Academic Year 2016/2017

|  |
| --- |
| [Type the document subtitle] | Adam McNEILL |

|  |  |
| --- | --- |
| SSD A2 Coursework Unit | THE MITCHELL SCHOOL OF MUSIC |

Contents

[Project Plan 3](#_Toc479832932)

[Introduction 3](#_Toc479832933)

[Problem Identification and scope 3](#_Toc479832934)

[System Approach 3](#_Toc479832935)

[Rapid Application Development (RAD) 4](#_Toc479832936)

[Dynamic Systems Development Method (DSDM) 4](#_Toc479832937)

[Scrum 4](#_Toc479832938)

[Waterfall 5](#_Toc479832939)

[GANTT Chart 6](#_Toc479832940)

[Use-Case Diagram 7](#_Toc479832941)

[ER Diagram 8](#_Toc479832942)

[User Requirements 9](#_Toc479832943)

[Functional 9](#_Toc479832944)

[Non Functional 10](#_Toc479832945)

[Design 11](#_Toc479832946)

[Normalisation 11](#_Toc479832947)

[Table Designs 12](#_Toc479832948)

[Student Table 12](#_Toc479832949)

[Instrument Table 13](#_Toc479832950)

[Rental Table 14](#_Toc479832951)

[Payment Table 15](#_Toc479832952)

[Stockist Table 16](#_Toc479832953)

[Repairs Table 17](#_Toc479832954)

[Form Designs 18](#_Toc479832955)

[Loading Screen 18](file:///G:\SSD%20REPORT.docx#_Toc479832956)

[Log In 20](file:///G:\SSD%20REPORT.docx#_Toc479832957)

[Add Student 22](file:///G:\SSD%20REPORT.docx#_Toc479832958)

[Add Rental 24](file:///G:\SSD%20REPORT.docx#_Toc479832959)

[Add Repair 26](file:///G:\SSD%20REPORT.docx#_Toc479832960)

[View Students 28](file:///G:\SSD%20REPORT.docx#_Toc479832961)

[View Instruments 30](file:///G:\SSD%20REPORT.docx#_Toc479832962)

[Testing Design 33](#_Toc479832963)

[Design of Test Plan 33](#_Toc479832964)

[Implementation 34](#_Toc479832965)

[Testing 34](#_Toc479832966)

[Testing Evidence 36](#_Toc479832967)

[Evaluation 42](#_Toc479832968)

[Evaluation of project plan 42](#_Toc479832969)

[Evaluation of user requirements 44](#_Toc479832970)

[Non Functional 45](#_Toc479832971)

[Evaluation of approach to the solution 46](#_Toc479832972)

[Evaluation of solution 46](#_Toc479832973)

[Evaluation of own performance 46](#_Toc479832974)

[Appendix: 47](#_Toc479832975)

[SQL Code 47](#_Toc479832976)

# Project Plan

# Introduction

I have been asked develop a system to help improve the administrative capability and reliability of the Mitchell school of music. They are a small business who run professional individual lessons during term time, specialist summer courses and also accommodate for instrument rental. He has a specific number of classrooms, private rooms and a single hall for performances. There are a range of rates for different scenarios that must be catered to, and each student is guaranteed a certain amount of live performances in local venues, in general they function as a professional music tuition service but lack professional administration, which over time is proving more and more costly due to reimbursement following human error, reams of perishable human resources and an increased demand for staff. They have contacted me and requested that I work with them to develop a powerful administration system that would improve time efficiency, decrease need for paper resources and ensure a hassle free administration service void of as much human error as possible.

# Problem Identification and scope

Whilst the school face a large variety of problems, the section I’ll be approaching is their instrument rental section. Problems they currently face include instruments not being returned on time, loss of signed rental documents and a shortage on staff. When I develop my system, provided that correct information is initially included, It will be possible for automatic billing and later-rental letters to be sent as well as very efficient to look up and find any individual instrument and find out where it is and what condition it is in. The efficiency and versatility of the system should be such that less staff will be needed since the time and expertise required will be lessened.

# System Approach

In creating this system, I will have to choose a programming methodology to follow

There are many different methodologies to choose from all featuring pros and cons which I will need to asses in order to choose the methodology which suits this project best. The main methodologies I had to choose from included the following:

## Rapid Application Development (RAD)

As issues with waterfall were identified RAD was developed to try and combat the problems associated with it. In stark contrast to waterfalls sequential approach, which for a large application could be a long process focused on extensive requirement gathering and design, RAD sought to emphasise on development. As result of the minimised emphasis on requirements phases sometimes requirements can be poor. I don’t think this would be a good idea methodology

for the project because the requirements are rigid and must be catered to strictly, and hence more time must be devoted to them than RAD would generally accommodate.

## Dynamic Systems Development Method (DSDM)

DSDM is built upon RAD and sought to improve it by adding structure and discipline in the aim to increase software quality. DSDM is an iterative methodology. It allows users to see functionality at the end of an iteration and provide feedback. It uses the four points used for prioritization in RAD. It aims to take a less rigid approach to functionality keeping the time of the project fast and fixed and compromising the amount of functionality delivered if problems arise. Due to the nature of the project I will likely be unable to contact the Mitchel School of Music after each phase, so again this methodology wouldn’t be ideal.

## Scrum

Scrum is part of the more modern Agile methodology approach. Like RAD and DSDM, Scrum uses iteration to split the development of the overall solution. Taking its name from the scrum in a game of rugby where team members pack closely together to form a unit, it seeks to emphasise teamwork by focusing on communication and collaboration throughout a project. A key factor of Scrum is keeping flexibility for the likely event of requirements changing, in the Scrum manner of thinking requirements aren't set in stone from the outset. I however lack a team to lead as a scrum leader, hence this would be an impossible methodology for me to pursue. Extreme Programming (XP)

XP uses all points of the agile manifesto making it a pure agile methodology. XP focus on short iterations in bursts of development which quickly churn out demonstrable functionality. The iterations are typically shorter than that of Scrum iterations. Requirements are relaxed to a greater extent. Providing work hasn't started on a piece of functionality it's requirements are allowed to change at any time. Extreme programming iterations follow a strict order of prioritisation for pieces of functionality which is defined by the customer’s priorities. The client however has a rigid set of unchanging user requirements, and likewise does not require early versions of product with demonstrable functionality

