provider message should appear and data should not be added to data row. | P | N/A |  | 3.1 |
| 22 | Testing surname with unexpected specifications | “txtxtxtxtxtxtxtxtxtxtxtxtxtxtx” (30 characters) | Error provider message should appear and data should not be added to data row. | P | N/A | 3 | 3.1 |
| 23 | Testing postcode with unexpected specifications | “-------” | Error provider message should appear and data should not be added to data row. | P | N/A | 3 | 3.1 |
| 24 | Testing date of birth with unexpected specifications | “01/04/2017”  (Next month) | Error provider message should appear and data should not be added to data row. | P | N/A | 3 | 3.1 |
| 25 | Pressing the update button to change a single value within a data row | Adding an ‘a’ to the end of a first name. | Data row should be changed to reflect the new first name. | P | N/A |  | 3.8 |
| 26 | Pressing delete with a row selected. | Create then delete duplicate row. | Data row should be deleted | P | N/A |  | 3.3 |

|  |  |
| --- | --- |
| Image No | Image |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |
| 11 |  |
| 12 |  |
| 13 |  |
| 14 |  |
| 15 |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test plan for: External exams | | | Tester: Developer | | | Test date: 27/02/2017 | |
| TestNo: | Test description: | Test data: | Expected outcome: | Pass/Fail  (P/F) | Corrective action: | Screen shot No: | Functional Objective: |
| 1 | Retrieving data row information into the form. | Click | Data is loaded from data grid into form fields. | P | N/A | 1 | 5.8 |
| 2 | Pressing the update button to change a single value within a data row | Adding an ‘a’ to the end of a field. | Data row should be changed to reflect the new first name. | P | N/A | 2 | 5.8 |
| 3 | Pressing delete with a row selected. | Create then delete duplicate row. | Data row should be deleted | P | N/A | 3 | 5.3 |
| 4 | Testing exam board with correct specifications | “Trinity College London” | Data is added as data row in database. | P | N/A | 4 | 5.1 |
| 5 | Testing grade with correct specifications | “D” | Data is added as data row in database. | P | N/A | 4 | 5.1 |
| 6 | Testing Exam title with correct specifications | “Flute” | Data is added as data row in database. | P | N/A | 4 | 5.1 |
| 7 | Testing exam date with correct specifications | “28/01/2018” | Data is added as data row in database. | P | N/A | 4 | 5.1 |
| 8 | Testing exam time with correct specifications | 10:00 | Data is added as data row in database. | P | N/A | 4 | 5.1 |
| 9 | Testing exam fee with correct specifications | “150” | Data is added as data row in database. | P | N/A | 4 | 5.1 |
| 10 | Testing accepting entries with correct specifications | Checked | Data is added as data row in database. | P | N/A | 4 | 5.1 |
| 11 | Testing exam board with incorrect specifications | “------------------------------------“ | Error provider message should appear and data should not be added to data row. | P | N/A | 5 | 5.1 |
| 12 | Testing Exam title with incorrect specifications | “---------------------------------------“ | Error provider message should appear and data should not be added to data row. | P | N/A | 5 | 5.1 |
| 13 | Testing exam date with incorrect specifications | “99/99/9999” | Error provider message should appear and data should not be added to data row. | P | N/A | 5 | 5.1 |
| 14 | Testing exam time with incorrect specifications | 99:99 | Error provider message should appear and data should not be added to data row. | P | N/A | 5 | 5.1 |
| 15 | Testing exam fee with incorrect specifications | 999999999999 | Error provider message should appear and data should not be added to data row. | P | N/A | 5 | 5.1 |

|  |  |
| --- | --- |
| Image No | Image |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test plan for: Exam entries | | | Tester: Developer | | | Test date: 27/02/2017 | |
| TestNo: | Test description: | Test data: | Expected outcome: | Pass/Fail  (P/F) | Corrective action: | Screen shot No: | Functional Objective: |
| 1 | Retrieving data row information into the form. | Click | Data is loaded from data grid into form fields. | P | N/A | 1 | 4.8 |
| 2 | Pressing the update button to change a single value within a data row | Changing the student number. | Data row should be changed to reflect the new data. | P | N/A | 2 | 4.8 |
| 3 | Pressing delete with a row selected. | Create then delete duplicate row. | Data row should be deleted | P | N/A | 3 | 4.3 |
| 4 | Testing exam no with correct specifications | A number from the combo box. | Data is added as data row in database. | P | N/A | 4 | 4.1 |
| 5 | Testing student no with correct specifications | A number from the combo box. | Data is added as data row in database. | P | N/A | 4 | 4.1 |
| 6 | Testing paid with correct specifications | Checked. | Data is added as data row in database. | P | N/A | 4 | 4.1 |
| 7 | Testing date paid with correct specifications | “06/04/2017” | Data is added as data row in database. | P | N/A | 4 | 4.1 |
| 11 | Testing date paid with incorrect specifications | “99/99/9999“ | Error provider message should appear and data should not be added to data row. | P | N/A | 5 | 4.1 |
| 12 | Clicking pay on a form that has not been paid sends user to payment form. | Click | User is sent to payment form. | P | N/A | 6 | 4.11 |

