

Sandbach School



**OBJ: Create an embedded management software built to modernize
a hotel environment from a paper based system, as to meet
workplace standards.**

Lewis Buttle

Contents

Analysis

Project Background History

Project definition

Computational Methods Justification

How - using computational methods - will the aim be achieved?

Data Handling, Inputs and Outputs

Stakeholders

Situational Stakeholders

Stakeholder Survey Results

Research

Preliminary Research

Software Research

Features

Fundamental Features

Advanced Features

Limitations

Requirements

Success Criteria

Design

Problem Decomposition

Initial System Flowchart

Database Relationship Diagram

GUI Plan

Variable Table

Function Table

Key Stages Planning

Final Test Plan

User Satisfaction Survey Plan

Development

Key Stage 1 - Start up system

Key Stage 2 - Database Design & Home Screen Foundation

Key Stage 3 - Booking System with Database (Including Search Function)

Key Stage 4 - Calendar and Calendar Functions

Key Stage 5 - Settings

Evaluation

[Post Development Testing](#)

[Feature Testing & Testing for Robustness](#)

[End User Testing](#)

[Reflection on Function testing \(Usability Features\)](#)

[Fundamental Features](#)

[Advanced Features](#)

[Success Criteria](#)

[Maintenance Issues](#)

[Limitations](#)

Analysis

Project Background History

The idea of providing temporary hospitality to travellers dates back millennia, and the service was a recognizable trait of any early flourishing civilization. The service conceptually arose like any other business idea: to exploit a missing market. This is because in the early

years of civilization (Roman Empire, around 300 AD), trade was essential to daily life, and caravans dispersed across empires to share their own discoveries, and these early travellers needed a place to stay without needing to settle down, as they were likely soon to leave. From this problem, the Romans developed a solution in the form of mansions - to house travellers in temporary 'apartments' whilst they stayed in their empire, though it was initially for the very rich traveller or to give a good impression to diplomats from other empires. Arguably, this shows that luxury hotels were invented before normal hotels, however the same foundation of modern hotels still follows the Roman concept to this day.

The 'comfort' aspect of modern hotels was arguably conceived by the Greek Empire, as the Greeks focused a lot on philosophy, and how and why humanity enjoys the things they do. So from this, the feeling of relaxation was researched and the first 'thermal baths' (or more modernly, the 'spa') were created. The concept was similar to the Roman mansions, though it was a solution to a different problem - to relax. Despite it being an opposing solution, it still follows the foundation of modern hotels in which they aim to provide an environment of utmost service and relaxation for their clients.

From these conceptions the roots of hotels spread with the Roman Empire across Europe, with thermal baths (inspired by the Greeks) being organized in England, Switzerland and the Middle East. From this development, caravanserais were propagated across the middle east to give a resting place for caravans, pilgrims and refugees travelling across the Roman Empire. Unlike the Roman mansions however, these establishments were much more autonomous and ran locally without government intervention. The fact that they were cheap to instigate demonstrates just how quickly the hotel ideology spread during these late Roman years, developing past the fall of the Roman Empire in 476 AD and eventually peaking once travelling became dangerous around the era of the crusades (1095-1297).

It was not until the 15th century when hotels became recognized as an industry, as France declared that all hotels must keep a register of guests and hotels wanting to be created must first be given government

permission. This law was later adopted by Britain, but accommodating to all hospitality based services, such as thermal spas and inns. The hotel industry in Britain burgeoned from this law, with more than 600 inns being registered across the country, and with government support it allowed more thought to be put into the architecture and image of the inns. It was around this time when the concepts of spas and inns finally conjoined to whelm their hospitality relation, where hotels would serve as a place to stay for temporary homing during travel, but also serving the relaxation aspect parallel to this hospitality.

The developing architecture was almost a way to ‘show off’ a hotel’s aspects and to be unique against competition, and this competition is what sparked the development of shared standards in the hotel industry. For example, French hotels would advertise that luxury cuisine was served in their hotel and from this business success, other hotels followed this standard and suddenly serving food was a part of the hotel experience. As well as cuisine, clubs and societies being formed for hotels became a standard, however this modernly grew outdated only to the elite of hotels: usually those which pride their establishment on their history.

With hotels being as widespread as they are today, it would infer that the competition has arisen since the 19th century, with the public actually having a choice as to where they want to stay locally, instead of there being a designated hotel to a certain area, when hotels were more scarce. With this choice, standards are of a hotel’s utmost importance to maintain, to provide the reason for a guest to stay with them and not another hotel service nearby. These standards will serve a basis to the goals of my project.

The fundamental value apparent throughout the history of hotels - innovation to management is fundamental to staying relevant within the industry.

Project definition

A problem fought by many hotels over the past few decades has been the modernization of their room management system. Completely

replacing a paper-based system*; whilst still maintaining the simplicity of it, is a difficult problem to overcome. The paper based systems also had time to perfect how they best contributed towards modern hotel standards, so making this rather large change of system - to a less perfected system, can easily be seen as coming with risk to hotel order. Hence, this modernization aspect of the objective will need to not only modernize, but also be as simple of a transition as possible (being easy to learn for staff, for example).

*A paper based solution meaning the recording of: customer information, which room they are using, how much it will cost, etc - all onto paper via manual writing. It would then likely be stored alphabetically in a filing cabinet, and retrieved from this cabinet manually when needed. A painstakingly slow, but highly simple solution.

The main problem I will be solving is the need in a hotel to store, organize and retrieve data of customers wanting to stay at the hotel. Providing a platform to store this information computationally will be the absolute minimum, but the modernization aspect will be just as important to the success of this project's solution.

The term 'embedded management software' in the objective is important, as it means to have a software which is only in control of a smaller aspect of the entire hotel system, meaning the project is rightfully limited to just a certain (though very fundamental) aspect of the hotel system. This embedded nature was specifically chosen as to focus on the highest possible quality for a smaller part of the system, rather than create a broader software but with the quality being dampened by this broadness. Therefore, the software will be constantly targeted towards management, and every action will need to be relevant to hotel management.

Alongside this idea of the system being 'embedded', I will need to prove that the software can be integrated with a larger system, therefore I will create a door lock system which I will then link to the management system. If they are able to communicate between one another, and be able to share information to make certain decisions then I will deem the 'embedded' nature of the objective as a success.

In conclusion, breaking down the objective into more quantifiable aims would be easier for development comparison. Hence the more extended objective:

A software which deals with the assigning of rooms to guests within a hotel, which can replace a non-computer based system with minimal risk due to change. It will need to be easy to learn, and quick to set up and organize. It must also be proven to be compatible with other systems within the hotel.

Computational Methods Justification

Those in charge of hotel standards over the past century have taken advantage of technology to meet standards, specifically jobs which would operate much more ergonomically through a computer system. However, it is justifiable to say that a human operating some of these jobs which involve guest interaction would be preferred over a computer system, due to a computer not being able to give a 'warm welcome' as much as a receptionist may give: neglecting the fact that hotels aim to give the best experience to guests that they can. This is an example of an ethical issue faced from using computational methods, though they can be minimized through certain social means (for the greeting example, an automatic greeter system could output a welcome followed by the visitor's name). These issues will be somewhat resolved and discussed throughout development.

From this ethical exception is where the project's aim spawns from, as to purposefully restrict myself only to the daily problems of the hotel industry which would benefit fully from using a computer - avoiding any questioning from those advocating for business ethics. Aside from ruling out what *couldn't* be solved fully with a computer, the justification for the project's aim in terms of computational methods is described below.

The main aim would benefit from becoming computer-based for a number of reasons, though the priority justification links to the need for the software to be simple, and computational methods allow this by having a larger array of design possibilities to offer. To better explain this, think of how an integrated development environment (IDE) has

many different design features to give a user a total ‘sandbox’ feel to how they develop their software, and this freedom is exactly what this project needs, as the software can then be designed to have an interface as simple; or as complicated, as I feel necessary due to this amount of freedom. In comparison, a paper based system lacks this openness in technicality, and is extremely one dimensional in how data organization systems could be developed through it.

Through this limitation, a paper based system is completely condemned from developing as a computational system would be able to due to this lack of freedom. For example, using paper simply means noting a customer’s details, filing the details in alphabetical order, and then retrieving these notes physically when necessary although, this tedious system is completely understandable as the paper-based system has had the opportunity of centuries to develop and reach its ‘maximum’ efficiency. Due to reaching this maximum efficiency it seems it leaves the paper based system little space for improvement, unlike what a computational system offers by being developed on IDEs, giving the opportunity to expand and edit a system at any time by just adjusting certain lines of code. This newfound freedom with computational systems makes the computational method seem like next natural ‘step’ in development for these sorts of systems.

The use of computers can also aid in the pursuit of being ‘easy to set up’ (as described in the aim). With modern storage devices being able to carry large amounts of data in rather small vessels, this would allow the entire software’s code to be stored on a small, portable storage device (Such as an SD card). Therefore, if a modern computer is put in place within a hotel it would only need the SD card to be inserted and installed onto computer to finish setting up the system (as well as installing the appropriate softwares onto the other systems such as the lock, so that they can all interact). In terms of compatibility, the computer would also only need to download the appropriate drivers to understand the code I will be writing in (C#) and if it is a windows computer, it will be very much likely that it will already be understood. The fundamental point being, this ability for large storage and quick installation via computers runs parallel

with the desire for quick and easy installation from my project, hence computational methods would be suitable.

Aside from the main ‘freedom’ justification, there are many smaller yet justified reasons for a computational system in place of a paper system:

- Saves on space by no longer needing to fill filing cabinets full of physical information (Data can be saved digitally onto much smaller, versatile storages of memory such as an SD card)
- Information can be backed up much easier with the ability to copy data straight into another card of digital storage, whilst filing cabinets would take much more effort to copy identically, with less room for human error.
- Backing up also provides insurance in case of accidents, such as if a fire occurred in the hotel then data would be much more protected if it was backed up onto multiple servers away from the hotel, rather than in a flammable filing cabinet.
- Data can be inputted, stored, and outputted all from one location, thus saving hotel staff time and effort.
- Data can be organized, searched and compared all through the computer via set coded algorithms, preventing the painstaking task of ordering paper-based files alphabetically. Once again the computational method also minimizes human error this way by handling searches and organizing data on strict algorithms, without the ability to get distracted or make a mistake. This would also save time for the staff in the case they needed to quickly search for a certain customer’s information.
- Customer’s data can be protected much easier on a computer, as computers provide the ability to encrypt information using highly advanced cryptology, so even if the data is stolen, it will still be almost impossible to decipher. This immense security is a major factor in favouring computers over paper for handling sensitive data, as with paper the information is much easier to steal when not digitally confined, and it will likely be unencrypted because of reading efficiency issues (as you would not want to decrypt customer information yourself manually to read it).

How - using computational methods - will the aim be achieved?

In a more physical planning and utility sense - computational methods offer a vast potential for a stable foundation in which to build the code. I will be using the IDE Visual Studio to develop the entirety of my project on as it offers an open platform for development. It offers many of my project's fundamentals, which are:

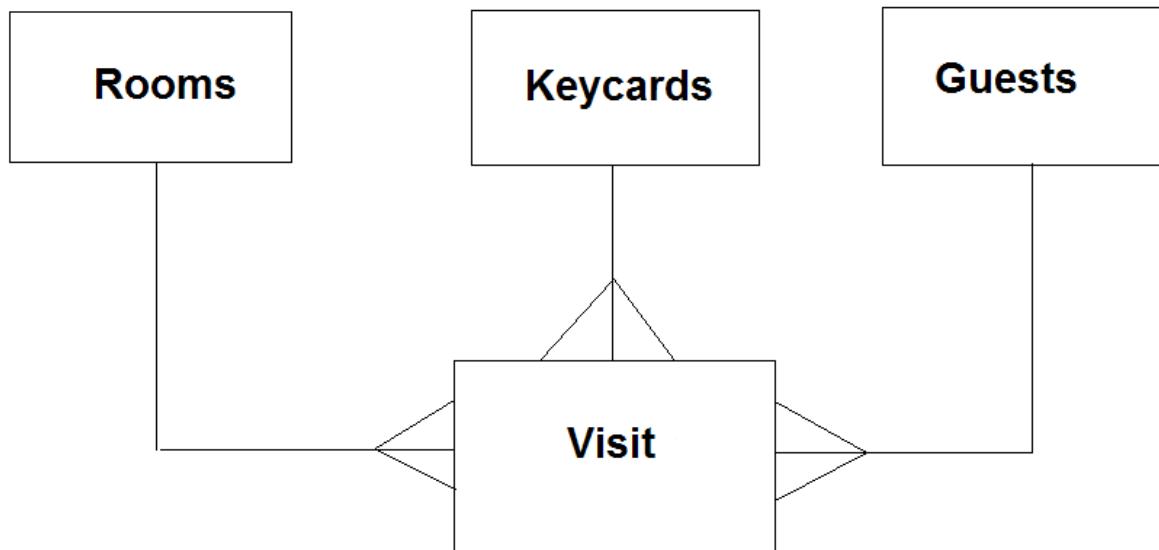
- Database functionality.
- Ability to write in C#.
- Modules (Linking code systems together).
- Graphical user interface functionality.
- Debugging (Stepping, variable checking, breakpoints, syntax highlighting).
- *And most importantly*, the ability for all these functions to interact with one another.