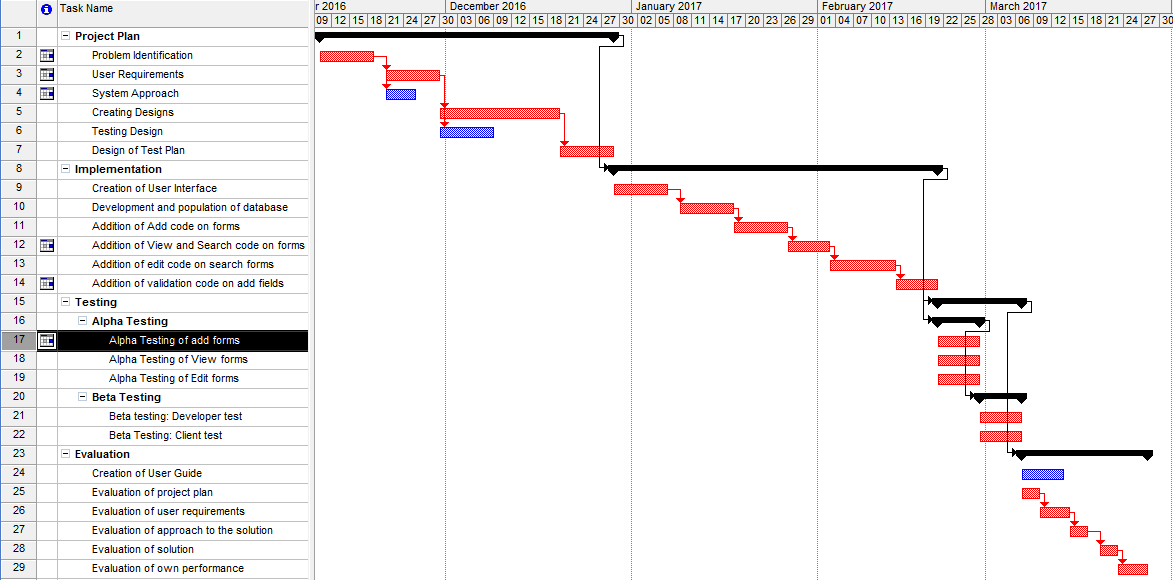
## Waterfall

One of the more traditional methodologies, waterfall looks at the stages of the development process and implements them linearly and completely. This sequential implementation of the core stages of development is often represented in diagrams which resemble the flow of a waterfall. This is a very intuitive and simplistic although primitive way of dealing with a software project. Its origins lie in industries which are by nature very rigid and sequential such as construction. When we construct a bridge we design and build it, in that order. We don’t expect the design to change throughout the building process or new features to be added when construction is completed. As such waterfall puts great emphasis on requirements gathering and design.

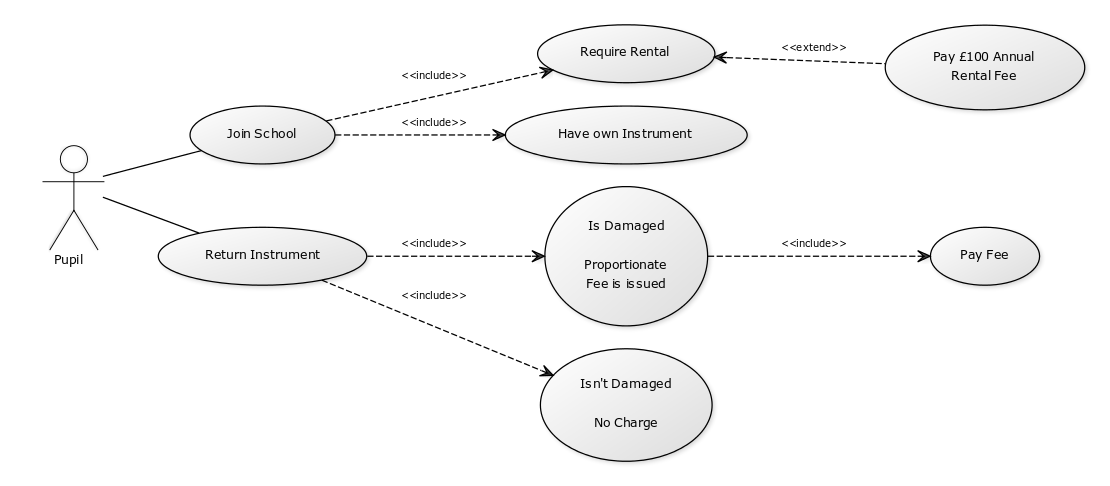
Although it takes the longest, I’d like to use this methodology in the project. As I will be developing the system alone, it is best to take its paced step-by-step approach wherein I completely fulfil each step of the process in order to ensure a high quality final product.

This is the methodology I would like to use during the creation of this project.

## GANTT Chart



# Use-Case Diagram



# Initial ER Diagram

**Instrument** - Instrument ID, Instrument Type, Instrument Name, Instrument Brand, Stockist ID, Instrument Cost, Purchase Cost

**Damage and Repair** – Repair\_ID Rental ID, Damage Analysis, Repair Date, Repair Cost,

**Rental** – Rental ID, Instrument ID, Student ID, Date Rented, Due Return Date

**Student** –Student ID Student Forename, Student Surname, Student Address Line 1, Student Address Line 2, Student Post Code, Phone Number

**Card** – Card\_ID, Student ID, Card Type, Card Number, Name on Card, Expiry Date, Security Code

**Stockist** – StockistID, Stockist Name, Stockist Address Line 1, Stockist Address Line 2, Stockist Post Code, Stockist Phone Number

Payment - Payment\_ID, Card\_ID, Student\_ID, Rental\_ID, /Repair\_ID, Amount, Payment\_Date

# User Requirements

I have been asked develop a system to help improve the administrative capability and reliability of the Mitchell school of music. They are a small business who run professional individual lessons during term time, specialist summer courses and also accommodate for instrument rental. He has a specific number of classrooms, private rooms and a single hall for performances. There are a range of rates for different scenarios that must be catered to, and each student is guaranteed a certain amount of live performances in local. The section I’ll be covering is that based around the instrument rental offered by Mitchell’s school of music. It will involve the registration of new pupils, their respective instruments, whether or not they avail of the rental system and a 7 separate log of instruments available for rental alongside their conditions and rental/repair history.