|  |  |
| --- | --- |
| Image no | Image |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test plan for: block booking | | | Tester: Developer | | | Test date: 27/02/2017 | |
| TestNo: | Test description: | Test data: | Expected outcome: | Pass/Fail  (P/F) | Corrective action: | Screen shot No: | Functional Objective: |
| 1 | Retrieving data row information into the form. | Click | Data is loaded from data grid into form fields. | P | N/A | 1 | 5.8 |
| 2 | Pressing the update button to change a single value within a data row | Changing the student number. | Data row should be changed to reflect the new data. | P | N/A | 2 | 5.8 |
| 3 | Pressing delete with a row selected. | Create then delete duplicate row. | Data row should be deleted | P | N/A | 3 | 5.3 |
| 4 | Testing student no with correct specifications | A number from the combo box. | Data is added as data row in database. | P | N/A | 4 | 5.1 |
| 5 | Testing number of lessons no with correct specifications | A number from the combo box. | Data is added as data row in database. | P | N/A | 4 | 5.1 |
| 6 | Testing ability level with correct specifications | An item from the combo box. | Data is added as data row in database. | P | N/A | 4 | 5.1 |
| 7 | Testing tuition choice with correct specifications | An item from the combo box. | Data is added as data row in database. | P | N/A | 4 | 5.1 |
| 11 | Testing number of lessons with incorrect specifications | “9999999999999999“ | Error provider message should appear and data should not be added to data row. | P | N/A | 5 | 5.1 |
| 12 | Clicking pay on a form that has not been paid sends user to payment form. | Click | User is sent to payment form. | P | N/A | 6 | 5.11 |

|  |  |
| --- | --- |
| Image no | Image |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test plan for: student report | | | Tester: Developer | | | Test date: 27/02/2017 | |
| TestNo: | Test description: | Test data: | Expected outcome: | Pass/Fail  (P/F) | Corrective action: | Screen shot No: | Functional Objective: |
| 1 | Data is displayed when form loads | Click | Data is displayed to user. | P | N/A | 1 | 13.1 |
| 2 | The user can select queries from a generated list of potential queries | Click | Combo box of queries. | P | N/A | 2 | 13.2 |
| 3 | The user can add a query and the table will display matching results. | Click | Table is queried. | P | N/A | 3 | 13.2 |
| 4 | The combo box will be populated with potential queries. | Click | Combo box is full. | P | N/A | 4 | 3.2 |
| 5 | Pressing the clear button resets the report to displaying all records. | Click | Report is reset to default. | P | N/A | 5 | 13.3 |

|  |  |
| --- | --- |
| Image no | Image |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test plan for: Tutor report | | | Tester: Developer | | | Test date: 27/02/2017 | |
| TestNo: | Test description: | Test data: | Expected outcome: | Pass/Fail  (P/F) | Corrective action: | Screen shot No: | Functional Objective: |
| 1 | Data is displayed when form loads | Click | Data is displayed to user. | P | N/A | 1 | 13.1 |
| 2 | The user can select queries from a generated list of potential queries | Click | Combo box of queries. | P | N/A | 2 | 13.2 |
| 3 | The user can add a query and the table will display matching results. | Click | Table is queried. | P | N/A | 3 | 13.2 |
| 4 | The combo box will be populated with potential queries. | Click | Combo box is full. | P | N/A | 4 | 13.2 |
| 5 | Pressing the clear button resets the report to displaying all records. | Click | Report is reset to default. | P | N/A | 5 | 13.3 |

|  |  |
| --- | --- |
| Image no | Image |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test plan for: booking report | | | Tester: Developer | | | Test date: 27/02/2017 | |
| TestNo: | Test description: | Test data: | Expected outcome: | Pass/Fail  (P/F) | Corrective action: | Screen shot No: | Functional Objective: |
| 1 | Data is displayed when form loads | Click | Data is displayed to user. | P | N/A | 1 | 13.1 |
| 2 | The user can select queries from a generated list of potential queries | Click | Combo box of queries. | P | N/A | 2 | 13.2 |
| 3 | The user can add a query and the table will display matching results. | Click | Table is queried. | P | N/A | 3 | 13.2 |
| 4 | The combo box will be populated with potential queries. | Click | Combo box is full. | P | N/A | 4 | 13.2 |
| 5 | Pressing the clear button resets the report to displaying all records. | Click | Report is reset to default. | P | N/A | 5 | 13.3 |

|  |  |
| --- | --- |
| Image no | Image |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test plan for: payment report | | | Tester: Developer | | | Test date: 27/02/2017 | |
| TestNo: | Test description: | Test data: | Expected outcome: | Pass/Fail  (P/F) | Corrective action: | Screen shot No: | Functional Objective: |
| 1 | Data is displayed when form loads | Click | Data is displayed to user. | P | N/A | 1 | 13.1 |
| 2 | The user can select queries from a generated list of potential queries | Click | Combo box of queries. | P | N/A | 2 | 13.2 |
| 3 | The user can add a query and the table will display matching results. | Click | Table is queried. | P | N/A | 3 | 13.2 |
| 4 | The combo box will be populated with potential queries. | Click | Combo box is full. | P | N/A | 4 | 13.2 |
| 5 | Pressing the clear button resets the report to displaying all records. | Click | Report is reset to default. | P | N/A | 5 | 13.3 |

|  |  |
| --- | --- |
| Image no | Image |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |

# Evaluation

## Evaluation of functional and non-functional requirements

### Functional requirements

|  |  |
| --- | --- |
| 1. Menu |  |
| 1.1 | The main menu is the starting form of the application and appears in centre screen as planned. |
| 1.2 | The main menu is easily navigable and is capable of accessing the appropriate areas of the application. |
| 1.3 | The main menu is easily navigable and is capable of accessing the appropriate areas of the application and allows the user to exit the entire program when closed or the exit button is pressed. |
| 1.4 | Allows the user to exit the entire program when closed or the exit button is pressed. |
| 1. Student form |  |
| 2.1 | The form can add new records to the back-end database through the use of the form when all fields are entered correctly. |
| 2.2 | Created records are added to the back-end SQL database successfully when entered correctly and the create button is clicked. |
| 2.3 | The delete button will delete the selected record as requested by the user, deleting the record selected within the data grid view. |
| 2.4 | The primary key value will increment by one for each new record that is added to the SQL database starting from 1. |
| 2.5 | Combo boxes, check boxes and masked text boxes are used extensively to prevent incorrect data from being entered to the extent possible under the specified conditions. |
| 2.6 | The form displays to the user when entered data is incorrect, what was incorrect, also does not allow the data to be added to the back-end database. This was done through the use of error providers which appear next to the incorrect fields and inform the user of what was wrong. |
| 2.7 | The form displays all necessary records as required and specified by the user. |
| 2.8 | The user can select a record from the data grid view to modify and load its details into the form so as they can be amended and updated to the backend database. |
| 2.9 | The form is capable of sorting the data grid view based on the requested field in both ascending and descending order by clicking the field title at the top of the data grid view. |
| 2.10 | The form is capable of searching for data specified data requested by the user within a column requested by the user. |
| 1. Tutor form |  |
| 3.1 | The form can add new records to the back-end database through the use of the form when all fields are entered correctly. |
| 3.2 | Created records are added to the back-end SQL database successfully when entered correctly and the create button is clicked. |
| 3.3 | The delete button will delete the selected record as requested by the user, deleting the record selected within the data grid view. |
| 3.4 | The primary key value will increment by one for each new record that is added to the SQL database starting from 1. |
| 3.5 | Combo boxes, check boxes and masked text boxes are used extensively to prevent incorrect data from being entered to the extent possible under the specified conditions. |
| 3.6 | The form displays to the user when entered data is incorrect, what was incorrect, also does not allow the data to be added to the back-end database. This was done through the use of error providers which appear next to the incorrect fields and inform the user of what was wrong. |
| 3.7 | The form displays all necessary records as required and specified by the user. |
| 3.8 | The user can select a record from the data grid view to modify and load its details into the form so as they can be amended and updated to the backend database. |
| 3.9 | The form is capable of sorting the data grid view based on the requested field in both ascending and descending order by clicking the field title at the top of the data grid view. |
| 3.10 | The form is capable of searching for data specified data requested by the user within a column requested by the user. |
| 1. Exam entry form |  |
| 4.1 | The form can add new records to the back-end database through the use of the form when all fields are entered correctly. |
| 4.2 | Created records are added to the back-end SQL database successfully when entered correctly and the create button is clicked. |
| 4.3 | The delete button will delete the selected record as requested by the user, deleting the record selected within the data grid view. |
| 4.4 | The primary key value will increment by one for each new record that is added to the SQL database starting from 1. |
| 4.5 | Combo boxes, check boxes and masked text boxes are used extensively to prevent incorrect data from being entered to the extent possible under the specified conditions. |
| 4.6 | The form displays to the user when entered data is incorrect, what was incorrect, also does not allow the data to be added to the back-end database. This was done through the use of error providers which appear next to the incorrect fields and inform the user of what was wrong. |
| 4.7 | The form displays all necessary records as required and specified by the user. |
| 4.8 | The user can select a record from the data grid view to modify and load its details into the form so as they can be amended and updated to the backend database. |
| 4.9 | The form is capable of sorting the data grid view based on the requested field in both ascending and descending order by clicking the field title at the top of the data grid view. |
| 4.10 | The form is capable of searching for data specified data requested by the user within a column requested by the user. |
| 4.11 | The payment form is accessible, transferring the necessary data over to make a payment for a record, it will also return necessary details to this form and send the user back here once payment has been completed. |
| 1. External exam form |  |
| 5.1 | The form can add new records to the back-end database through the use of the form when all fields are entered correctly. |
| 5.2 | Created records are added to the back-end SQL database successfully when entered correctly and the create button is clicked. |
| 5.3 | The delete button will delete the selected record as requested by the user, deleting the record selected within the data grid view. |
| 5.4 | The primary key value will increment by one for each new record that is added to the SQL database starting from 1. |
| 5.5 | Combo boxes, check boxes and masked text boxes are used extensively to prevent incorrect data from being entered to the extent possible under the specified conditions. |
| 5.6 | The form displays to the user when entered data is incorrect, what was incorrect, also does not allow the data to be added to the back-end database. This was done through the use of error providers which appear next to the incorrect fields and inform the user of what was wrong. |
| 5.7 | The form displays all necessary records as required and specified by the user. |
| 5.8 | The user can select a record from the data grid view to modify and load its details into the form so as they can be amended and updated to the backend database. |
| 5.9 | The form is capable of sorting the data grid view based on the requested field in both ascending and descending order by clicking the field title at the top of the data grid view. |
| 5.10 | The form is capable of searching for data specified data requested by the user within a column requested by the user. |
| 1. Block booking form |  |
| 6.1 | The form can add new records to the back-end database through the use of the form when all fields are entered correctly. |
| 6.2 | Created records are added to the back-end SQL database successfully when entered correctly and the create button is clicked. |
| 6.3 | The delete button will delete the selected record as requested by the user, deleting the record selected within the data grid view. |
| 6.4 | The primary key value will increment by one for each new record that is added to the SQL database starting from 1. |
| 6.5 | Combo boxes, check boxes and masked text boxes are used extensively to prevent incorrect data from being entered to the extent possible under the specified conditions. |
| 6.6 | The form displays to the user when entered data is incorrect, what was incorrect, also does not allow the data to be added to the back-end database. This was done through the use of error providers which appear next to the incorrect fields and inform the user of what was wrong. |
| 6.7 | The form displays all necessary records as required and specified by the user. |
| 6.8 | The user can select a record from the data grid view to modify and load its details into the form so as they can be amended and updated to the backend database. |
| 6.9 | The form is capable of sorting the data grid view based on the requested field in both ascending and descending order by clicking the field title at the top of the data grid view. |
| 6.10 | The form is capable of searching for data specified data requested by the user within a column requested by the user. |