I will be using C# for the project, due to it being perfectly functional to achieve the objective aim, as well as it being a coding language I am most attuned with.

The solution will most prominently require the use of databases to store; and then retrieve, the data of the customers and which room they will be staying in. It is much easier to organize information in a computational database as it not only links information together easier than; say, a paper based method but it keeps this information much more 'separated' in a security sense by keeping customer information and room information apart from one another (seen in database infrastructure image below). To justify computation usage even further, the individual databases could be encrypted with only the relevant database (the 'Visit' database) being available to the hotel staff, so customer data is kept totally private with only their relevant customer information being used - abiding the Data Protection Act (1998).

Databases used in Visual Studio in the project will have access to SQL, meaning other code in Visual Studio can initiate a specific search within

the database files. For example, a member of staff could be looking for certain customers visits in a certain week, so they could use computation via SQL to search for this information quickly and efficiently. The lock system could also use SQL to compare if the method of entry matches



the correct visit times inputted.

Abstraction is a computational method for planning which I will use to output the most efficient solution to the problem, by ignoring unnecessary details. By only focusing on these details (Hotels having multiple rooms, hotels having multiple visitors, location of the hotel) and ignoring the incoherent details (Different sizes of the rooms, hotel appearance, etc) it will allow me to focus on these certain details, thus permitting a more focussed solution.

I will be mainly using a methodology of a relaxed waterfall method, where I will progress throughout stages of planning, creation and then evaluation. The first cycle will likely be huge in comparison to any cycles to come after, as I aim to have a somewhat working prototype from the first cycle. Any other cycles will be for smaller features that will still require a small amount of planning, and evaluation. The method will be relaxed due to the size of the project, which will allow me to make changes quite late into development without much trouble, and if planned correctly then change will be at a minimum. The use of

methodology is another computational method, which is useful due to a methodology concentrating my efforts onto the most relevant parts of development, hence providing a good efficiency.

Data Handling, Inputs and Outputs

The main bulk of information is going to be transferred through the database mentioned above, as the database will serve as the ‘storage’ component of the input-output system. It will store all quantitative data such as customer information in the database, and binary inputs (such as if a button is clicked or not) will simply initiate their purpose and the decision will not be stored.

Initial setup will require special inputs, such as how many rooms the hotel has. It will prompt this input once on setup, and would only be changeable again perhaps in a settings menu. Storage will then be used to check if this initial setup has been already performed, so that guest information and currently occupied rooms can be loaded up on the software and outputted onto the screen.

Data will be inputted through the windows form which will have text boxes to input text, which will be compatible with a keyboard input. The text in the box will then be submitted into the system via a submit button for storage into the database. Buttons will also be a smaller input, to navigate the menu of the software. These inputs will not require storing, and will only provide the output of changing the menu.

Outputs will mainly consist of information on a screen showing when an action has been performed, and the majority of processes will be comparisons and storage between the database and the program. A special output however, will be the lock system which will output upon the signal requesting the lock be opened. The specific output could be the magnet being demagnetized to stop the magnet holding the door.

In a simplified example format:

Keypad scanner inputs boolean(Scanned or not scanned) and integer(Specific number of key card) data. This will be processed by communicating with the database to validate entry, and the date and time of the attempted entry will be stored. The output will then be the door unlocking, or remaining locked. It could also produce an appropriate sound as an output upon scanning the keypad.

The windows form will take; from a mouse peripheral, floating point data and boolean data, to navigate where the mouse is placed on the GUI and whether or not it is 'clicking' respectively. As mentioned above, many of these mouse movements will not be required to be stored - only processed and outputted as the mouse movements are not relevant to be stored for the problem solution. The output of the mouse floating point data will be the movement of the digital pointer on the user's screen, and the boolean click will select whatever the pointer is currently on, such as button that will change to a new interface. Clicking this button will produce a new input for the system, to then process and have an appropriate output depending on the purpose of the button (such a 'submit' button will then alert the system to run a method to store whatever data has been inputted within the text boxes.)

Referencing the text input mentioned before, the form will also have to take keyboard peripheral inputs (integer data to be referenced by the unicode table). This input will not necessarily be stored permanently, it will only be stored temporarily in the text boxes. It will only be stored permanently into the database once the data is submitted, though this is an input-output process by itself. The output of the keyboard will be the text being displayed within the appropriate textbox on the monitor.

Other inputs may include ease of access disability peripherals, where a range of them will need to be tested with the software to ensure full compatibility for a wide range of inputs.

Stakeholders

The most abstract nature of a stakeholder for this project will be any client simply based within the hotel industry (and the system will have a real use for them), but a more accurate and concise stakeholder will not only desire the system itself, but the change in which the system provides - meaning the system is best suited for them and they will gain from every function of the product. The detailed specifics are below:

As a foundation, the software will appeal to any hotel organization looking for a solution to organizing customer and hotel information, and interlacing the information quickly and securely. Any hotel looking for a modernization upgrade their possibly outdated fashion of storing information, to save time and effort by using quicker methods of information inputting, would benefit from this solution.

My solution would appeal to smaller organizations in the hotel industry, perhaps more independently ran businesses that have not yet modernised. This may mean the target market is looking for more efficient alternatives to a paper based system, though are daunted by the idea of modernisation. This is relevant to the device, as I aim for the device to be easy to implement: so that it can replace existing systems without causing too much of a transition scar on the business. This need for modernisation will be the key feature within my stakeholders.

Stakeholders will also be smaller; more independently ran, organizations as they are much more likely to be looking for a modernisation upgrade for their system. In contrast, hotel chains such as Premiere Inn would likely not benefit from my solution, as their economical state has allowed them to keep a modern standard for their hotels, so their hotels will already have a computer-based system to keep ahead of the hotel industry. Therefore, larger chains of hotels would not be the best suited to design this system for, so specifying certain design choices toward independent hotels would have the most positive effect from implementing the system.

The stakeholder would likely have more than one room in their hotel, though the product will be very much viable to have a use for; say, a

single room bed and breakfast establishment. However, the product will be designed for the stakeholder that has multiple rooms, as to satisfy the ability to better organize rooms - whilst a single roomed establishment would only benefit from the simple customer data storage function of the software because room management is non-existent for them. This is because it would not suit my database model, as databases will be used to organize data easier, but if it is only via one room then that amount of information could easily be stored without a computer. Perhaps the ability to store data over a long time through my program would be appealing to single room establishments.

Stakeholders will likely already have a computer within their hotel, as to be able to run my software on it. I would imagine a majority of hotels nowadays must have some sort of computer on their reception desk, though a minority (perhaps an older generation of hotel owners) may be lacking a computer. A computer must be present for the software to be ran. Despite having a computer in their hotel, they may still be lacking a computer-based solution to room management, or perhaps the software they are using already is primitive and underdeveloped to their needs - such as not being able to be embedded. The idea of a stakeholder looking for an upgrade in software is also viable, as it means they are still looking for a 'modernization' in some smaller aspect.

Also, the visitors will be a user of my software as well, as they will be using the lock system integrated into the software. These visitors will be general members of the public looking for security, and peace of mind of their belongings and person when staying at a hotel. They may also be concerned about privacy of their information, and how it is distributed.

Situational Stakeholders

My project would best suit a hotel which takes in a high frequency of visitors, due to the organization of the database being able to input and output certain data very quickly. Thus a hotel with the stress of needing

to communicate a barrage of information, with a keyboard this could be done with much less stress to time constraints.

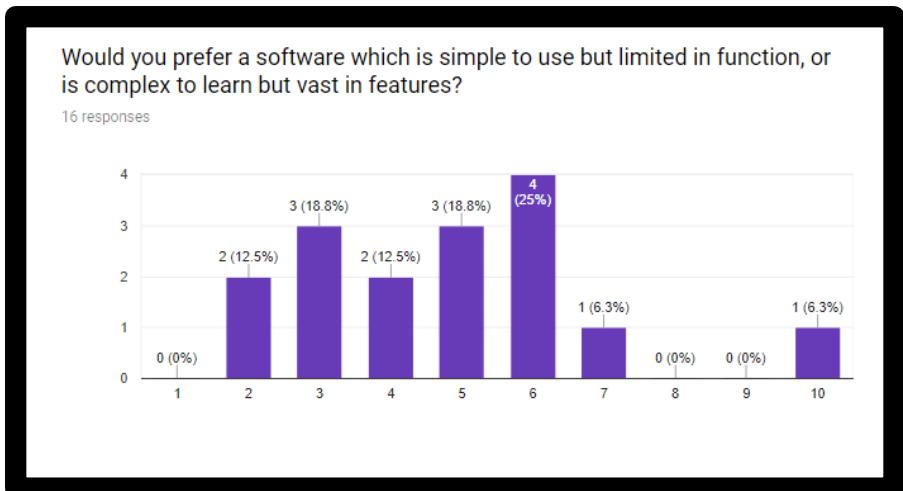
A hotel with security problems may also see benefit in the lock system of the software, with the management and lock systems connecting to provide a better peace of mind for guests by having more secure rooms, whilst also deterring crime by having strong security. With information of customers being stored in a database, identity theft and other types of fraud could also be deterred through the use of the management system, with data being recorded much more permanently on a hard drive.

A hotel which requires the use of certain peripherals for their members of staff (such as a staff member with a certain mobility disability) would benefit from the portability of the software, with it being able to input and output information all from one terminal, without the need to move. Also to those users which have difficult inputting information (Missing limbs or motor impairments for example) would be able to use a peripheral to access the software as it is ran on a computer. This would benefit to those moving from a paper-based system to a computer system. This compatibility with peripheral devices will need to be tested.

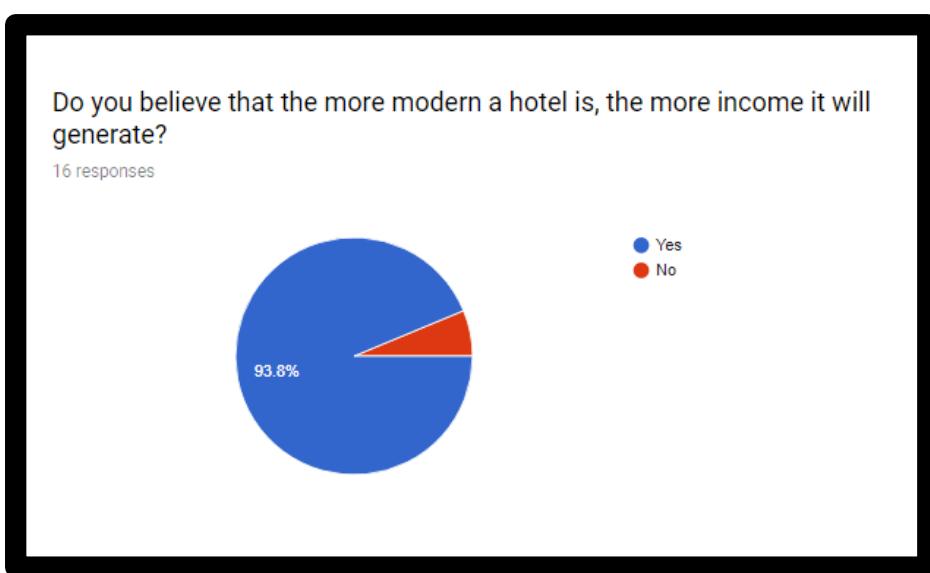
The specific stakeholders will be from a range of local hotel staff, where I will be giving them a questionnaire concerning the software already used by their company. I aim to have a set of results from at least 10 users, which will give me a larger range of results to decipher a pattern from, and from these results I hope to have better understanding of what the target market requires from their software. I will then act upon better; more reasonable, goals from these question results.