## Functional

1. The System should be able to keep information on each instrument available for rental
2. Any person wishing to rent an instrument has to be a registered as a student with the school
3. Students may only rent instruments that they are taking lessons in
4. The System should be capable of containing details of all students.
5. The System should keep track of rentals by keeping records of each rental
6. The System should be able to create a document that draws information from the student and instrument table in order to create a signable agreement for the rental of instrument.
7. The System should have a record of each payment made to the company from students pertaining to rental
8. The System should be able to identify who has and hasn’t paid their instrument rental fees, and contain a method of contact to request this payment
9. The System should be able to record and calculate damages and repair costs.
10. The System will have a method of storing information on the stockists that provide their rentable instruments
11. The system must give students an option to renew rental of an instrument after one year.
12. The system will automatically request replacement instruments from stockists should an instrument be damaged beyond repair during rental period, or should the cost of repair exceed the price of a new instrument

## Non Functional

1. Ensure a comprehensive set of designs are created in advance of development
2. Use an easily readable font
3. Ensure the system has minimal bugs and is appropriately tested
4. Use complimentary colours and colours that aren’t harsh to the eyes
5. A log in system for security purposes
6. A user friendly, self-self-explanatory UI so that people who aren’t used to using a PC for administration won’t have a large learning curve

# Design

# Normalisation

0NF

Rental: Student Forename, Student Surname, Student ID, Address Line 1, Address Line 2, Post Code, Phone Number, Skill Level, Fees Paid, Instrument Type, Instrument ID, Cost, Damage, Repair History, Stockist, Brand, Condition, Date Rented, Due return date.

1NF

Rental: Instrument Type, Instrument ID, Cost, Damage, Repair History, Stockist, Brand, Condition, Rental Period

Student: Student Forename, Student Surname, Student ID, Address Line 1, Address Line 2, Post Code, Phone Number, Skill Level, Fees Paid, Instrument Type, Instrument ID

2NF

Rental: Instrument Type, Instrument ID, Cost, Damage, Repair History, Stockist, Brand, Condition, Rental Period

Student: Student Forename, Student Surname, Student ID, Address Line 1, Address Line 2, Post Code, Phone Number, Skill Level, Fees Paid, Instrument Type, Instrument ID

3NF

Rental: Student ID, Instrument ID, Rental Period

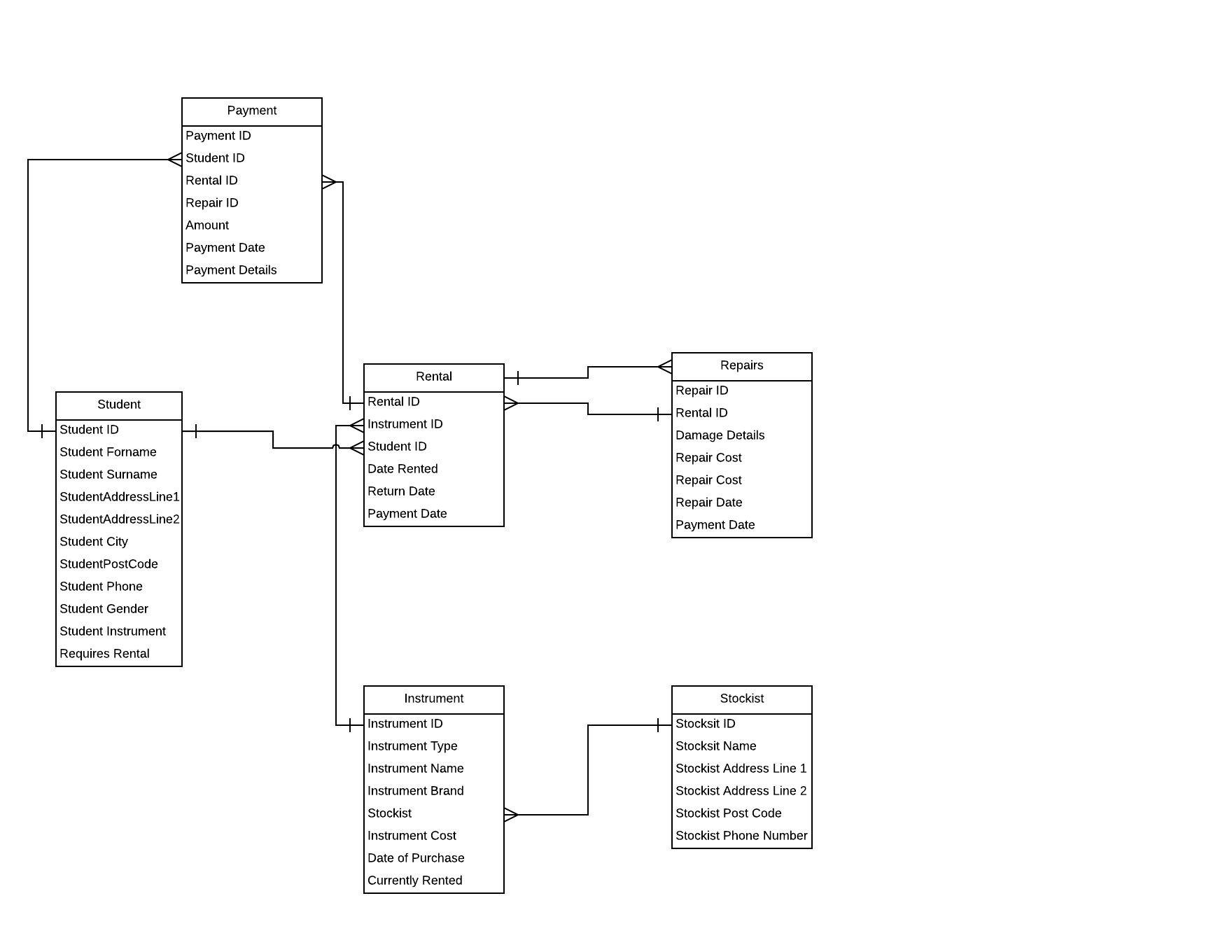
Student: Student Forename, Student Surname, Student ID, Address Line 1, Address Line 2, Post Code, Phone Number, Skill Level, Fees Paid, Instrument Type, Instrument ID

Instrument: Instrument ID, Instrument Name, , StockistID, Instrument Cost, Repair history, Condition

Returns: Student ID, Instrument ID, Instrument Cost, Repair Cost, Damage.