| 6.11 | The payment form is accessible, transferring the necessary data over to make a payment for a record, it will also return necessary details to this form and send the user back here once payment has been completed. |
| 1. Payment form |  |
| 7.1 | Created records are added to the back-end SQL database successfully when entered correctly. |
| 7.2 | Created records are added to the back-end SQL database successfully when entered correctly. |
| 7.3 | The form does allow the user to enter and choose their own payment method and payment details as needed, those necessary will as a result be saved within the record of the payments SQL table. |
| 7.4 | The form returns the user to the form that brought them here once the payment is complete or the payment has been cancelled. |
| 7.5 | The form provides necessary details as required to tell the user what they are paying for on the left side of the form. |
| 1. Payment history form |  |
| 8.1 | The form displays all necessary records as required and specified by the user. |
| 8.2 | The form is capable of searching for data specified data requested by the user within a column requested by the user. |
| 8.3 | The form is capable of sorting the data grid view based on the requested field in both ascending and descending order by clicking the field title at the top of the data grid view. |
| 1. Timetabled lessons form |  |
| 9.1 | The form can add new records to the back-end database through the use of the form when all fields are entered correctly. |
| 9.2 | Created records are added to the back-end SQL database successfully when entered correctly and the create button is clicked. |
| 9.3 | The delete button will delete the selected record as requested by the user, deleting the record selected within the data grid view. |
| 9.4 | The primary key value will increment by one for each new record that is added to the SQL database starting from 1. |
| 9.5 | Combo boxes, check boxes and masked text boxes are used extensively to prevent incorrect data from being entered to the extent possible under the specified conditions. |
| 9.6 | The form displays to the user when entered data is incorrect, what was incorrect, also does not allow the data to be added to the back-end database. This was done through the use of error providers which appear next to the incorrect fields and inform the user of what was wrong. |
| 9.7 | The form displays all necessary records as required and specified by the user. |
| 9.8 | The user can select a record from the data grid view to modify and load its details into the form so as they can be amended and updated to the backend database. |
| 9.9 | The form is capable of sorting the data grid view based on the requested field in both ascending and descending order by clicking the field title at the top of the data grid view. |
| 9.10 | The form is capable of searching for data specified data requested by the user within a column requested by the user. |
| 9.11 | The lessons can only be booked on dates, rooms, tutors and students that do not conflict with other such data. |
| 9.12 | The form only allows and auto-completes some data as is expected of the user so as to prevent mistakes from being made. |
| 1. Lesson timetable form |  |
| 10.1 | The form displays information on rooms and times on which lessons are or are not booked as well as rooms that are otherwise unavailable to be booked for classes. |
| 10.2 | The form allows the user to select a day through the use of a date-time picker through which they can select the desired day to see what classes are on for said day. |
| 10.3 | Hovering the mouse over cells reveals data about the cells respective class and provides a tooltip on the information contained within. |
| 10.4 | Clicking on an empty cell offers the user to be sent to the timetable lessons form to create a set of lessons starting from the selected date in that room at that time. |
| 10.5 | Clicking on an empty cell offers the user to be sent to the timetable lessons form to create a set of lessons starting from the selected date in that room at that time. |
| 10.6 | Cells are displayed with different colours respective of the data held within, red – room in use. Yellow – class. White – clear. |
| 1. Room form |  |
| 11.1 | The form can add new records to the back-end database through the use of the form when all fields are entered correctly. |
| 11.2 | Created records are added to the back-end SQL database successfully when entered correctly and the create button is clicked. |
| 11.3 | The delete button will delete the selected record as requested by the user, deleting the record selected within the data grid view. |
| 11.4 | The primary key value will increment by one for each new record that is added to the SQL database starting from 1. |
| 11.5 | Combo boxes, check boxes and masked text boxes are used extensively to prevent incorrect data from being entered to the extent possible under the specified conditions. |
| 11.6 | The form displays to the user when entered data is incorrect, what was incorrect, also does not allow the data to be added to the back-end database. This was done through the use of error providers which appear next to the incorrect fields and inform the user of what was wrong. |
| 11.7 | The form displays all necessary records as required and specified by the user. |
| 11.8 | The user can select a record from the data grid view to modify and load its details into the form so as they can be amended and updated to the backend database. |
| 11.9 | The form is capable of sorting the data grid view based on the requested field in both ascending and descending order by clicking the field title at the top of the data grid view. |
| 11.10 | The form is capable of searching for data specified data requested by the user within a column requested by the user. |
| 1. Tutor takes form |  |
| 12.1 | The form is accessible through the tutor form by pressing proficiencies when a tutor record is selected. |
| 12.2 | The proficiencies that are shown are those of the specified tutor as selected on the previous form. |
| 12.3 | Created records are added to the back-end SQL database successfully when entered correctly and the create button is clicked. |
| 12.4 | The form is capable of searching for data specified data requested by the user within a column requested by the user. |
| 12.5 | The form is capable of sorting the data grid view based on the requested field in both ascending and descending order by clicking the field title at the top of the data grid view. |
| 12.6 | The user can select a record from the data grid view to modify and load its details into the form so as they can be amended and updated to the backend database. |
| 12.7 | The delete button will delete the selected record as requested by the user, deleting the record selected within the data grid view. |
| 12.8 | The primary key value will increment by one for each new record that is added to the SQL database starting from 1. |
| 12.9 | Combo boxes, check boxes and masked text boxes are used extensively to prevent incorrect data from being entered to the extent possible under the specified conditions. |
| 1. Student reports |  |
| 13.1 | The form displays all necessary records to the user of the specified table and the details that go along with it. |
| 13.2 | The user is able to query the report with any column the user wishes that is part of the table and is provided with every potential option that could be queried for that option. |
| 13.3 | The user can clear the set query when it is no longer needed through the clicking of the reset button provided. |
| 1. Tutor reports |  |
| 14.1 | The form displays all necessary records to the user of the specified table and the details that go along with it. |
| 14.2 | The user is able to query the report with any column the user wishes that is part of the table and is provided with every potential option that could be queried for that option. |
| 14.3 | The user can clear the set query when it is no longer needed through the clicking of the reset button provided. |
| 1. Payment reports |  |
| 15.1 | The form displays all necessary records to the user of the specified table and the details that go along with it. |
| 15.2 | The user is able to query the report with any column the user wishes that is part of the table and is provided with every potential option that could be queried for that option. |
| 15.3 | The user can clear the set query when it is no longer needed through the clicking of the reset button provided. |
| 1. Block booking reports |  |
| 16.1 | The form displays all necessary records to the user of the specified table and the details that go along with it. |
| 16.2 | The user is able to query the report with any column the user wishes that is part of the table and is provided with every potential option that could be queried for that option. |
| 16.3 | The user can clear the set query when it is no longer needed through the clicking of the reset button provided. |