Stakeholder Survey Results

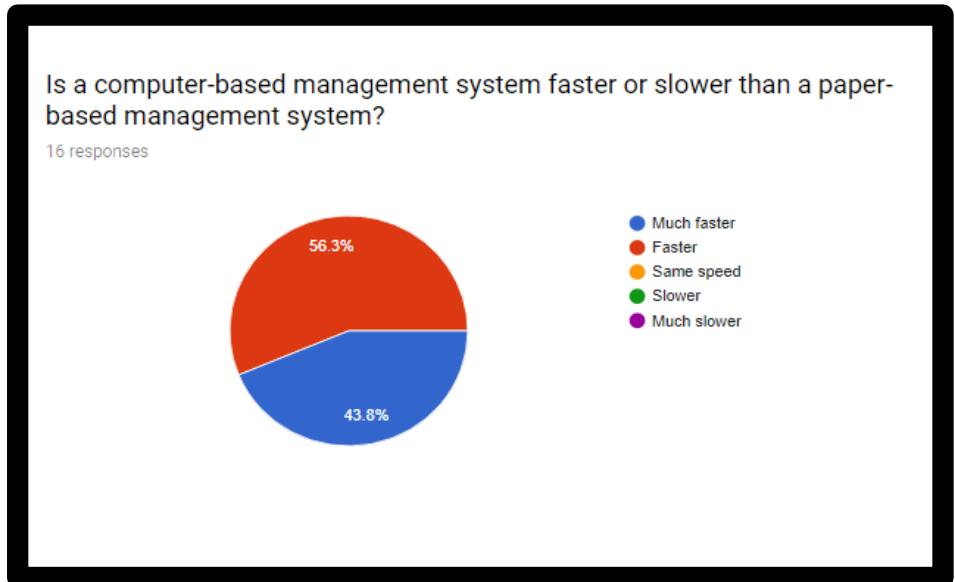
I assembled a survey which was heavily based on the discussion of stakeholders above, with many of the stakeholder descriptions being raised to attention to see if they matched the true audience. I was able to distribute the survey through family friends who worked in local hotels, as well as emailing multiple hotels across the country.



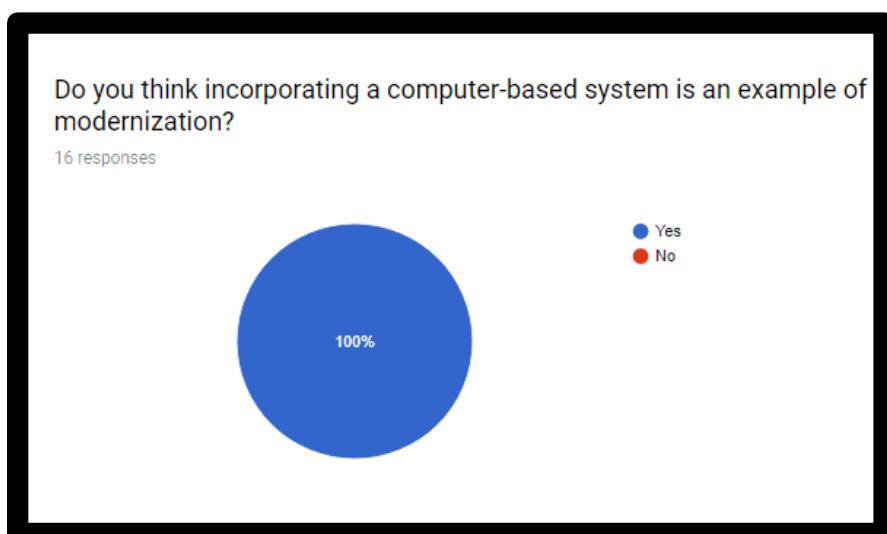
The results from this are quite broad, though the majority seem to be towards simplicity, though nobody chose to be fully simple. This seems to mean that the hotel workers care more a balance between simplicity and complexity of the programs features. From this question, I learned not to tend toward an extreme (where I was originally going to go extreme simple). I will create a rightful balance between these two factors, by inserting all the complex features but keeping it easy to learn, without bombarding the user with information on how to work the program on useless features. Features will always need to relevant.



The purpose of this question was to link how the modern aspect of the solution will be fundamentally beneficial to the hotel, and not just become modern for the aesthetic. The outcome truly shows that many hotels agree with this modern-aspect being important for their business, and this feature will not be a wasted vision for this project - modernization does serve a true purpose.

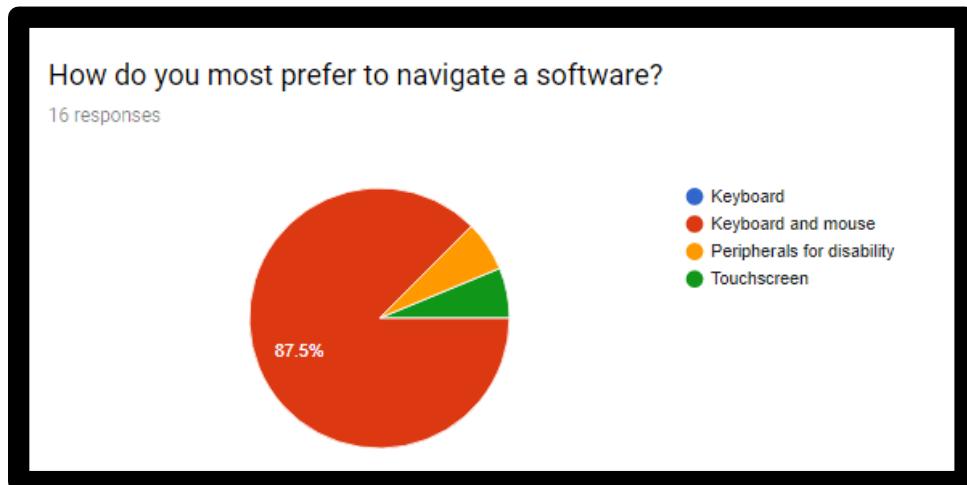


Quite simply, this question shows that there is no debate over the relative speed of a computer based solution, as nobody thought a computer based solution was slower than the paper based. This evidences the need for a computational method over a paper method, as it shows its efficiency in the workplace. The answers to this question also justifies my earlier explanation of how a computer solution is faster than a paper-based solution. This show I am working in the correct direction in terms of having a solution for the speed aspect of management in a hotel.



Quite clearly, these answers show clarity of the impact of a computer system in terms of keeping a hotel modern, as workers believe that a computer system would enable a more modern environment, thus linking

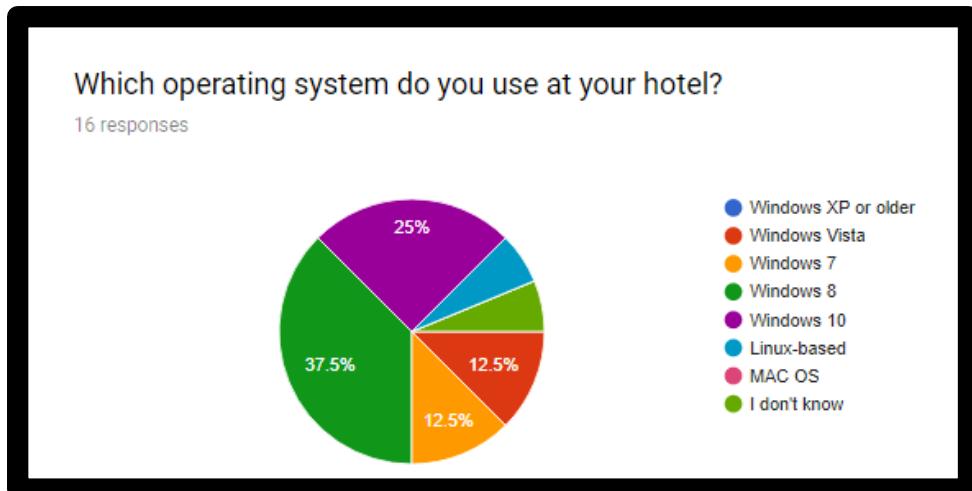
with the previous ‘business’ question to show that installing a computer system would have a correlation with hotel income.



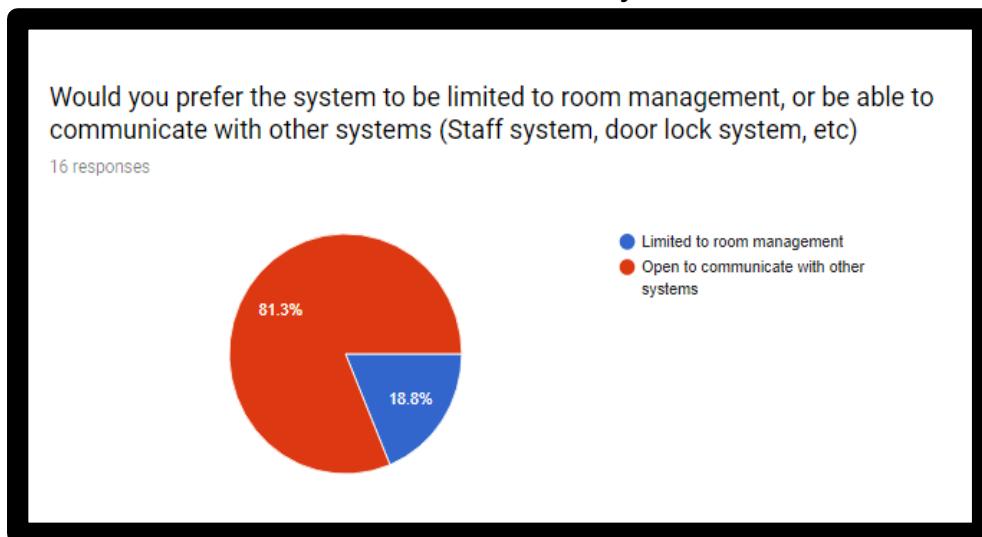
This question on peripherals was in the interest of seeing the range of hardware devices that will need to be compatible with the program, and to see which ‘priority’ peripheral will need to be used most extensively (in the result of this case, the keyboard and mouse will both need to be utilized fully to ensure the interest of the majority of stakeholders is met). I included ‘keyboard’ as an option as I understand some programs can be utilized solely by a keyboard peripheral, so I explored the interest in this usage - however there was no interest shown.

The one vote for ‘peripherals for disability’ shows that some members of staff may need special peripherals to navigate a computer, such as special mouses and keyboards. This vote brought to my attention that I will need to test a range of these devices to ensure they are compatible with communicating with my software.

The ‘other’ vote with the suggestion of a touchscreen would be a very interesting addition to the project and would harbour a better show of modernization, however this would be difficult to add as a feature due to my limitations (See the limitations section).

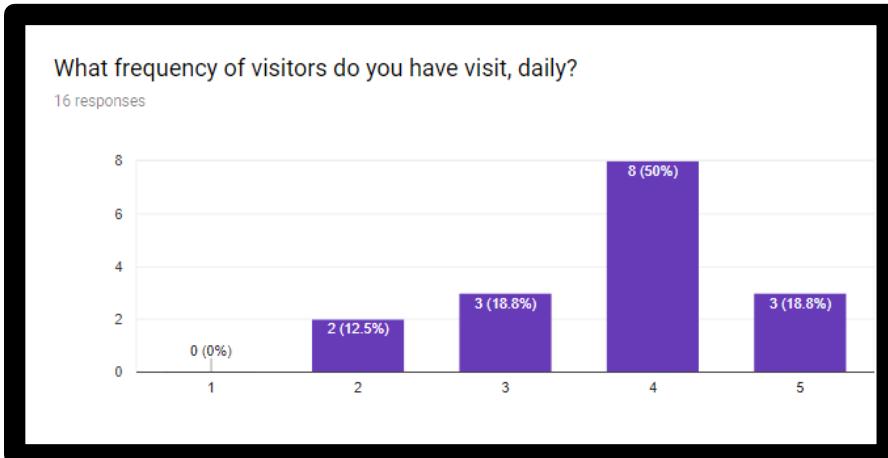


The purpose of this question was to see the range of operating systems that my stakeholders are using, to see which I will need to test the program on. The majority answered with windows-based operating systems from many generations, though there was a vote for a linux system. One vote was also for 'I don't know' which means they could be on a range of operating systems. From this question, testing will need to be made on many operating systems to ensure compatibility. Installation will also need to be tested on these systems.