Stockist: Stockist Name, Stockist ID, Instrument Name

# Final ER Diagram



## Table Designs

### Student Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Data Type** | **Length** | **Sample Data** |
| Student Forename | Char | 20 | *Joe* |
| Student Surname | *Char* | *20* | ***Blogs*** |
| Student ID | *Integer* | *4* | *1234* |
| Student Address Line 1 | *VarChar* | *20* | *Somewhere* |
| Student Address Line 2 | *VarChar* | *20* | *Avenue* |
| Student Post Code | *VarChar* | *7* | *BT666TB* |
| Student Phone | *Integer* | *10* | *07712345* |

### Instrument Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Data Type** | **Length** | **Sample Data** |
| Instrument ID | Integer | 4 | 3546 |
| Instrument Type | *VarChar* | *10* | *Guitar* |
| Instrument Name | *VarChar* | *30* | *Les Paul* |
| Instrument Brand | *Char* | *20* | *Gibson* |
| Stockist ID | *Integer* | *4* | *2345* |
| Instrument Cost | *Currency* | *5* | *£600* |

### Rental Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Data Type** | **Length** | **Sample Data** |
| Rental ID-Int(4) | *Int* | *4* | *2326* |
| Instrument ID-int(4) | *Int* | *4* | *1234* |
| Student ID-int(4) | *Int* | *4* | *4444* |
| Rental Status-Bool | *Bool* | *1* | *Returned* |
| Date Rented-Date | *Date* | *8* | *11/11/11* |
| Return Date-Date | *Date* | *8* | *11/11/12* |
| Payment Date-Date | *Date* | *8* | *11/11/11* |

### Payment Table

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Data Type | Length | Sample Data |
| Payment\_ID | Integer | 4 | *2323* |
| Card\_ID) | Integer | 4 | *1234* |
| Student\_ID | Integer | 4 | *1111* |
| Rental\_ID | Integer | 4 | *2468* |
| Repair\_ID | Integer | 4 | *4354* |
| Amount | *Currency* | *5* | *£30* |
| Payment\_Date | *Date* | *8* | *11/11/2011* |

### Stockist Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Data Type** | **Length** | **Sample Data** |
| Stockist ID | Integer | 4 | 2323 |
| Stockist Name | *VarChar* | *10* | *Matchetts Music* |
| Stockist Address Line | *VarChar* | *20* | *24 Ormeau Avenue* |
| Stockist Address Line | *VarChar* | *20* | *Belfast* |
| Stockist Post Code- | *VarChar* | *7* | *BT78 2FT* |
| Stockist Phone | *Integer* | *10* | *077654321* |

### Repairs Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Data Type** | **Length** | **Sample Data** |
| Repair ID | Integer | 4 | 6879 |
| Rental ID | Integer | 4 | 4657 |
| Damage Details | *Varchar* | *50* | *Snapped Strings* |
| Repair Cost | *Currency* | *5* | *£50* |
| *Repair Date* | *Date* | *8* | *11/12/2017* |

## Form Designs

Loading…%

**lblLoading**

**pgrLoadingBar**

### Loading Screen

A timer used to increase the progress bar with its increment, not visible

A timer used to increase the progress bar with its increment, not visible

|  |  |  |  |
| --- | --- | --- | --- |
| **Form** | **Property** | **Property Name** | **Purpose** |
| frmLoadingScreen | Progress Bar | pgrLoadingBar | Mainly aesthetic; allows the rest of the system time to load |
| frmLoadingScreen | Label | lblLoading | To display the percentage of the bar that has been filled by |
| frmLoadingScreen | Timer | Timer | Increase progress bar by 20 every second in increments of 5 |

### Log In

Log In

Log In

**lblLogIn**

**txtUsername**

**lblUserName**

**lblPassword**

**txtPassword**

**btnLogIn**

Username

Password

|  |  |  |  |
| --- | --- | --- | --- |
| **Form** | **Property** | **Property Name** | **Purpose** |
| frmLogIn | Label | lblLogIn | To display the name of the page |
| frmLogIn | Label | lblUsername | To display ‘Username’ beside txtUsername |
| frmLogIn | Label | lblPassword | To display ‘Password’ beside txtPassword |
| frmLogIn | Textbox | txtUsername | Space to enter username credential which will be compared with the stored credentials to grant access to the rest of the system |
| frmLogIn | Textbox | txtPassword | Space to enter password credential which will be compared with the stored credentials to grant access to the rest of the system |
| frmLogIn | Button | btnLogIn | Verifies Username and Password, displays frmMainMenu |