### Non-functional requirements

|  |  |
| --- | --- |
| 1. General |  |
| 1.1 | An SQL server is used to store data records as needed for the application. |
| 1. The form design |  |
| 2.1 | The form design is understandable, could be better, but is understandable and fairly easy to use. |
| 2.2 | The forms attempts to limit the input of false data as much as possible under the conditions of creation. |
| 2.3 | The forms are understandable, less so at first but over continued use can grow on the user quickly. |

## Evaluation of Technique

### Use of waterfall-DSDM

The two main methodologies used in the development of my application are waterfall and MoSCoW, making use of the sequential development of waterfall, going from one part to the next and finishing with testing and the development structure of MoSCoW in which the most important aspect of the application were completed first and foremost.

This was rather effective as I had a good idea in my mind where the most important elements of the project were and so they were completed one by one until they were all complete. Doing this allowed the application to feel functional above all else and perform to the standard expected, allowing it to operate in a way that was requested within the case study. However, as the use of MoSCoW would suggest, the application could have been much better if more time were available as some of the less important features of the application could have still been added (see future development). However, for what is there, the application is fully functional, these being the must have, should have and a few could have features that were intended for the application.

## Evaluation of Self Performance

### Time Management

#### General

I feel as though I have accomplished my primary goal of the completion of the piece of software, although it may be missing certain minor features because of the time constraint, I believe I have accomplished the must have and should have criteria of the MoSCoW aspect of DSDM. For the most part I was able to stick to my Gantt chart plan, sequentially completing areas of the project in order of necessity.

#### Program

The program was completed on time although unfortunately missing a few minor features and quality of life improvements. The completion of the program was a priority and I worked on the must-have features before moving into the should-have features of the program.

#### Documentation

My primary goal being the completion of the software first made that my top priority over should have aspects of documentation as it would be useless without a working program to alongside it. Thus, the testing, while complete and documented was done after the completion of the program and most of the remaining time after the completion of the program was dedicated to it.

### Planning

#### General

When planning, I found myself locking down what my program was going to be without considering how this would work effectively, upon development of the program, I quickly realised that the way I had planned to implement some of these things were not the ideal way of doing so, as such, I have drifted from the plan in a few areas of the application.