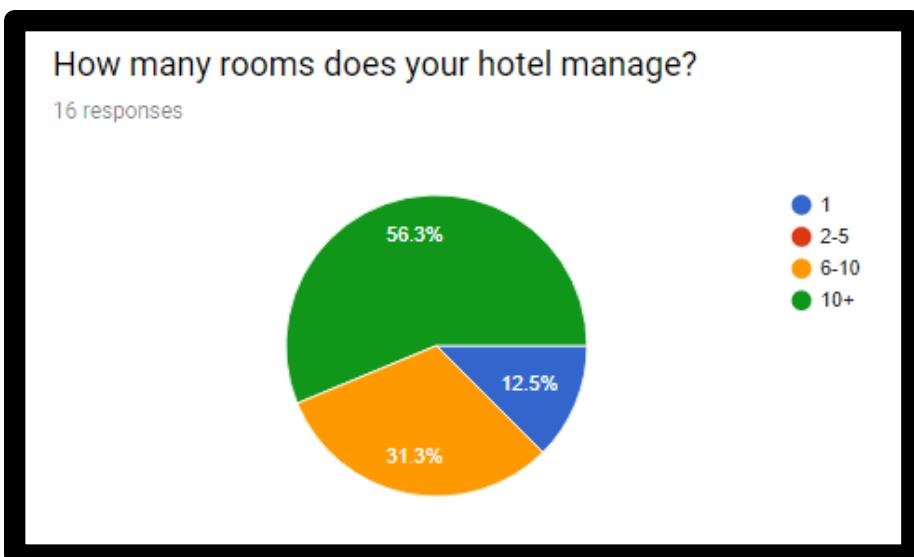


With the majority of votes being in favour of an embedded system, there is still a sizeable amount desiring a singular system for whatever reason (Perhaps security issues, or just the simplicity of keeping it singular). From this result it gave me the idea of giving the option of embeddedness upon installation of the program, as the lock system is not necessary to be connected to the management system. This option will be an advanced feature to work on, but would be better to please the larger demographics. For a basic feature however, I will develop the

code in favour of it being embedded with the lock system and the installation option will come after.

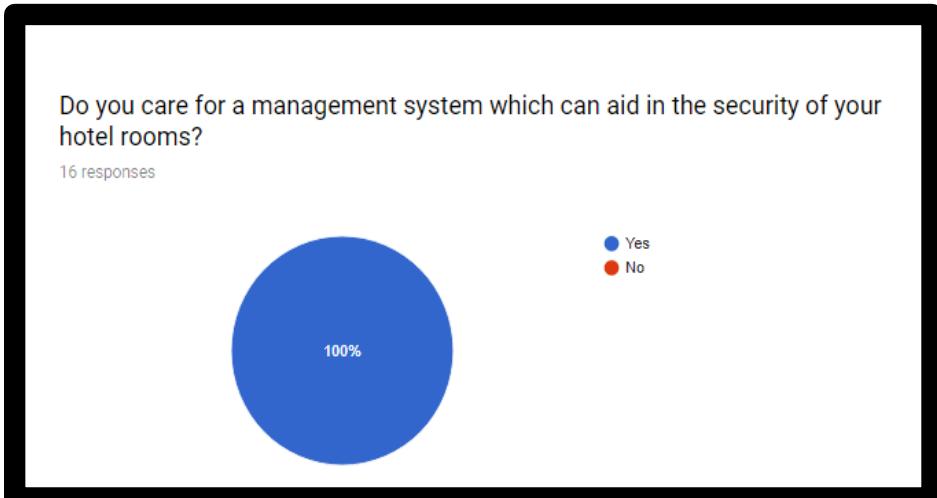


This question relates to my stakeholder description as having a high frequency of visitors, with my program giving them better organization for storing this high amount of information. Proven from the answer, many of my stakeholders do have a high frequency of visitors (As the scale of 1-5 was a relative scale of low to high, not specific quantities). This ensures that the stakeholders will benefit the most from my project's objective of organization.

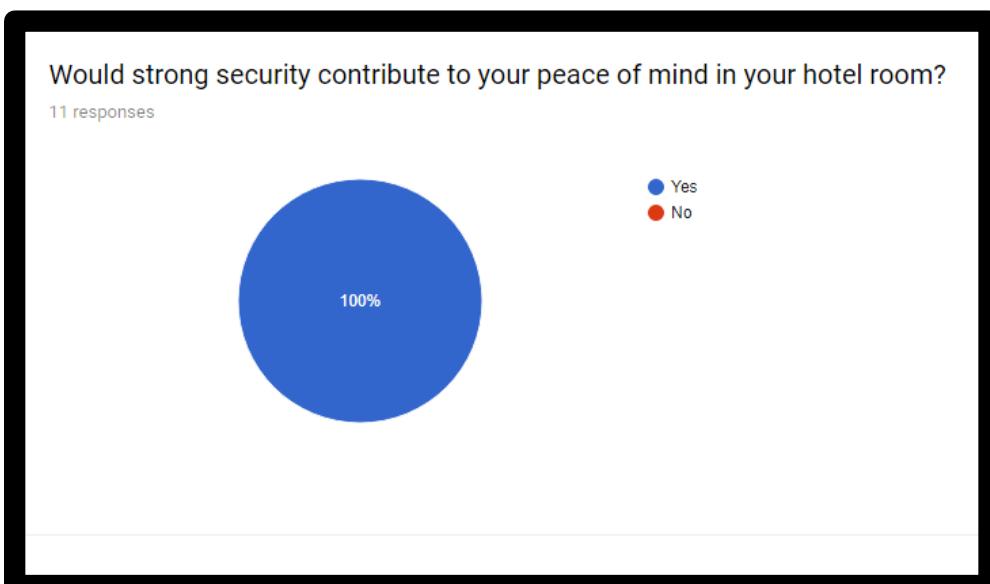


Once again this ensures I have the correct audience, with more than half of the stakeholders claiming to have over 10 rooms. This means they will benefit the most from the managing aspect of my software if they need to manage more rooms. As described before however, the people who answered that they rent one room will still benefit from the management,

but not as much as those who organize many rooms (which luckily, is the majority of the stakeholders).



Quite a bland question, though it means stakeholders find the importance of a system which can mingle with the systems in charge of security in the hotel, and shows they value security very much. The definite nature of these answers shows that security will have to be a prime objective when developing the code. Safeguarding was a feature that spawned from this answer, as it is still an example of non-traditional security that you would image. (See advanced features for information on safeguarding)



Asked to a group of people who have are accustomed to staying at hotels, this question had quite a foreseen response from the group. It shows that security is important to the visitors, and special measures will

need to be made to ensure security is maintained and invoked within the software.

Research

Preliminary Research

Before researching specific management software for comparison to my project goals, I wanted to reassure my goals with some research into what hotels strive for as a business.

Vitosha Park Hotel is a pristine example of a hotel with a reasonable idea of standards within the hotel industry. It is a four star hotel situated in Bulgaria, though what is of most value to this project is their publicly shown “hotel values”. These values cover what the hotel aims to achieve for their guests, and at what capacity they aim to complete the job at, as well as internal issues within the hotel. I will use a portion of these values as comparisons to whether or not the software is developing parallel with the corrects needs of a real hotel - these points will appear again within the success criteria as justification.

The three values I found most appropriate to my project’s goals were:

1. **“Attention to the needs of those who work for us”**
2. **“Straightforward, responsible and reliable management”**
3. **“Global vision and innovative approach”**

Also, researching into hotel ergonomics, keycard sensors are supposed to make a noise upon scanning a card to inform the user that they have correctly scanned the card, and to not otherwise leave the user confused.

Software Research

For software research, I will be looking at multiple solutions to the same problem I am solving (Room management), and describing the features they encompassed when producing their solution. I will then identify the advantages and disadvantages of each product - and how I could adapt these for my own product.

Innroad (www.innroad.com)

This software is an example for a basic managing program with no bespoke functionality for certain businesses - it simply provides the ability to manage rooms on a simple UI. It also includes other services such as the counting of stock inventory within the hotel (If the hotel serves food, for example) - this means it has a level of embeddedness.

The main advantage of this software in relation to my solution, is definitely the UI and how it displays room information. I had not thought of displaying information in this fashion previously, and had only expected to create a search for specific rooms to check their availability. However, displaying all of this information at once in one timetable negates the need for a search (unless there are many, many rooms) due to all of the information being available for viewing in one page. This UI

also relates to the organization goal of the solution, where information needs to be organized efficiently, and this UI seems to be the most efficient organization of data. Having all the rooms on one page and ordered alphanumerically also prevents the need for multiple page navigating to find a certain room - all the information relevant to management is presented at once.

The colour scheme is another aspect of the program I have not yet thought about, and innroad is an excellent example of how important a colour scheme can be. The use of single cell shade colours, and how they contrast deeply with each other allows visual clarity within the program, meaning the certain bits of information that need to stand out to the user, do stand out. Colour being used as part of a key also adds to this visual clarity by associating certain rooms or events with certain colours, and this separates the events for the user to easily search.

Colour paradigms investigation

I researched into the psychology of colours within hotels, and their effect on the minds of guests. Initially I expected there to be a perfect colour scheme to use in a hotel however, it appeared that colour is totally dependent on the style of the hotel (Which may sound quite obvious). For example, the colour purple indicates royalty and wealth - whilst brown indicates an old fashioned feel. As the management solution is not catered to a specific *style* of hotel, I will not deem colour scheme as important, and only a coherent set of colours will be used. Furthermore, the program could offer a colour scheme option as an advanced feature, as to cater to hotels that care about their style in all proceedings of their hotel.

Colour scheme is primarily used in the marketing of a product (For a brand image, for example). This, however, does not concern the solution hence colour scheme for marketing will be ignored.

Innroad (continued)

The search function shown in the image above the calendar UI is another feature that will aid my solution's organization nature. I like that Innroad has its search function to be very easily accessible by placing it

on the main screen, and the input of search seems very simple to use and learn. Using drop down boxes for searches instead of raw information input would validate the user's actions safely, and also be easier to navigate with pre-made options, rather than; for example, the user needing to type out the entirety of '12:00 bookings'.

I could not find any general disadvantages with Innroad, other than the questioning of the 'features to simplicity' ratio that I questioned within the stakeholders section. One of the goals of this project is to create a product which is simple to navigate and use - with minimal training. Whilst I would say the interface of Innroad is simple and easy to navigate, the 'overload' of features may be a problem to those just picking up the program. This is only a minor disadvantage (as it comes with the advantage of more features).

Innroad Summary

Advantages:

- Search function
- Colour scheme organization (Use of a color 'key')
- Ability to embed with other systems in the hotel.