### Add Student

Add Student

Add Student

**lblTitle**

**lblForename**

**lblSurname**

**btnExit**

Student ForeName

Clear Fields

Exit

**btnClear**

**btnAdd**

Student Surname

Student ID

**txtForename**

**txtSurname**

**lblStudentID**

**txtStudentID**

Address

Postcode

Phone Number

**txtPhone**

**lblPhone**

**txtPostcode**

**lblPostcode**

**txtAddress**

**lblAddress**

|  |  |  |  |
| --- | --- | --- | --- |
| **Form** | **Property** | **Property Name** | **Purpose** |
| frmAddStudent | Label | lblTitle | Display title of form |
| frmAddStudent | Label | lblForename | Display the name of the corresponding field |
| frmAddStudent | Label | lblSurname | Display the name of the corresponding field |
| frmAddStudent | Label | lblStudentID | Display the name of the corresponding field |
| frmAddStudent | Label | lblAddress | Display the name of the corresponding field |
| frmAddStudent | Label | lblPostcode | Display the name of the corresponding field |
| frmAddStudent | Label | lblPhone | Display the name of the corresponding field |
| frmAddStudent | Textbox | txtForename | Textbox allows the user to insert the relevant text to the corresponding label |
| frmAddStudent | Textbox | txtSurname | Textbox allows the user to insert the relevant text to the corresponding label |
| frmAddStudent | Textbox | TxtStudentID | Textbox allows the user to insert the relevant text to the corresponding label |
| frmAddStudent | Textbox | txtAddress | Textbox allows the user to insert the relevant text to the corresponding label |
| frmAddStudent | Textbox | txtPostcode | Textbox allows the user to insert the relevant text to the corresponding label |
| frmAddStudent | Textbox | txtPhone | Textbox allows the user to insert the relevant text to the corresponding label |
| frmAddStudent | Button | btnAddStudent | When this button is clicked, if all mandatory fields are filled correctly, a record of a student will be added to the student database according to the data in the textboxes. |
| frmAddStudent | Button | btnClearFields | The content of all current textboxes on screen will be cleared on click |
| frmAddStudent | Button | btnExit | This will close the program when clicked. |

### Add Rental

Add Student

Add Rental

**lblTitle**

**lblStudentID**

**lblRentalDate**

**btnExit**

Student ID

Clear Fields

Exit

**btnClear**

**btnAdd**

Rental Date

**txtStudentID**

**dtpRentalDate**

Instrument ID

Payment Date

**dtpPaymentDate**

**lblPaymentDate**

**txtInstrumentID**

**lblInstrumentID**

|  |  |  |  |
| --- | --- | --- | --- |
| **Form** | **Property** | **Property Name** | **Purpose** |
| frmLogIn | Label | lblTitle | To display the name of the page |
| frmLogIn | Label | lblUsername | To display ‘Username’ beside txtUsername |
| frmLogIn | Label | lblPassword | To display ‘Password’ beside txtPassword |
| frmLogIn | Textbox | txtUsername | Space to enter username credential which will be compared with the stored credentials to grant access to the rest of the system |
| frmLogIn | Textbox | txtPassword | Space to enter password credential which will be compared with the stored credentials to grant access to the rest of the system |
| frmLogIn | Button | btnLogIn | Verifies Username and Password, displays frmMainMenu |

### Add Repair

Add Student

Add Repair

**lblTitle**

**lblRentalID**

**lblDamageDetails**

**btnExit**

Rental ID

Clear Fields

Exit

**btnClear**

**btnAdd**

Damage Details

**txtRentalID**

**lblPaymentDate**

PaymentDate

Repair Cost

**dtpPaymentDate**

**lblRepairCost**

**dtpPaymentDate**

**lblPaymentDate**

Repair Date

txtRepairCost

|  |  |  |  |
| --- | --- | --- | --- |
| **Form** | **Property** | **Property Name** | **Purpose** |
| frmLogIn | Label | lblTitle | To display the name of the page |
| frmLogIn | Label | lblUsername | To display ‘Username’ beside txtUsername |
| frmLogIn | Label | lblPassword | To display ‘Password’ beside txtPassword |
| frmLogIn | Textbox | txtUsername | Space to enter username credential which will be compared with the stored credentials to grant access to the rest of the system |
| frmLogIn | Textbox | txtPassword | Space to enter password credential which will be compared with the stored credentials to grant access to the rest of the system |
| frmLogIn | Button | btnLogIn | Verifies Username and Password, displays frmMainMenu |

## View Students

Add Student

Add Rental

**lblTitle**

**lblStudentID**

**lblRentalDate**

**btnExit**

Student ID

Clear Fields

Exit

**btnClear**

**btnAdd**

Rental Date

**txtStudentID**

**dtpRentalDate**

Instrument ID

Payment Date

**dtpPaymentDate**

**lblPaymentDate**

**txtInstrumentID**

**lblInstrumentID**

|  |  |  |  |
| --- | --- | --- | --- |
| **Form** | **Property** | **Property Name** | **Purpose** |
| frmLogIn | Label | lblTitle | To display the name of the page |
| frmLogIn | Label | lblUsername | To display ‘Username’ beside txtUsername |
| frmLogIn | Label | lblPassword | To display ‘Password’ beside txtPassword |
| frmLogIn | Textbox | txtUsername | Space to enter username credential which will be compared with the stored credentials to grant access to the rest of the system |
| frmLogIn | Textbox | txtPassword | Space to enter password credential which will be compared with the stored credentials to grant access to the rest of the system |
| frmLogIn | Button | btnLogIn | Verifies Username and Password, displays frmMainMenu |

## View Instruments

Add Student

Add Rental

**lblTitle**

**lblStudentID**

**lblRentalDate**

**btnExit**

Student ID

Clear Fields

Exit

**btnClear**

**btnAdd**

Rental Date

**txtStudentID**

**dtpRentalDate**

Instrument ID

Payment Date

**dtpPaymentDate**

**lblPaymentDate**

**txtInstrumentID**

**lblInstrumentID**

|  |  |  |  |
| --- | --- | --- | --- |
| **Form** | **Property** | **Property Name** | **Purpose** |
| frmLogIn | Label | lblTitle | To display the name of the page |
| frmLogIn | Label | lblUsername | To display ‘Username’ beside txtUsername |
| frmLogIn | Label | lblPassword | To display ‘Password’ beside txtPassword |
| frmLogIn | Textbox | txtUsername | Space to enter username credential which will be compared with the stored credentials to grant access to the rest of the system |
| frmLogIn | Textbox | txtPassword | Space to enter password credential which will be compared with the stored credentials to grant access to the rest of the system |
| frmLogIn | Button | btnLogIn | Verifies Username and Password, displays frmMainMenu |

# Testing Design

Log In form

This form is required before the rest of the system can be accessed, with the function to ass new users contained within the rest of the system to add security to the system, satisfying non-functional requirement 5. The clearly labelled buttons and textboxes adhere to the guidelines outlined in the other non-functional requirements

Menu form

This form the user can access every other form in the system with the only exception being the loading screen which only appears upon start up, making it a useful access hub for the rest of the system. This clearly labelled navigation system satisfies non-functional requirement 6.