#### Program

The program does follow the plan for the most part however does differ in some places in which a better is discovered either in terms of efficiency or the user experience, usually both.

#### Documentation

The plans could be updated with the current design of the application in mind, however this would appear a futile effort as the updated designs would serve no purpose as the project is complete and as such wouldn’t so much as be a plan as it would be a sketch of something that is already complete.

### Skills developed

#### Programming

The project has improved my skills as a programmer in many ways, most notably, this project has introduced me to the SQL scripting language which is a language I had not experienced before the start of this project. I could see knowledge of this language as being quite useful in the future, particularly if I ever need to create a server, wireless or otherwise to store extensive amounts of data.

The project has also allowed me to test and make use of areas of the C# language I may not have been previously familiar with, particularly the versatility of the language and its ability to work with other applications, languages and resources. Some examples of this include the way in which the C# language can communicate, process and present data from SQL, a very useful tool for the development of wireless and local servers.

The Regex language is also something I had not come into contact with prior to the start of development of this application and while I do not know how to write it myself, receiving a basic understanding of how this language functions has made me curious to learn more as to its potential uses in my future projects.

The project has also inadvertently helped me develop my understanding of how resources, extensions and frameworks operate within the C#/visual studio environment as many issues cropped up, particularly when it came to the area of reports in which I needed to fix an issue in which they would not load correctly when switched between home and school computers, as well as providing insight of the usage of tools such as github which was an option I investigated in order to help solve the aforementioned report issue.

A significant area of improvement for me was the way in which I write my code in terms of efficiency both for me and the machine the code will be ran on. This was done in many different ways, largely by improving my understanding on previously known concepts such as switched and most importantly classes which was a topic before now, I had a fairly limited understanding of but this year, this understanding has grown greatly and I am now fairly confident in my abilities to use such a powerful element of C#.

#### Planning

From this project, I have examined how setting up solid plans for an application can be detrimental in the long-term functionality of the program as locking it down to one plan reduces the adaptability and prevents improvements to be made during development. As a result of this I have learned to adapt my future plans to accommodate for changes that I may decide to make in the future for the benefit of the program, even if it contradicts the plans laid out prior to the development.

### Problems encountered and how they were dealt with

* **ReportViewer WinForms and ReportViewer Common version 11 issue** – One of the most extensive issues discovered within my program was the issue of the ReportViewer component used in the displaying of reports of tables within the SQL database. The issue was that the required extensions, “Windows.ReportViewer.Winforms” and “Windows.ReportViewer.Common”, were not available on home PCs by default, the ideal way to fix this was to download the necessary tools, the SQL tools for visual studio, the easiest way of doing which was to modify the installation of Visual Studio 15 and add the tools. This seemed simple at first but quickly developed into a much greater problem as it turned out that the version of the ReportViewer on the school computers was version 11 and the one provided by the modification of VS15 was version 12, version 11 being the older, now outdated version of this tool, however, since it was the school computers on which our programs would be tested and running version 12 ReportViewer on them would lead to an immediate exception being thrown, a compromise would need to be made. Either I would need to start working solely on the school computers, massively reducing the development and testing time available for the application or find a better alternative. After searching through many sources on the internet and different extensions available for download and on github, nothing worked quite right and would not load on the school computers as there would always be something that wouldn’t work quite right and lead to an immediate crash.

In the end, after much trial and error, I loaded the extension with version 12 on my home computer and moved my program onto the school computer and put the ReportViewers into the application, my thought process behind which was the hope that these ReportViewer components were backward compatible with past versions of ReportViewer versions. This was the case and I was able to continue development on home computers using version 12 with the version 11 components within my application.

* **Unable to delete data rows that have been referenced in other areas of the application** – A rather late realisation occurred near the deadline of the project in which it was discovered that data rows could not be deleted if the data row was referenced by another data row in another table. The was realised surprisingly late simply by chance when testing the delete function on the student form, more surprising is the fact that this was not realised sooner. This issue ran through almost the entire application as quite a number of the tables were referenced by other tables, making a fix for this quite extensive. Some solutions to this issue came to mind, I could either; write a piece of code which would check all relating tables for references and delete any data rows which referenced this data row, however, there were 2 main issues with this plan, one; the code would be quite extensive and while I believe I could have done it, it also would have been incredibly time consuming. Another option was to add an additional field to each SQL table, a bit value which determines if the row has been deleted, making it so that no row is ever actually deleted, it is simply told not to appear if the deleted value is true. There was a similar issue with this solution however as further thought revealed that the entire application would need to have been went through to set it so as the deleted rows did not appear in any of the data grid views or combo boxes which would be populated by primary keys and most importantly, this would have taken an extreme amount of time with quite a high chance that something would be missed, so, although possible, this idea was also discounted due to the lateness of the issues discovery and how close I was getting to the deadline. The final solution was the most unfortunate but also the most doable under the time constraints, this was to simply have an error message appear if the user ever tried to delete a data row that was relied upon in another data row. The issue with this is the hit that it deals to the usability of the program as this would mean that the user would need to sequentially go through each form and delete every other occurrence of that piece of data. A very tedious effort, however necessary as the alternatives would take far too long to implement and even then, issues would likely remain, causing data that may not have been wished to be deleted to be deleted and leaving gaps of information which may have been important. Unfortunately, this left me with the previously mentioned safe option of the error message.