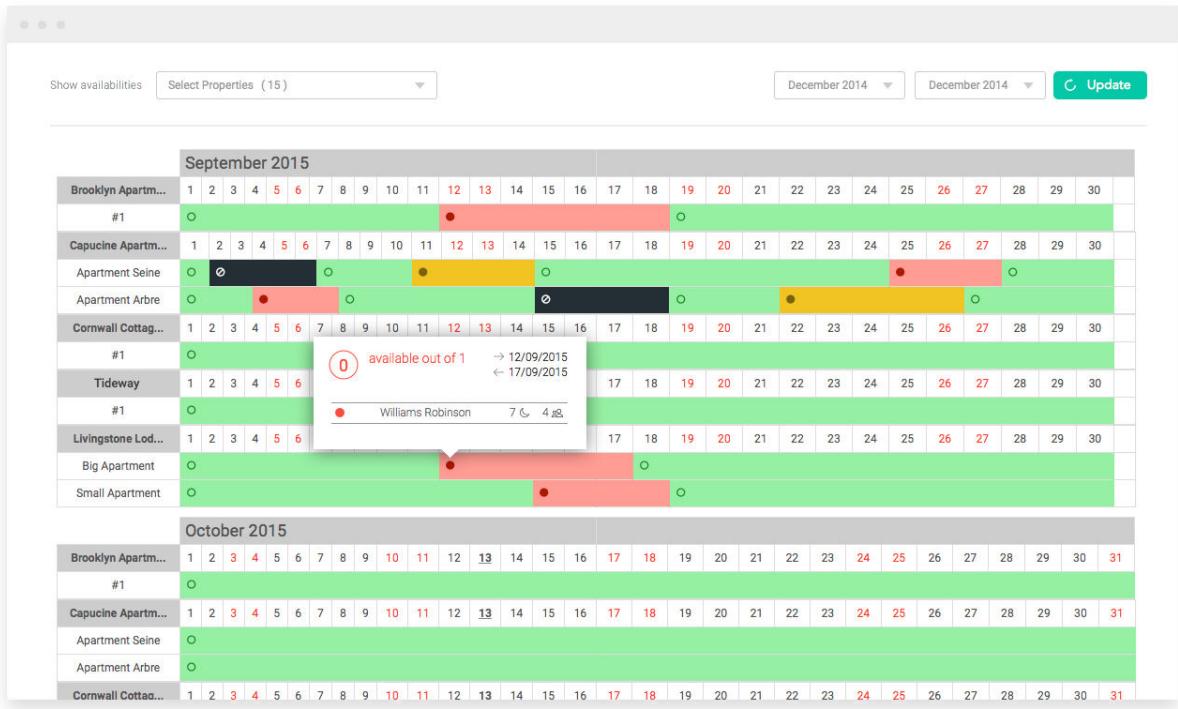
Disadvantages

- Overload of features affecting simplicity for some people

Lodgify (www.lodgify.com)

Lodgify is a web based solution to the management problem. From their website, the company is explicit in mentioning the system's simplicity: for both customers and staff.

Lodgify differs quite a lot from Innroad, in which Lodgify has a lot more of a focused solution into the management and booking of rooms whilst Innroad dealt with a broader part of the hotel system (dealing with inventories, etc). Furthermore, this focused solution by Lodgify is a lot more relevant to my solution - meaning the features of the program will be heavily beneficial to the project.



Due to the program being web based, this means that the program can be accessible through many different devices (such as downloading the app on your smartphone, and viewing the list of reservations from there). This accessibility would be great for a hotel manager that needs to see reservations from home, or perhaps an owner of multiple hotels. This advantage would be great for my solution due to the accessibility being modern, and taking the fullest advantage of modern computation - however, the project limitations would likely make it difficult to have the time to develop a web based solution. Although, I will list some sort of web-based access to the database as an advanced feature, which I will only implement at late stage development if I have the time.

Web based solutions however, have the disadvantage of security issues. This is because information is being transmitted outside of the local area of the hotel, this meaning that; given the correct force, any hacker could access the reservations information. Whilst there is specific service to protect from this, it is expensive and would be much cheaper just to maintain the information locally within the hotel. As this web based feature would be innovative for my solution, I am aiming for a simple program that is simple to implement, so a web based solution where an expensive security and networking system will need to be made does not match this simplicity.

Lodgify also offers online booking, which dissolves the need for a receptionist to input reservation data. The information is entered manually by the customer, and all payment methods and room times are inputted by the customer. In retrospective, this method is the maximum solution to relieving stress and maximizing simplicity for the hotel staff by needing to input room and customer information - by removing the problem altogether. Moreover, this advantage requires the system to be web based and aforementioned, a web based solution heavily pressures the project's limitations and would influence the quality of the code.

The service by lodgify also includes a site builder, which gives the hotel the freedom of creating their own online booking site for the customer. This total freedom is another advantage of the service, by allowing the user to use certain templates with their own colour scheme and photography to have their own bespoke website. I could add this feature to my previously mentioned colour scheme customization - though likely not as advanced as to provide templates and movable customization due to limitations. I could, however, use the idea of setting a user's picture for the background of the program as this seems quite easy to implement with it being quite important to the user's freedom and comfortability with the software. Also, allowing this image customization would give a bespoke feeling to the program with the hotel being able to input their own logo onto the program - purely for the aesthetic.

Lodgify Summary

Advantages:

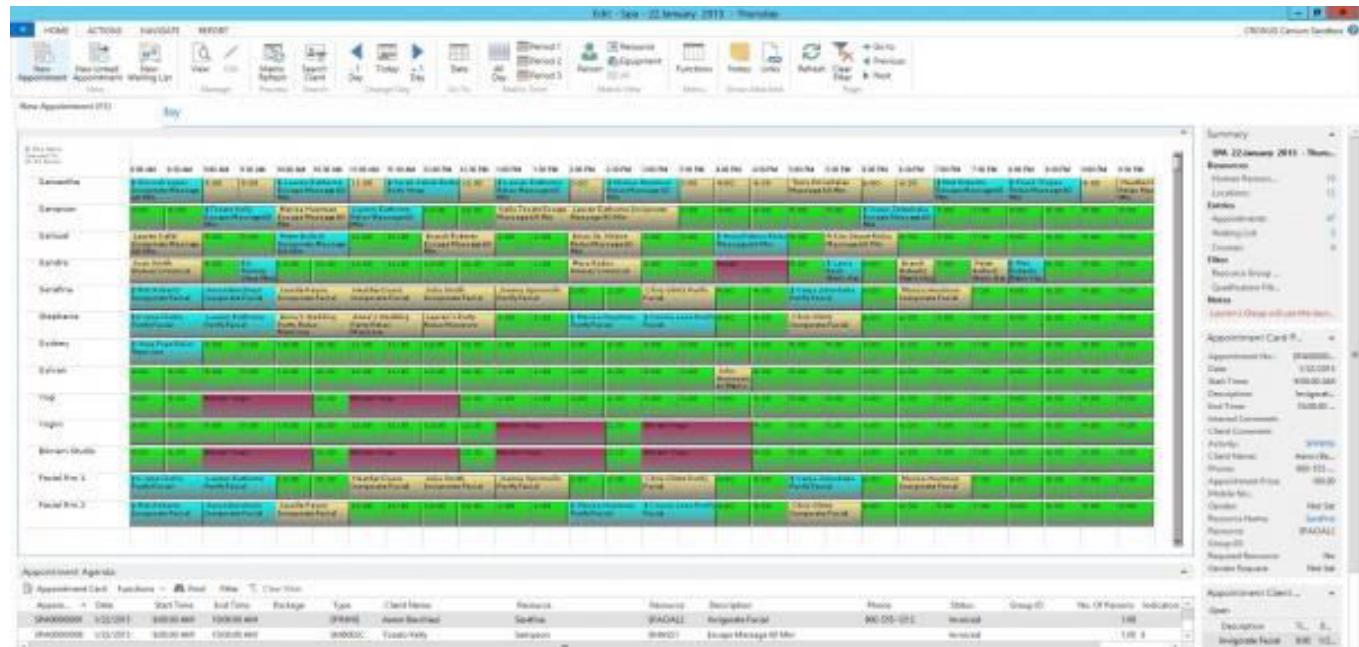
- Web based accessibility
- Colour and image customization.
- No need for staff to input reservation - online booking.

Disadvantages

- Complicated to set up system (Needs an entry form for both customers and staff)
- Expensive to maintain (Web servers, security, etc)

- Embedded nature seems limited due to it being web based.
- More room for error with it being web based - thus taking away from the simplicity of setting up.

Cenium Hospitality (www.cenium.com)

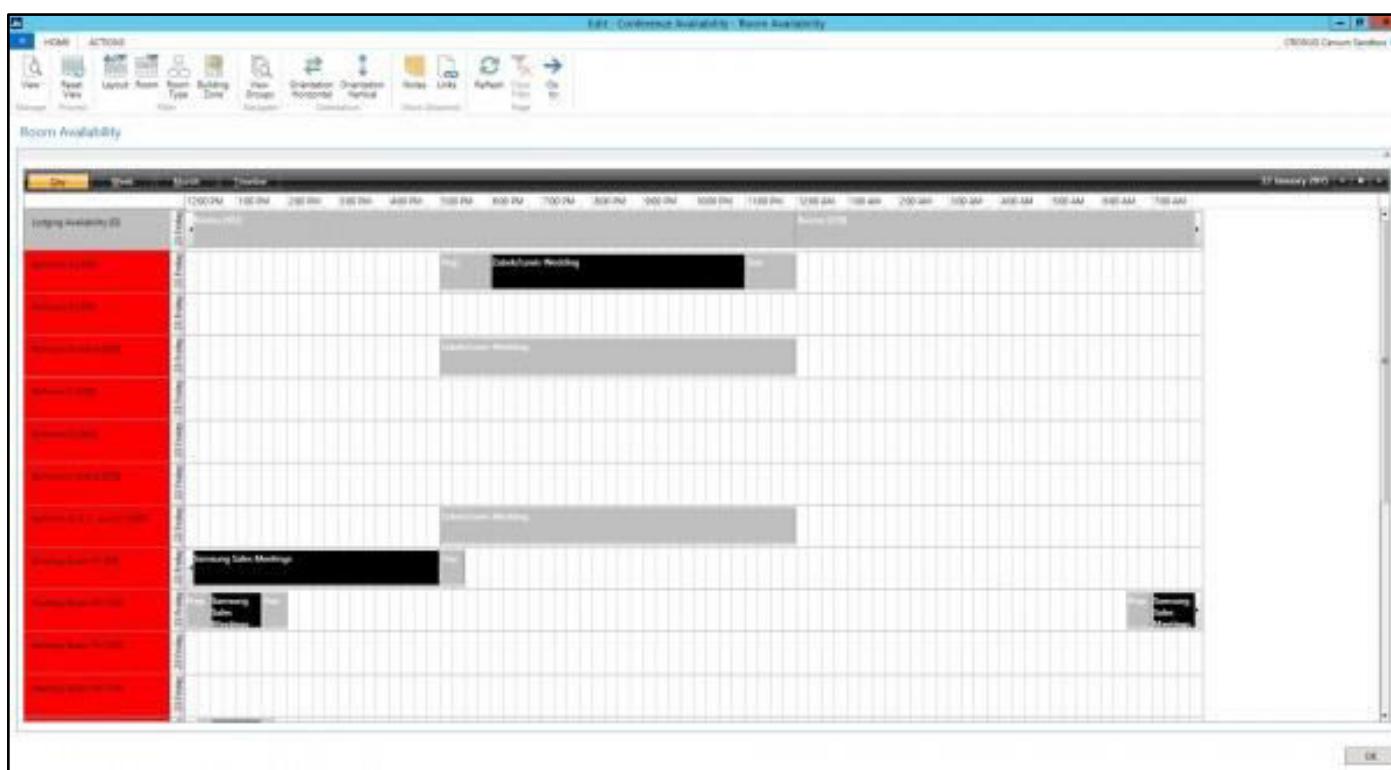


Cenium hospitality is an example of a simple solution which is a lot less popular than the previous researched softwares. From the image above, it is suitable to assume that less development has gone into this solution than the other; more professional, softwares. Due to the limited development on this product, it is a lot more realistic as to what I will achieve from my product - due to my project having many limitations to time and workforce power.

The program is dedicated to being a solution to management for hotel staff, and positively manages many parts of a hotel - not just the rooms. From the image shown above, it depicts the management of the spas and spaces throughout the hotel (described as the 'lifestyle matrix'). From this, it depicts when events are ongoing within certain rooms, such as a hotel hosting a party in one of its main rooms. This is a nice feature for more deluxe hotels, though the feature may be unwarranted by singular hotels that do not require the means to manage extra space - and my stakeholder research would suggest that my program would appeal to those hotels which appear to be smaller. From this, the feature

of managing space other than rooms would not have as much an impact as an additional feature within this project.

A negative point for this program is that the project caters to one style of hotel - the more ornate and deluxe. This would limit the software by being unappealing to a large branch of stakeholders, and (specifically for my project) would not run parallel with the goal of being simple and unspecific to a certain type of experience of person. From Cenium, I can therefore learn how to not cater to a specific demographic and be as permeable as possible to any user.