Add Form

This form contains three tabs which allow the user to choose between which type of record they wish to add to the system, the navigation of which is clearly outlined by the tabs labels, again adhering to the guidelines specified in the non-functional requirements. Requirements outlining the storage of details will be achieved through the inclusion of the database tables the designs of which are found earlier in the document. The three requirements specifying the ability to add records to these tables are fulfilled by clicking on the respective ‘add’ buttons in their respective tabs.

A user using this form will first identify what they wish to add (a student, a rental or a repair) and they would click on the respective tab and fill in the details in the labelled fields, the verification will ensure the data is appropriate for the table when the click add, and if so then the data will be added to the respective table.

# Design of Test Plan

|  |  |  |
| --- | --- | --- |
| # | Form | Intended Result |
| 1 | frmLoadingScreen | The percentage outlined in ‘lblLoading’ will correspond with the percentage of ‘pgrLoadingBar |
| 2 | frmLoadingScreen | Once the progress bar fills the user should automatically be brought to ‘frmLogIn’ |
| 3 | frmLogIn | When incorrect credentials are added and ‘btnLogIn’ is clicked, msgbxIncorrect will appear prompting the user to check their credentials |
| 4 | frmLogIn | When correct credentials are used and ‘btnLogIn’ is clicked, the user will be brought to frmMain following being shown msgbxCorrect which will display a personalised welcome message based on their name. |
| 5 | frmLogin | The data in txtUsername and txtPassword will be compared to the credentials in the ‘users’ file which will then determine who can access the rest of the program |
| 6 | frmMenu | Upon clicking ‘tspAddNewUser’ the user will be brought to ‘frmAddUser |
| 7 | frmAddUser | Upon clicking ‘btnMen’ the user will be brought back to ‘frmMenu’ |
| - | frmAddUser | Upon clicking ‘Add User’ the three textboxes will be checked, and: |
| 8 | FrmAddUser | If the credentials have already been taken, a messagebox will appear asking the user to correct the problem |
| 9 | frmAddUser | If the two password fields don’t match then the user will be prompted to correct the problem by a messagebox |
| 10 | frmAddUser | If the credentials haven’t been taken and the passwords fields match, a new user will be added to ‘Users.txt’ which can be used to sign in in the future. A messagebox will appear confirming their success. |
| 11 | frmMenu | When btnAddForm is clicked, the user will navigate to frmAddForm |
| 12 | frmAddForm | If an invalid postcode is inputted then on attempt to add the record (by clicking btnAddStudent) a messagebox will prompt the user to correct it |
| 13 | frmAddForm | If all data inputted is appropriate then clicking ‘btnAddStudent’ will add the record to the student table in the database. |
| 14 | frmReport | After typing an instrument name and the fillby button, the report will be filled with a list of every student that plays that instrument. |

# Implementation

# Testing

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Form | Intended Result | Pass/Fail | Evidence | Corrective Action |
| 1 | frmLoadingScreen | The percentage outlined in ‘lblLoading’ will correspond with the percentage of ‘pgrLoadingBar | Pass |  | N/A |
| 2 | frmLoadingScreen | Once the progress bar fills the user should automatically be brought to ‘frmLogIn’ | Pass |  | N/A |
| 3 | frmLogIn | When incorrect credentials are added and ‘btnLogIn’ is clicked, msgbxIncorrect will appear prompting the user to check their credentials | Fail |  | Error msgbx command was mistyped and hence did not execute when appropriate, this has since been fixed |
| 4 | frmLogIn | When correct credentials are used and ‘btnLogIn’ is clicked, the user will be brought to frmMain following being shown msgbxCorrect which will display a personalised welcome message based on their name. | Pass |  |  |
| 5 | frmLogin | The data in txtUsername and txtPassword will be compared to the credentials in the ‘users’ file which will then determine who can access the rest of the program | Pass |  |  |
| 6 | frmMenu | Upon clicking ‘tspAddNewUser’ the user will be brought to ‘frmAddUser | Pass |  |  |
| 7 | frmAddUser | Upon clicking ‘btnMenu’ the user will be brought back to ‘frmMenu’ | Pass |  |  |
| - | frmAddUser | Upon clicking ‘Add User’ the three textboxes will be checked, and: | N/A |  |  |
| 8 | FrmAddUser | If the credentials have already been taken, a messagebox will appear asking the user to correct the problem | Pass |  |  |
| 9 | frmAddUser | If the two password fields don’t match then the user will be prompted to correct the problem by a messagebox | Pass |  |  |
| 10 | frmAddUser | If the credentials haven’t been taken and the passwords fields match, a new user will be added to ‘Users.txt’ which can be used to sign in in the future. A messagebox will appear confirming their success. | Pass |  |  |
| 11 | frmMenu | When btnAddForm is clicked, the user will navigate to frmAddForm | Pass |  |  |
| 12 | frmAddForm | If an invalid postcode is inputted then on attempt to add the record (by clicking btnAddStudent) a messagebox will prompt the user to correct it | Pass |  |  |
| 13 | frmAddForm | If all data inputted is appropriate then clicking ‘btnAddStudent’ will add the record to the student table in the database. | Pass |  |  |
| 14 | frmReport | After typing an instrument name and the fillby button, the report will be filled with a list of every student that plays that instrument. | Pass |  |  |

# Testing Evidence

|  |  |  |  |
| --- | --- | --- | --- |
| # | Evidence | # | Evidence |
| 1 |  | 2 |  |
| 3 |  | 4  +  5 |  |
| 6 |  | 6 |  |
| 7 |  | 7 |  |
| 8 |  | 8 |  |
| 9 |  | 10 |  |
| 11 |  | 11 |  |
| 12 |  | 12 |  |
| 13 |  | 13 |  |
| 14 |  |  |  |

# Evaluation

## Evaluation of project plan

I felt as though the project plan was absolutely integral to the success of the project. The creation of table and form designs took a significant initial investment of my time however it also meant that when implementation began, I had a solid set of designs from which to work from. When developing in Visual Studio last year I often found myself lost because my designs were lacking somewhat, and the ideas I may have had for my project weren’t fully thought through. Learning from that experience I invested a lot more time into my planning stage this year, and once development began the rigorous preparation stage payed off almost immediately when I was faced with creating tables. Normally, I’d have had to guess at what fields and tables I would need and make up for mistakes later, but as I had tested and planned these tables and their functions throughout my planning stages I knew exactly what I would need in order for the system to work properly, which provided me with an extremely streamlined development process. My project plan had several stages including;