### Backup routine

I backed up my project with daily iterations whenever I worked on it, this was done as part of the process in which I would transfer my project between my laptop and home computer, a copy would also be frequently be left within the school system in case of extreme disaster. All files worked on in regards to the project were kept in the same folder, these were the project file, the documentation word document, the txt file in which I typed all SQL code, the publisher file in which all form designs and ER diagrams were done and the excel spreadsheet for the Gantt chart.

I maintain every backup made as the file sizes are very small and so can all be stored with minimal impact to hard drive space, around 30 iterations were made in total as there were many days in which the project would remain on a system for more than a few days, such as holidays and weekends. The increments of the project were saved with the name of the project followed by the designation, consisting of a 3-decimal point iteration, for example, 0.0.1 was the first iteration of the project, 0.0.2 being the next. Whenever a significant enough change was made, I would then increment the second decimal point or whenever the third decimal point hit 10 so as to prevent the system from getting too unnecessarily complicated.

### Testing

Testing was an incredibly tedious procedure which I found largely unnecessary to be documented as many of the things being tested were tested in the process of testing other things. None the less it was completed through performing three main tests on each input on each data input field, these being a valid, an invalid and an unexpected test, for example, for valid, entering the kind of data you would expect to have entered within that field, for invalid, some data that is out of range such as too many or too few characters and unexpected such as entering words into a field that expects a number. I also tested the creation capability of data rows when both incorrect data was entered and when correct data was entered, same for update.

Due to the tedious nature of testing, I found myself procrastinating doing it, finding much more worthwhile aspect of the project to work on in that time, as a result of this, the testing aspect did not follow the Gantt chart to plan, it ran slightly over its allotted time. If I were to redo this project under different circumstances however, I believe that I would do the testing completely differently, performing a more logical methodology to the way in which I ensure the program works the way it should, time spent testing, is time spent not improving the program, therefore appears to me as time wasted, while it is important to do so as to ensure all areas of the project are working as expected, testing things that will very obviously work is simply a waste of time and could be better spent improving the functionality of the program.

## Future development

### Overall

* **Basic improvements** – Some basic improvements could always be made to the functionality of the program in many areas, a few objects that could be a few more pixels either way, some code that could be more efficient etc. There is always room for improvements in just about any program.

### User experience improvements

* **The form design** ­– Admittedly, while the layout and appearance of the forms are functional and well enough organised to be operated by someone with a basic understanding of what they are doing, they are not ideal. There are a few areas which could potentially be much more visually appealing and laid out in a slightly more intuitive way to the everyday user. A minor improvement, but one that could make all the difference to someone’s day.

### Efficiency improvements

* **Efficient and clean storing of data tables, data adapters and command builders** – The data access class of the application appears messy and unintuitive, there are many variables all with similar names and while efficient enough, it could be better. I had a plan to create 3 arrays each with a size of the number of tables within the dataset. One array would be a data adapter array; another would be a data table array and the last would be a command builder array. Each would then be populated by the objects of the type the array was, this would leave 3 arrays of the different data types, a much cleaner solution than the loose variables that are lying around in the current version. However, this would leave each object unlabelled and would make accessing the appropriate data table for example, unintuitive at best, as such, through the use of enums, each instance of each object in each array could be enumerated to a value of a given name, for example, student could be the enum value of 0 with the enum name, student. This would allow the programmer to access the student data table for example as follows, {dtArray[Tables.Student]}, in this case, the array for data tables would be called dtArray and the enum would be called Tables and the enum for the student table would be Student, thus allowing easy and efficient access to each part of the data access class using only 3 arrays, not including the methods. This was however not done for the reason of it being rather theoretical and, while I did experiment with it, there was not enough time to implement it for fear of breaking something that would waste time fixing, also, it would not have affected the running efficiency of the program much, simply making the code tidier and easier to understand. Having all the data tables in an array would have also been useful for if I ever needed to search for something in all of the tables at the same time using a foreach loop.

### Feature improvements

* **Multiple search criteria when searching data grid views** – The idea of multiple search criteria was an area I experimented with in the design of the reports, however, the way in which the queries worked in reports was slightly more complicated than the way in which it is done for the data grid views, as each query combination would require its own query that would need designed. This was not an issue for data grid views however and I could have done it with relative ease, however, with time constraints, this was a could have feature that did not make it into the final product, the intention was to have a list box of all the active query criteria and the user could add and remove criteria as needed.
* **Multiple search criteria when searching data tables with the report viewer** – This was an idea which was actually the original plan for the project, however, quickly changed upon realising that each query combo needed its own query built explicitly for that purpose, overall making it a futile effort that was more trouble than it was worth. However, given enough time, it should have been possible to set up a few combo queries for at least a few of the possible and most likely combination of searches that the user may choose to perform.