Shown above, Cenium's interface is quite special in that it gives accessibility to all configuration options on the main display. Alongside Innroad, this feature is deemed quite important as a factor to simplicity, with the option of needing to navigate many menus and pages to find a certain configuration would evoke stress and confusion in a user - whilst having all the options on one page and labelled with pictures makes navigating the software quite easy for a user of any experience.

The 'room type' option could be useful in my project by giving a bit more information to a room rather than just a number - especially if booking is being made on the same program. Room type as an option means the

user can label rooms depending on their size, distance from the ground floor, or if the room has a window or not. This would be important in selling to customers with certain needs, and would save the staff the confusion of needing to check a room first as all the information is saved on the program.

The ‘refresh’ option could also be important to a user where multiple users can access the same storage database. A tool like this would reduce the chance of a room being booked twice accidentally, due to the screen updating to the user. However, this feature may only be necessary if the system is planned to be held across some sort of network (either a LAN or a WAN), which is not initially planned for the outcome of this project.

Orientation options are also well used for customization within the system, as shown in the image. It allows for both vertical and horizontal orientation, which may include changing the layout of the booking information as to which axis the times and rooms are on. This is simply for the preference of the user, and could be used in my solution to contribute to the previous standard of *“meeting the needs of those who work for us”*.

Another small, yet important detail is the ability to navigate the calendar once the information is beyond the suitable page size. This program uses ‘sliders’ which can be used by clicking and dragging, or simply by using the scroll wheel. Visual studio offers this feature to use these sliders, and I will most likely be using them for their professional, and simple usage.

Cenium Summary

Advantages:

- Simple, concise interface which all useful navigation tools on one screen.
- A visual representation of room bookings, in a ‘time by room’ format.
- A broad range of configurations, with the freedom to label rooms with any details.

- Offers a range of ways to display the calendar information (Orientation, etc)

Disadvantages

- Is a solution which is too specific to a demographic.
- Offers features which may be unused by many users.

Features

Fundamental Features

- Initial Setup

All of the researched programs offered the ability; upon initial startup, to take in the relevant information of the hotel such as rooms, their sizes, and the hotel's name (for a bespoke aesthetic). This information can then be put into a blank database, ready for the inputs concerning the customer to be matched with the room - and for however long. This information can also be changed (in the event the hotel builds more rooms, for example), so that certain rooms can be added and taken out of the database. This initial setup gives a sense of direction for the program, and shows that the program is working on the management solution as soon as it starts up - thus showing dedication. Once entered, this initial setup will only need to be ran once, where the setup data can be restored upon restarting the program.

- **Ability to input room and customer information and indicate a price**

To seamlessly be able to book a visitor into a room will be the main purpose of the program. It will need to be especially simple and quick to input only the essential pieces of information, as to prevent annoyance to the booking client, especially if the visitor is booking by phone or through an online form. At this stage, I imagine a booking will only require the following information (though this is highly subject to change):

1. Length of visit
2. Size of room(s) required
3. Visitor's name and phone number
4. Special requirements

As well as retrieving this information, the program must then accumulate the room and visit length information to produce a price, which will be prompted upon startup as to how much the hotel charges per room and per night. A manual input of price will also be provided to the hotel user, in case of required discounts.

- **Ability to input room and customer information from a range of different input devices**

Without a doubt, this feature will be most fundamental to the management of rooms - and all of the researched programs had this feature. It is simply the method in which the information is received which matters in this feature, as my stakeholders prefer to use a range of devices to navigate a program, including devices necessary for those with physical disability. Hence, the program will be able to take inputs from a range of devices. The information will be passed through text boxes prompting text from a peripheral, and then submitted upon pressing a 'submit' button. This feature is also justified by my preliminary research standard of "*meeting the needs of our staff*" being important within the workplace, and this feature meets the needs of staff which require, or even just prefer, certain peripheral input devices.

- **A Graphical User Interface**

All of the researched programs used graphical user interfaces, as the use of a command line interface would be extremely outdated and over-complicated in respect to viewing and navigating the stored information.

The research candidates made the most out of the graphical user interface in that they all used some sort of colour scheme to represent the customer's information on the calendar. The use of colour will be very prominent in helping organize, as it will make certain elements stand out over other colours, thus aiding in helping the user identify certain elements faster than others. Colour and images will be used in the solution as well, such as using a picture of a grey cog to identify a settings menu. The graphical user interface will be used to the best of its potential to simplify information, so that a user may digest it faster. A GUI will be used for its means to communicate inputs and outputs in a more interactive sense using buttons and images.

- **Calendar Display**

This consists of using the GUI to display the room accommodation of the hotel, such as the exact dates which certain rooms will be occupied and who they will be occupied by. The calendar display will be display on a 2D array, with time being on the 'X' axis and rooms being on the 'Y' axis. This 'Y' axis will have its information taken from the initial setup. Colour will be incorporated into the calendar to separate information, otherwise the information can become quite monotonous visually without colour to make certain elements stand out. Before researching, the calendar was not even going to be included, however, every software uses a calendar form of organization and therefore it must be a standard for hotel management software, hence it would be most professional to incorporate it into development. I image this feature will be the most complex feature to design, and will succumb to most of the development time due to the complexity of the feature.

- **Saving and Restoring Data**

This is a feature that will need to be included for the program to reach the status of functionality, as the program would have no use if it was unable to save and restore the information inputted by a user from a previous session. All the necessary information can be saved within a database upon shutting the program down, and there will need to be a way to restore this database; loading it from a certain disk location that will need to be negotiated with the user, with minimal hassle. Without

this feature, users would find it much slower to use this computational method if data was lost every time the program was reset.

Advanced Features

- Keypad Scanning and unlocking upon validation

The lock system will have the ability to scan key cards, and take a coherent input to be processed from this scan. This information will then be compared with an appropriate value from the customer and room database to validate entry. This use of sharing information is justified, due to the stakeholders being in favour of a management system which can *communicate with other parts of the hotel system*, thus the information between the keypad and the customer database is a well tuned feature for my stakeholders. It is also the proof to be used to validate that the management system is embeddable with other relevant systems. However, after researching the prices of even the cheapest keypad systems, I have decided to label this feature as an advanced feature due to the chance of implementation being difficult due to expense. Moreover, it will still be vital to have some sort of cheaper, achievable system to replicate the keypad system as to show that the systems are portable (as this is an essential part of the success criteria).

- Safeguarding (Storing Unlock Attempts)

Any attempted unlock will be stored in a document, perhaps saved in the database alongside the keypad or room entity. There will be a general document listing every attempted unlock in the hotel which will tie together multiple pieces of information. The document would be expected to store data like this:

KEYCARD5	ROOM5 SUCCESSFUL	ATTEMPT: 15/5/2017
16:46		
KEYCARD2	ROOM3 UNSUCCESSFUL	ATTEMPT: 16/5/2017
18:54		
KEYCARD2	ROOM2 UNSUCCESSFUL	ATTEMPT: 16/5/2017
20:20		
CSKEYCARD2	ROOM2 SUCCESSFUL	ATTEMPT: 16/5/2017
20:24		

Above shown is how information would be stored, where each line stores a specific unlock attempt.

It will use the keycard primary key as to make sure the entity is successfully unique and identifiable. Shown on the fourth entry, 'CSKEYCARD2' would represent 'Cleaning Staff' and their keycard for room 2. This is an example of keeping entities identifiable, as the keycard can then only be linked to cleaning staff. There will be more entities for spare key cards. (Perhaps SPAREKEYCARD5).

This feature is primarily for safeguarding, as if there is a theft or a break in at the hotel then the information above can help in any investigation. It can be used to identify any suspicious activity with people accessing rooms, and to help rule out certain scenarios. The primary function of this feature is to maintain a safe environment for both staff and visitors to the hotel, and to ease the majority of stakeholders which believe security is extremely important to their peace of mind within a hotel visit.

- **Responsive 'Beeping'**

Responsive beeping involves having an audible response to a keycard being scanned, and such as a high pitched sound to indicate unlocking, or a deeper different sound to indicate failure to access. From my preliminary research, the impact of this feature is small yet useful for a customer, especially those with sight or similar disabilities, where seeing a light flash green may not be enough to indicate that their keycard worked. This is a small feature that is advanced simply due to the fact that its fruition is dependent on my ability to work with hardware, and hardware limitations may incur negatively on the use of buzzers to output information. The possible knowledge in electronics required to make this buzzer work is what makes this an advanced feature.

- **Colour scheme organization**

Giving the user the ability to change the colour scheme of the program was an idea from the colour paradigm research, where every hotel that uses the solution may have their own style of colours, thus I could not exactly set a specific colour for the program. I could have a multiple set

of colour schemes to choose from for the user (where they could choose one on the initial startup). I would not give the total freedom to the user of deciding the colour of every element, as this would be highly confusing and most likely a waste of time for a user, when they could just choose from a broad range of colour schemes. As mentioned in the calendar, I will need to make sure some elements are consistently high contrast, as to make them stand out to a user. On top of this, there could be a high contrast or specific option to those with colorblindness, as this would give the ability to organize easier to a larger demographic. This is an advanced feature as colour scheme customization is not fundamental to the completion of a working solution, it simply an aesthetic layer to appeal to more of my stakeholders.

- Search Function

The search function would likely be incorporated into the calendar system, and would give the user the ability to search a long list of rooms for certain features. With the search function, it would include the 'room labelling' feature (similar to that used in Cenium) to give more use to the search function by singling out certain rooms, by being able to search for rooms with a window, or a room with two single beds, for example. The search function is an example of simplifying work via a computational solution for the staff, and this would apply to my stakeholders who agreed that a computational solution is faster than a paper based solution.

Limitations

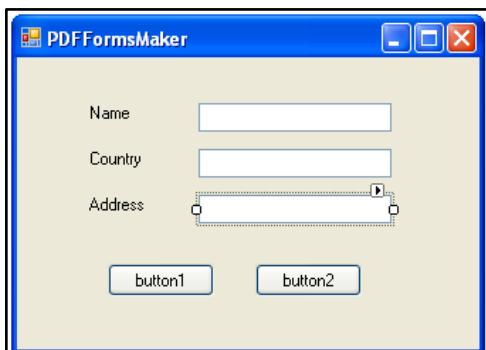
- Technology restraints

As I do not have access to the most modern technology involving RFID (Radio-Frequency Identification), this will limit the development of the product, by the management system not having access to some of the features of modern RFID scanners such as using smaller chips or detecting signals more efficiently. However, this limit will be quite small considering the scope of the project, as I will only require RFID technology for a simple keycard scanner which will detect frequencies at a fixed distance. This limit may embellish on future iterative developments of the project, however due to the project likely not

developing past a few simple iterations, this limit will have little to no effect on the outcome. I will minimize the effect of this limit by maximizing the use of features of standard consumer RFID scanners, to test to make sure they work at a suitable standard for the solution.

- Time restraints

With the project having about a year for development, this limits my ability to iterate multiple times through development to 'perfect' the project features, as when I finish one prototype of the solution I would professionally release it to a test group, where I would then record feedback - before planning and developing more solutions to those problems. Whilst this can be done on a smaller scale in my evaluation, a full scale iteration would take too much time. This limitation ties in with the workforce limitation, where I am the only person developing this solution - therefore I cannot split the evaluation and next iteration between two groups of people, as I can only focus on one aspect at once. This restraint has an effect on most advanced features, especially those where a new sector of knowledge will need to be learnt. For example, the ability to use the program via web access was a great researched idea for stakeholders, however the feature I deemed too advanced to learn within the given time frame - as I already have to develop an understanding in C# advanced arrays and databases. Only non-fundamental features were cut for time, meaning I can focus on developing the fundamentals of the project and any remaining time can be dedicated to advanced features. The appearance of the project will also be limited by time, due to the appearance (other than calendar colour) being non fundamental - and will likely result in having the appearance of the default Visual Studio navigation buttons, shown below.



A basic, default windows form. Minimal time has been used to develop this.	A more stylized form. Time and effort has been used to make it look much more aesthetically pleasing.
---	---

- Financial restraints

This limitation has links with mostly every other limitation, in that it restricts the project's potential in terms of materials. Materials could include the physical items used to test the project's workability (RFID scanners, test computers) and these have already been discussed to their limitation extent in the technology restraint section. Materials could also include the workforce working on this project, such as requiring money to hire a team to work on different parts of the project, and this would then minimize the time restraint. The point being, that financial limits affect many other restraints involving the project, though the solution is very much achievable on a minimum budget. Sacrifices will need to be made in terms of technology, though the technology will be tested to meet a certain standard for the solution. Whilst finance was not the main issue, the feature of web access would be somewhat expensive to run properly - with a domain needing to be purchased and the website needing to be hosted with a maximum uptime, this would grow to be expensive. Hence, the feature was dropped from development. Also, after researching the prices of RFID scanners, they are highly expensive and highly against the limitation of finance for this project, although there are cheaper alternatives. This financial limitation affects the quality of scanner, and therefore may produce unsucces on certain parts of the success criteria.