* An overview of the task at hand
* The development of user requirements with the client
* Extensive research into the available programming methodologies to use
* Creation of a plan through Microsoft project including a GANTT chart
* Initial ER diagram followed by normalisation and a final ER diagram
* Creation of a Use-Case diagram
* Table designs including test data
* Form designs including property tables and some pseudo-code
* The testing of my designs against use requirements to ensure they fully satisfied all agreed requirements, and
* A test plan, which outlined all the functions I would have to test once the system was developed

This plan was certainly extensive, and took a very significant amount of my time, but each section of it proved integral to the success the production of the system; it both assured the quality of the final product by creating a blueprint created in close conjunction with the client, whilst also saving time and money by streamlining the development process in such a way that I was essentially following my own previously outlined instructions, with the only real issues I encountered being in the learning curve associated with linking SQL with C# and database I/O with the user (which will be elaborated upon in later sections of this evaluation)

Overall, each section of the plan can be directly linked with at least one instance of beneficial utility later in the development process:

*An overview of the task at hand:*

This one was an obvious necessity; I read the case study provided and highlighted they key weaknesses in the Mitchel Music administration system. I then carefully chose which section would be the most important to approach. I chose to develop the rental section of the system because the most money was involved in the school’s constant instrument rental and damage issues in single instalments. I then summarised this in this first section before continuing on to the next section.

*The development of user requirements with the client*

I’ll keep this section brief given that a much more specific insight into this section is provided in the next section of the evaluation. In terms of planning however, these requirements formed the crux of my system. They provided me with essentially a checklist of features to develop, which then automatically shaved the rest of the wealth of the features the system could have contained. Having said that however, my first attempt at requirements were created before I had a vast knowledge of SQL and database systems in general, so a few of my requirements were created on a false pretence of what and wasn’t possible, particularly in terms of scope with the time given, such as an original inclusion of development in the payment section of the system, where I would include cards and an ability to process cheques. It turned out that a professional instance of digital commerce is quite a complex system and a large industry on its own. In a slightly later redraft I chose to exclusively record the cost of payments associated by person, leaving the actual processing of currency to the people at Mitchel’s Music.

*Extensive research into the available programming methodologies*

Once again this is a topic which is addressed in a greater detail in a later section of this evaluation; namely “Evaluation of approach to solution”. Overall however the few days spent researching these approaches proved highly beneficial throughout the rest of the development process. Adhering to any methodology over a freelance approach ensures that all factors of development are considered and these guidelines relieved me of a potentially time consuming and dangerous task of figuring out the best approach myself.

*Creation of a plan through Microsoft project including a GANTT chart*

This section had an initial learning curve of learning to use a new software, in Microsoft Project. Having said this however, it also proved very useful to me. It gave me time periods in which to complete each section of the system and as long as I followed those guidelines I was guaranteed to finish the system in the agreed time. The GANTT chart seen in this document is my second attempt at a GANTT chart following a first attempt which was still relatively strong but featured less break down of tasks and gave me no window for overlap, which seemed like a good idea at the time but just resulted in ambiguity of task completion and strict time windows. The freedom within some sections (particularly in the implementation section) proved invaluable later, and this is reflect in the included iteration of my GANTT chart.

*Initial ER diagram followed by normalisation and a final ER diagram*

When I created my first ER diagram I was new to the concept of Entity Relationship modelling and as can be seen in my initial ER I didn’t have the greatest idea of how to develop relational databases. I began my research and started to understand that whilst the fields were valid in my initial ER, they weren’t efficient in terms of data keeping, and I didn’t specify relationship types.

## Evaluation of user requirements

Testing of Designs against User Requirements

1. *The System should be able to keep information on each instrument available for rental*

A database file is used and maintained which then holds the information so long as the file isn’t manually deleted

1. *Any person wishing to rent an instrument has to be a registered as a student with the school*

On the rental form, a student ID is a required field meaning only registered students will be able to rent.

1. *Students may only rent instruments that they are taking lessons in*

The rental form instrument selector will automatically filter based on the instrument associated with the selected students instrument of choice.

1. *The System should be capable of containing details of all students.*

There is no maximum capacity for the database and it is dependent upon the storage space of the computer it is being used on.

1. *The System should keep details of each rental.*

A database file is used and maintained which then holds the information so long as the file isn’t manually deleted

1. *The System should be able to create a document that draws information from the student and instrument table in order to create a signable agreement for the rental of instruments.*

-Will return to this when I’ve completed my report

1. *The System should have a record of each payment made to the company from students pertaining to rental*

The payment table and the ‘view payments’ form will allow the user of the system to view all payments that are made to the company.

1. *The System should be able to identify who hasn’t paid their instrument rental fees, and contain a method of contact to request this payment*

The ‘fees paid’ field of the rental table will be automatically verified by cross referencing the payment tables records with the rental ID’s associated. For example, if the payment table lacks a record associated with the rental ID being verified, then its ‘fees paid’ would be set to ‘no’.

1. *The System should be able to record damages and repair costs.*

The repairs table accommodates this

1. *The System will have a method of storing information on the stockists that provide their rentable instruments*

A stockist will be associated with each instrument complete with contact information, seen in the designs.

1. *The system must give students an option to renew the rental of an instrument after one year.*

Once the date specified for return on a given rental record is reached, the user will be prompted to either renew this rental (Thereby extending the return date by a further year) or else marking the instrument as returned.

1. *The system will automatically request replacement instruments from stockists should an instrument be damaged beyond repair during rental period, or should the cost of repair exceed the price of a new instrument*

When damages are added associated with an instrument, all other cost of damages associated with these instruments will be added up and if it is greater than the cost of a replacement instrument then the system will automatically request a replacement instrument from the associated stockist.