### Additional features

* **A form in which the employees and students of the music school could check in and check out as they arrived or left the premises** – A from in which people arriving at the music school could check in and people leaving could check out, would allow for more effective organisation within the business, this system would operate simply, allowing a user to check their name off or onto a list, possibly with a record being left in the SQL database as to when people have checked in or out. There could also have been room to allow students or their tutors to check the student into or out of class, allowing for the lesson timetable to be updated with relevant information regarding whether or not the student turned up for class on that day.
* **A password system** – A password system for client/admin use, protecting certain vital areas of the system from tampering without correct authorization. Preventing students from messing with the internal workings of the school.
* **Instruments and instrument hire** – One system stated within the case study was the instruments the school had available, which could have been stored within the database and accessible through the application, as well as the feature of being able to rent said instruments out to students, requiring the storing of data relating to information about said instrument and its hiring status. This was not done due to time constraints.
* **More reports** – Only 4 reports exist within the application at present, all of which with the ability to query information. I would have liked to extend this number to all SQL tables so as each could have a report of their own, allowing them to be viewed and printed as requested by the user. Only 4 reports with queries were done however due to time constraints.

## Strengths and weaknesses of the project

### Strengths

* **Well coded** – If there is one thing I take pride in my work about, it is the quality of my code as I try and make it as efficient as possible, putting the efficiency and functionality of the project over everything else. As a result of this, the program is fast and functional to the extent the tools available to me would allow me to make it at my current skill level.
* **The lesson timetable and timetabled lessons forms** – This form is likely the most complex one in the application, making use of classes to store data about each individual cell as well as adapting and changing colour based on different conditions on that day, as well as changing based on the day of the week.

### Weaknesses

* **User interface design** – The look and feel of the forms may not be ideal for a new user as some things may not be immediately clear and effective usage of the tools available within the program may take some getting used to for new users.
* **Commenting on code** – as it is a necessary part of the course, it has been implemented in a way that may be adequate however not up to the standard expected of many people.

Personally I find the process of commenting on code before it is written to be a waste of time, while I do understand that it is important if working in a team, I know that is not what I am doing and am fully capable of reading what my code says simply by looking at it, I find the code to be easy enough to navigate without the use of comments and while I am aware it would help others read it, there is nobody else working on it but me, making it a redundant and inefficient process.

* **Under-Reliance on try-catches** – I have used try-catches extensively, however perhaps not to the extent expected. Try-catches are useful for ensuring that data entered does not lead to an exception being thrown within the application, however, my research has shown that overuse of them heavily slows the speed of an application, significantly increasing compile times and taking far longer to process catching exceptions than simply using an if statement to detect the problem if it went wrong. Relying on try-catches should really be a fall back for if things go very wrong rather than relying on them to the extent where even simple calculations in which there it next to an impossible chance of an error (there always being the slight possibility of one in any system at any time) require a try-catch to ensure they do not fail to the point where there is a higher chance the try-catch will fail than there is the code it contains will fail.

# References to resources used within project

## Code

The only code copied is that of the regular expressions used to validate the postcode and the telephone number.

### Post Code regular expression code

return

(

Regex.IsMatch(value, "(^[A-PR-UWYZa-pr-uwyz][0-9][ ]\*[0-9][ABD-HJLNP-UW-Zabd-hjlnp-uw-z]{2}$)") ||

Regex.IsMatch(value, "(^[A-PR-UWYZa-pr-uwyz][0-9][0-9][ ]\*[0-9][ABD-HJLNP-UW-Zabd-hjlnp-uw-z]{2}$)") ||

Regex.IsMatch(value, "(^[A-PR-UWYZa-pr-uwyz][A-HK-Ya-hk-y][0-9][ ]\*[0-9][ABD-HJLNP-UW-Zabd-hjlnp-uw-z]{2}$)") ||

Regex.IsMatch(value, "(^[A-PR-UWYZa-pr-uwyz][A-HK-Ya-hk-y][0-9][0-9][ ]\*[0-9][ABD-HJLNP-UW-Zabd-hjlnp-uw-z]{2}$)") ||

Regex.IsMatch(value, "(^[A-PR-UWYZa-pr-uwyz][0-9] [A-HJKS-UWa-hjks-uw][ ]\*[0-9][ABD-HJLNP-UW-Zabd-hjlnp-uw-z]{2}$)") ||

Regex.IsMatch(value, "(^[A-PR-UWYZa-pr-uwyz][A-HK-Ya-hk-y][0-9][A-Za-z][ ]\*[0-9][ABD-HJLNP-UW-Zabd-hjlnp-uw-z]{2}$)") ||

Regex.IsMatch(value, "(^[Gg][Ii][Rr][]\*0[Aa][Aa]$)")

);

**As found at:** <http://stackoverflow.com/questions/5820820/regular-expression-in-c-sharp-uk-postcode>

### Telephone number regular expression code

return Regex.IsMatch(value, @"^(((\+44\s?\d{4}|\(?0\d{4}\)?)\s?\d{3}\s?\d{3})|((\+44\s?\d{3}|\(?0\d{3}\)?)\s?\d{3}\s?\d{4})|((\+44\s?\d{2}|\(?0\d{2}\)?)\s?\d{4}\s?\d{4}))(\s?\#(\d{4}|\d{3}))?$");

**As found at:** <http://stackoverflow.com/questions/25155970/validating-uk-phone-number-regex-c>

## Additional sources

**All other code was my own, although some has been inspired by research from different sources, most notably:**

* Examples and demonstrations provided within class.
* Stack Overflow.
* Microsoft MSDN.