- Workforce restraints

Lacking in workers dedicated to developing the project is a limitation. In a usual project development, groups of people with certain skills would be organized to develop certain functions, once the problem is decomposed. However, as I am solo in the development this means I cannot easily split my efforts between functions, and; in reality, would most efficiently spend my time focusing on one function at a time. Arguably there is a positive to working alone, as I do not need to use time organizing a group of people, and the image of the final goal does

not need to be communicated between peers - though in ultimatum it is much more of a limiting factor in terms of getting functions finished. Features that require a different set of knowledge such as the web based solution would coordinate much better if there was a larger workforce dedicated to the project completion.

- **Experience restraints**

My lack of experience within a hotel environment limits the potential of the solution, as I have not experienced the specific needs required within the environment, and may miss the opportunity to bring a solution to these needs without experiencing the problem myself. Whilst I have used stakeholder questionnaires as an alternative source of information for empathizing with the staff's problem, this is still not as effective as experiencing the problem myself. The specific features which I have lost in response to this limitation is unknown, as there may be an indescribable problem I have not realized beyond what the stakeholders have described. However, the negative effect from this limitation is marginalized through asking many questions to stakeholders, and to best empathize with the management problem as possible.

- **Demographic restraints**

As mentioned before, the fact that the many hotels within the hospitality industry like to differentiate in terms of style, makes it difficult to cater to a demographic of just 'the hotel industry' without making the product too simple or bland. For example, during research, the Cenium program caters a feature for those hotels which have spaces to rent out. Whilst this feature is useful to some, it may be a waste of time and only a source of confusion to some hotels which did not require the program to manage venue, just hotel rooms. This difference in style makes it difficult to produce a 'perfect' solution which pleases every single stakeholder - as even the stakeholder responses show a range of opinions on certain ideas. To minimize this limitation, I have decided to give as much choice to the user as possible. Examples of this include: colour customization, choice on allowing the program to embed (for privacy issues) and the orientation of the calendar.

- **Skill restraints**

Currently, I only have a basic knowledge of C# - which is not to the same level in which I hope to code to complete this solution properly. Some features are not related to C# at all, such as the databases element which; while still developed in Visual Studio, will need knowledge of SQL to fully manipulate the databases - especially for searches. Most skill can be developed for these needs, but alongside the time constraint this may become increasingly difficult for more advanced features. For example, I currently have little knowledge about smaller features of C# such as how to develop an array for the calendar, and whilst I will need to develop this understanding for the completion of the project, my skill limits developing the most efficient function for this array problem - and there will likely be many errors to fix throughout this stage of learning.

Requirements

Software

IDE

The completion of this project will require access to an IDE to develop the solution in the high level language, C#. I chose the IDE 'Visual Studio' to develop the code within, as C# can be utilized on the IDE, and it is also the IDE I am most confident with. Using Visual Studio will allow me to develop code the most efficiently, though without the complexity of the code being limited as it offers a wide range of features to aid in code development (such as stepping, variable checking, etc). To gain access to this requirement, I will simply need to download the program onto a computer with the correct requirements (see hardware below), as the program is free to use. Visual Studio also supports the use of databases, which is a fundamental requirement to complete the project.

Operating System

As Visual Studio is the only program I will specifically require to create code, I will just need a correct operating system to ensure I can run Visual Studio correctly and to its maximal potential. From the website, the list of supported operating systems are as follows:

- Windows 7 SP1 (x86 and x64)
- Windows 8 (x86 and x64)
- Windows 8.1 (x86 and x64)
- Windows Server 2008 R2 SP1 (x64)
- Windows Server 2012 (x64)

- Windows Server 2012 R2 (x64)

After researching, Windows 10 is also supported by Visual Studio, the website was just simply outdated to the recent release of Windows 10. I have already reached this requirement, as I have access to a home computer which runs on Windows 10 (x64), and computers at school which all run windows 7.

Web Browser

I will require a web browser for the research side of development, which I will need to use to access a range of websites which may aid in developing my knowledge on parts of the solution I am not as confident with (such as databases). My preferred web browser is Google Chrome, which I will have access to both at home and at school. The web browser will be a supporting source for the project's development.

Hardware

Computer specifications

The minimum hardware specifications required to run the 2013 version of Visual Studio are as follows:

- 1.6 GHz or faster processor
- 1 GB of RAM (1.5 GB if running on a virtual machine)
- 5 GB of available hard disk space
- 5400 RPM hard drive
- DirectX 9-capable video card running at 1024x768 or higher display resolution

These specifications are very much achievable in modern consumer computing, and I will have access to computers with these specifications.

These specifications also meet the requirements needed to run Google Chrome, where Google Chrome only requires a correct Windows version to be ran (7,8,8.1 and 10). Therefore, the requirements for Visual Studio is the minimum requirements for the entire project.

Web Access

I will need physical access to the internet, as to meet the requirement of needing a web browser. I could achieve this through having the necessary hardware to access the internet (router and necessary CAT

cabling). This requirement can already be met, as the computers with the correct specifications already have access to the internet via appropriate hardware. It will just need to be ensured that this connection is maintained over the length of development - as to consistently have access to resources for knowledge.

RFID Scanner

An appropriate scanner of reasonable quality will need to be purchased for the completion of the success criteria, as to ensure the solution can be embedded. Researching the prices of scanners, they are quite expensive and are against the limitation of finance on this project.

However, there are cheaper alternatives which I may have to pursue. The scanner will be required to be able to recognize different key cards, and not just take a single boolean input from a keycard (scan or not scanned) - as the security aspects relies on each keycard producing a unique input to the system.

Success Criteria

- Must be able to take coherent information from at least 5 different input peripherals, and 3 of those must be bespoke input peripherals designed for those with a physical disability. This is to ensure the program will be compatible with many of my stakeholder's input devices which will likely be just a keyboard and a mouse, however other devices must be tested to reach the standards of the entirety of the stakeholders (as one stakeholder uses a device designed for those with disability). I am choosing to test 5 navigation devices as that is a broad range in terms of input devices, to successfully come to the conclusion as to the program's compatibility.
- Must use colours and images effectively to aid in the simplicity of navigating the program. This will be tested by giving an inexperienced user the task of navigating to a certain screen within the program, where I will then ask for a score out of 10 on how simple it was to reach that destination. I aim to quiz 10 people minimum, and for the mean score to be above 7 to prove the program is a simple environment to learn to navigate. The score is

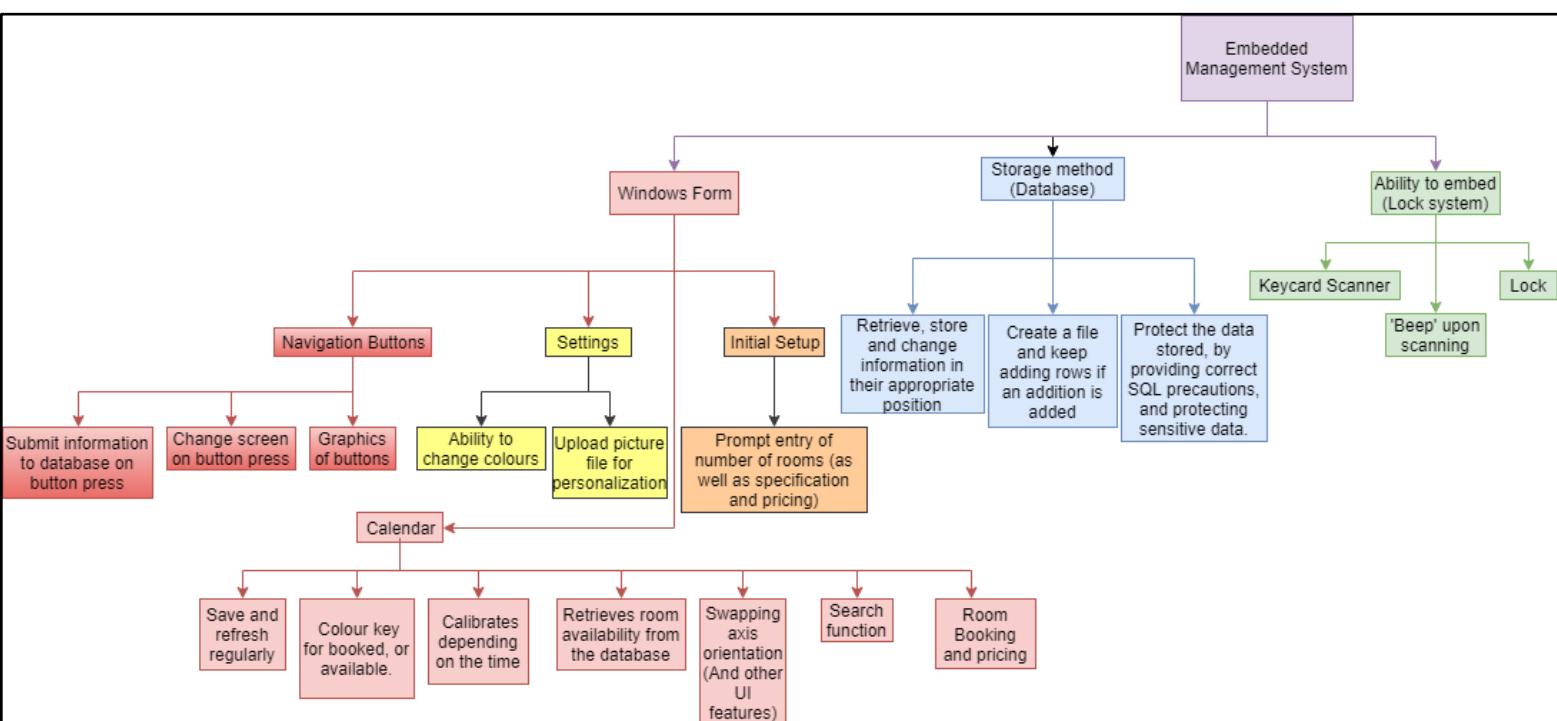
above 7 as this is quite a high number for setting a standard, though not too high where the value is unreachable.

- The program will need to be able to communicate with another system to prove it is embedded, therefore if I am able to send information through the lock system, prove that the information is passed and compared through the management system, and then an output is made via the lock system, then I will have proven that the management system is able to embed with other systems.
- The RFID scanner must be scanning to a professional standard of efficiency, and this will be tested through swiping the card 10 times across the scanner at a similar speed each time. I will expect the scanner to detect all 10 swipes. I have chosen to desire 100% efficiency out of this system, as it is a standard among RFID scanners to be that highly responsive. Anything less than 100% for scanning would be mediocre in terms of scanners, and could cause a problem to regular guests where scanning failures may become stressful to them - highly insulting to the hotel's code of standards concerning customers.
- The program's calendar feature must be simple to understand to somebody with no prior experience with the program. I will test this by setting up a test situation with rooms on the calendar, and then asking a random person to tell me whether or not a room is available at a certain time. I will time how long it takes for the user to find this room and answer the question just by viewing the calendar. From questioning a range of at least 10 people, I aim for a mean time of at least under 5 seconds. Under 5 seconds shows clearly the type of simplicity I am aiming for, by using colours and clear labels to show what spaces are allocated and to which rooms on the calendar. If a new user is able to navigate the calendar in under 5 seconds, then I would deem the UI to be successfully simple to navigate visually.
- The user must be able to successfully book a room to a customer, and input their information without confusion. I will test this by giving a random person a customer's information, and the room they wish to book and for however long. From the booking screen, the user should be able to input all of the information successfully into the correct input boxes, and able to allocate the correct room

and times without help from anybody else. If 10 consecutive testers are able to input the information successfully without making any mistakes then I will deem the simplicity of the booking system to the correct standard. I aim for 100% efficiency on this success as inputting information into the correct boxes should not prove to be confusing for a user, and if the format of the input form is so confusing that they input the information incorrectly, then this is a sign that the UI will need to be reworked to be simpler to follow.