## Non Functional

1. *Ensure a comprehensive set of designs are created in advance of development*
2. *Use an easily readable font*
3. *Ensure the system has minimal bugs and is appropriately tested*
4. *Use complimentary colours and colours that aren’t harsh to the eye*
5. *A log in system for security purposes*
6. *A user friendly, self-explanatory UI so that people who aren’t used to using a PC for administration won’t have a large learning curve*

## Evaluation of approach to the solution

The approach I took to this solution was the waterfall methodology, which in many ways worked effectively to my benefit, however fell short in some areas. I think the major positive with this approach was that as an inexperienced developed, waterfall provided me with a series of steps to complete in an order to ensure success. Other approaches such as extreme programming may have been more time effective, however the lack of structure may have left my work unprofessional. One advantage of iterative methodologies over waterfall would have been the use of timeboxing, which may have helped me with time management, however once again my inexperience leaves me glad that I chose waterfall because otherwise I’d be unable to take extra time overcoming difficulties in the systems production.

## Evaluation of solution

## Evaluation of own performance

When I began this project I had limited experience with C# and object oriented development, as well as no real experience in integrating it with other languages and no experience whatsoever in SQL. This project demanded the use of a user friendly object oriented front end system which consistently linked with an SQL database which was a huge step up for me but likewise one which is infinitely useful in the industry. It would be naïve at best however to say that it was only in the implementation of this system that I developed personally; the planning stage was excessive too, and I learned how to use the Microsoft office project software wherein I created the Gantt chart seen earlier in this document. Not only in software development but in wider life time management is absolutely paramount to success, and structuring an organised plan with time-based subsections is very good practice and something which both ensured my own success in this project but also improved by time keeping capabilities in my other subjects and extra-curricular activities. The planning stages also saw me utilising a Use-Case diagram for the first time and began my research into UML. I didn’t understand what normalisation was, however this task gave me an opportunity to learn the process and create a realistic normalised data set to the third normal form in a realistic situation. Taking this normalised data I was able to create an ER Diagram, something I was previously unfamiliar with but quickly learned about. Learning about the ‘crows feet’ system and how they can be used to indicate the relationships between tables was enriching and aided in my understanding and use of data throughout the rest of the project.

# Appendix:

# SQL Code

CREATE TABLE [dbo].[Instrument] (

[Instrument ID] INT NOT NULL,

[Instrument Type] NVARCHAR (50) NULL,

[Instrument Name] NVARCHAR (50) NULL,

[Instrument Brand] NVARCHAR (50) NULL,

[Stockist ID] INT NULL,

[Instrument Cost] MONEY NULL,

[Date Of Purchase] DATE NULL,

PRIMARY KEY CLUSTERED ([Instrument ID] ASC),

CONSTRAINT [FK\_Instrument\_ToStockist] FOREIGN KEY ([Stockist ID]) REFERENCES [dbo].[Stockist] ([Stockist ID])

CREATE TABLE [dbo].[Payment] (

[Payment ID] INT NOT NULL,

[Student ID] INT NULL,

[Rental ID] INT NULL,

[Repair ID] INT NULL,

[Amount] MONEY NULL,

[Payment Date] DATE NULL,

[Payment Details] DATE NULL,

PRIMARY KEY CLUSTERED ([Payment ID] ASC),

CONSTRAINT [FK\_Payment\_ToRental] FOREIGN KEY ([Rental ID]) REFERENCES [dbo].[Rental] ([Rental ID]),

CONSTRAINT [FK\_Payment\_Repairs] FOREIGN KEY ([Repair ID]) REFERENCES [dbo].[Repairs] ([Repair ID]),

CONSTRAINT [FK\_Payment\_ToStudent] FOREIGN KEY ([Student ID]) REFERENCES [dbo].[Student] ([StudentID])

CREATE TABLE [dbo].[Payment] (

[Payment ID] INT NOT NULL,

[Student ID] INT NULL,

[Rental ID] INT NULL,

[Repair ID] INT NULL,

[Amount] MONEY NULL,

[Payment Date] DATE NULL,

[Payment Details] DATE NULL,

PRIMARY KEY CLUSTERED ([Payment ID] ASC),

CONSTRAINT [FK\_Payment\_ToRental] FOREIGN KEY ([Rental ID]) REFERENCES [dbo].[Rental] ([Rental ID]),

CONSTRAINT [FK\_Payment\_Repairs] FOREIGN KEY ([Repair ID]) REFERENCES [dbo].[Repairs] ([Repair ID]),

CONSTRAINT [FK\_Payment\_ToStudent] FOREIGN KEY ([Student ID]) REFERENCES [dbo].[Student] ([StudentID])

);

CREATE TABLE [dbo].[Repairs] (

[Repair ID] INT IDENTITY (1, 1) NOT NULL,

[Rental ID] INT NULL,

[Damage Details] NVARCHAR (50) NULL,

[Repair Cost] DECIMAL (18) NULL,

[Repair Date] DATE NULL,

[Payment Date] DATE NULL,

PRIMARY KEY CLUSTERED ([Repair ID] ASC),

CONSTRAINT [FK\_Repairs\_ToRental] FOREIGN KEY ([Rental ID]) REFERENCES [dbo].[Rental] ([Rental ID])

);

CREATE TABLE [dbo].[Stockist] (

[Stockist ID] INT NOT NULL,

[Stockist Name] NVARCHAR (50) NULL,

[Stockist Address Line 1] NVARCHAR (50) NULL,

[Stockist Address Line 2] NVARCHAR (50) NULL,

[Stockist Post Code] NVARCHAR (8) NULL,

[Stockist Phone Number] NVARCHAR (11) NULL,

PRIMARY KEY CLUSTERED ([Stockist ID] ASC)

);

CREATE TABLE [dbo].[Student] (

[StudentID] INT IDENTITY (1, 1) NOT NULL,

[StudentForename] VARCHAR (50) NOT NULL,

[StudentSurname] VARCHAR (50) NOT NULL,

[StudentAddressLine1] VARCHAR (50) NOT NULL,

[StudentAddressLine2] VARCHAR (50) NULL,

[StudentCity] VARCHAR (50) NULL,

[StudentPostCode] VARCHAR (50) NULL,

[StudentPhone] NVARCHAR (20) NULL,

[StudentGender] CHAR (10) NULL,

[StudentInstrument] VARCHAR (50) NULL,

[RequiresRental] CHAR (10) NULL,

PRIMARY KEY CLUSTERED ([StudentID] ASC)

);