- Searching should be an accurate feature, and this will be tested by searching for 10 room numbers within the calendar, and if the search is able to find the place of every room searched on the calendar then I will deem the search function as a success.
- Input validations will also need to be tested, such as boxes that are only expecting a number (such as the 'age' input on the customer information). I will input an example of each data type into the boxes that are validated, and use black box testing to input these types and evaluate the output of whether or not the character is inputted, or rejected.
- I will need to use black box testing on the entire system - from the view of a customer. This is to test that the main function of the system is operational, so I will simply test a keycard that is expected to be rejected, and a keycard which is going to be accepted. I will then scan each card and record their outcome on the lock. If both the outputs are as expected (Lock for the first and unlock for the second) then I will deem the primary customer function of the system successful. Using black box testing for this is important, as it is as though I am empathizing with the customer to make sure that the system is working coherently with their actions. It will ensure that the expected outcome is always occurring when a keycard is scanned. Black box testing will also be prominent with testing the pricing of hotel rooms, where I will input a number of room sizes and work out how much the total charge will be, which I will hope the program correctly calculates.

Design



Problem Decomposition

Windows Form explanation

Navigation buttons

- Submit information to database on button press

The buttons will need to take a click to be activated, where they will then call on the submit function. This submit function will need to take the information stored within the appropriate box(es), and transfer the information to the database system, which will be awaiting the information and how to handle it. The submit function will also need to transfer a form of ‘metadata’, as to explain what the information being transferred actually is.

- Change screen on button press

The orientation of the program’s graphical user interface will change on certain button presses. For example, a button which indicates that it will give a ‘settings’ menu will be pressed, and a function named ‘Settings’ may be ran, which would either bring up a new window which allows the user to access the settings menu, or completely change the program’s main screen to a settings menu. This is vital for user navigation, and is an example of maximally benefiting from using a GUI. This will work by running an appropriate function upon pressing a button, which will initiate a new form to be opened with the relevant code inside that function.

- Graphics of buttons

All buttons used by the software will need to have appropriate visuals and sizes, in relation to the entire interface. As justified previously, using images on the buttons would make for a more modern and professional touch, instead of just labelling them with names. They could be both labelled and pictured, to attain the best level of simplicity for the user. The buttons will all need to be sized equally, or appropriately different to one another - and also parallel and in phase with each other on the software. This is to drive the software away from a ‘tacky’ look. I will either upload a ‘skin’ to the buttons which I have created using external graphics software, or the default button design options will be used in Visual Studio.

Settings

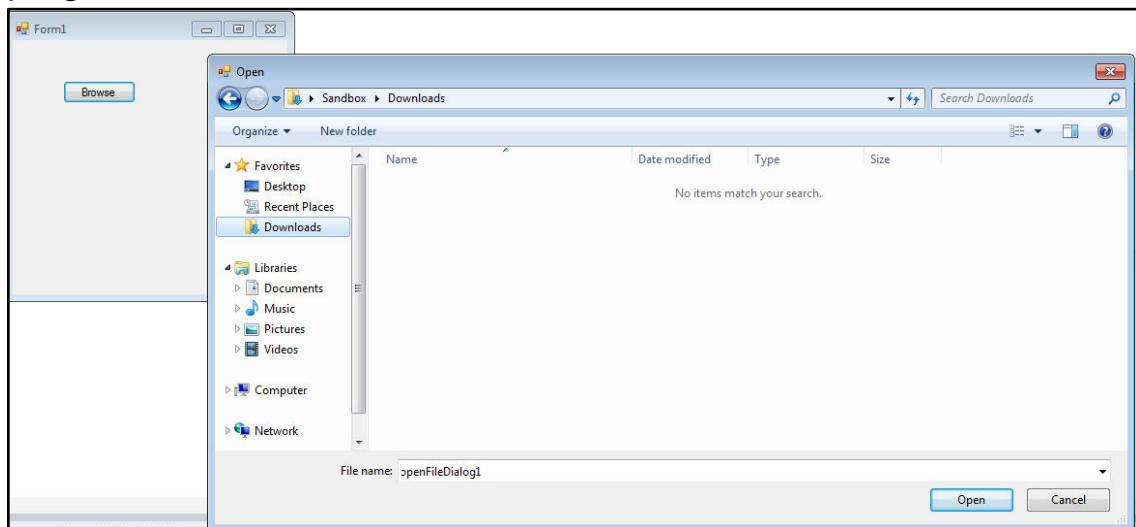
- Ability to change colours

In the settings menu, the user will be able to edit certain colour aspects of the program. This includes the main aesthetic of the program, where I will provide a range of colour schemes - to maintain simplicity in this customization. This colour scheme will be effective primarily for the background, though will use different shades for the calendar function to

match these backgrounds, yet successfully stand out to the user. When selecting a colour scheme, the schemes will be described and a small colour sample will be shown to the user, simply for making a decision easier without needing to switch between the schemes. I will make this work by colouring the program upon every startup using a variable (such as if the variable is set to “BlackAndWhite”, it will follow the correct path to colour the program correctly.) By creating a few of these variables with their own designated colours, this will be like having multiple templates for the user to choose from (which matches the success criteria of personalization).

- [Upload picture file for personalization](#)

This is another feature of the customization in the settings menu, which will allow the user to upload an image via a default file uploader provided by Visual Studio (shown below). It will have a prompt ‘Browse’ button which; when pressed, will establish a menu for browsing the user’s files. When the correct image is found, they can then upload the image to be used as a personalization for the program. The image will be saved in a specified directory alongside the program files, and will be retrieved upon every startup. There will be a validation if the file uploaded is not an image file, or if the file is too low of a resolution, to prevent confusing the user as to why their file is not being used. I will make this work by taking the file, and storing it in a designated area to be displayed on the program.



Initial Setup

- [Prompt entry of number of rooms \(as well as specification and pricing\)](#)

Upon setup, the user will be prompted to enter the number; and specific names, of the rooms in their hotel. This will be sent to the database to form a basic foundation to what needs managing within the hotel. The user will be allowed to remove and add extra rooms from the calendar view. The user can give specification to certain rooms as well, such as having an input to say that “Rooms 1-10 have balconies” (for example). This will be stored in the database alongside the rooms, and will be used primarily with the search function. For each room entity, a row will be created in the database. This function is going to only be ran once upon the first time the program is ran. If a database can be loaded from a previous use, this function will not be ran.

Calendar

- Save and refresh regularly

This feature is primarily to ensure that there is no overwriting problem within the program, as if there are multiple terminals to access the database then it is possible that two users may save information to the same room at once - which would become a problem for two customers expecting the same room at the same time. Hence, an automatic updater or a refresh button would serve useful to protecting from this problem. The function would simply take a new ‘image’ of the database and display it, until the next image is requested. I will also call upon the refresh function to take this updated image upon every return to the main menu of the program, hence keeping it up to date.

- Colour key for either booked or available

This means the certain rooms displayed on the calendar will have a distinct colour as to whether or not the room is available. A simple example may be that the room between July 15th and July 16th would be red to indicate it is occupied. The key of colours will need to be displayed by the calendar, for simplicity of recognition. I will do this alongside the refresh function described above, where there will be an ‘if’ statement for each square of the calendar, and pseudo code would follow:

```
if(Date is unbooked) {Make White}  
Else if (Date is booked, but visitor has not signed in) {Make Red}  
Else if (Date is booked and the visitor has signed in){Make Green}  
Else if (Date was booked and the visitor has signed out){Make Yellow}
```

- Calibrates depending on the time

Working alongside the refresh function, the calendar will need to update in real time to accommodate for what the calendar is displaying. The start of the calendar display should always start at what the actual current day is. This makes it easier for users, as booking information from the past may be irrelevant to what they are looking for. This means the function will be actively running and ran upon every refresh, by taking the system time and ensuring the necessary information is then displayed when an ‘image’ of the calendar is taken. Just by taking the date, this can be searched for in the calendar as to ensure that the calendar starts on the current date, to make it less confusing for a new user trying to interpret the calendar.

August 2014							September 2014						
Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So
				1	2	3	1	2	3	4	5	6	7
4	5	6	7	8	9	10	8	9	10	11	12	13	14
11	12	13	14	15	16	17	15	16	17	18	19	20	21
18	19	20	21	22	23	24	22	23	24	25	26	27	28
25	26	27	28	29	30	31	29	30					

Oktober 2014							November 2014						
Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So
		1	2	3	4	5	3	4	5	6	7	8	9
6	7	8	9	10	11	12	10	11	12	13	14	15	16
13	14	15	16	17	18	19	17	18	19	20	21	22	23
20	21	22	23	24	25	26	24	25	26	27	28	29	30
27	28	29	30	31									

- Retrieves room availability from the database

This function will likely be called upon when refreshing, and will take the information of when the rooms are unavailable (where it will then have the information of when they are available). Processing this information back to the form so that it can be displayed is what this function will be responsible for, as initially I believe it would be best to use an array of booleans, with true for available and false for taken. This will likely be done by using the date of the earliest booking and the latest booking, and then knowing that these will be the limits to iterate through when

updating the calendar, as to not iterate through unnecessary information - thus saving time for the user.

- Swapping axis orientation (And other UI features)

This is a simple UI option which will give the user the ability to display the room availability however they prefer. For example, if they prefer the calendar view shown above (with a calendar for one single room's availability) then they have the option to view it like that, though if they wanted to see the availability of more rooms at once then they could view the many rooms on an 'axis' sort of view (X being time and Y being rooms). Swapping these axis is an example of a feature of having full control over the display of data, which liberates a user's preference. If the calendar can be created, then it would not be difficult to create a new calendar but by swapping the axis, and swapping it out with the original calendar.

- Search function

Using the previously mentioned 'room preferences', the user will have a search display alongside the calendar to find certain rooms. The search will take information such as 'keyword preferences' where the user may input something about the room (A balcony, for example) to search for. Also, they may want to limit the search to a range of rooms (Only search between rooms 1-10 for example). The general boolean preferences will be an option as well, such as the user needing to tick a box to ensure they are looking for a two bedroom room, for example. Hence, the search function will collaborate with the '**Initial setup**' problem solution, by using the user inputted information about the rooms(Size of room, balcony or not, etc) to meet an itinerary, as decided by the search function. The search function will need to access the database (perhaps using an SQL search in the process) to compare this search input information with the information stored for all the rooms. Specific string comparisons may need to be made, though this would induce problems through slight syntax not matching - therefore a system which limits the user's choice to either boolean or limited integer inputs would be highly preferable for the ease of development.

- Room Booking and pricing

There will be an option alongside the calendar to 'book a room', which will produce a form with multiple input spaces for the customer requirements, which will work with the search function to find an

available room with the correct length of time suitable for the customer's visit. In the database, the different rooms will have a user-set pricing standard depending on their size, as well as needing to be multiplied or having an addition depending on the length of the customer's stay. It will need to be a quick process, as mentioned within features, the information will need to be quick and simple to process, hence the search function will need to be quick to activate and to find the price of the visit quickly. Some sort of receipt system, or storage of billing will need to put in place alongside this feature.

Database Storage explanation

- Retrieve, store and change information in their appropriate position

Storing information means the database will be able to take inputs from the C# form, and organize them appropriately in a table of entities. For example, there will be a table to represent the room entities, and when information about 'room 1' is inputted within the form, it will be sent to the database and either placed in the table for the 'room 1' entity if the room for the information already exists (general information such as room size or bed sizes), or create a new column if a new form of information is added by the hotel staff (more vague room information, such as if the hotel provides internet or mini fridge access to only some rooms).

Retrieving the information means that a request will be made by the form for a specific set of information from the table, and when this request is made it will simply make a copy of what is in the database, and return that information. An example of this problem will be the search function, where a request for information will need, though this information will not be direct database placements, it will be information to search for within the places. It will likely require iterating through the database to find a 'match' to this information, before then returning this information and its matching entities, as described above.

- Create a file and keep adding rows if an addition is added

A database file will need to be created upon installing and running the program, which will be defaulted to store the appropriate information about rooms, keycards, and visitors. The database file will need to be relational for these three key entities to be best represented, and for them to best interact and share information with one another. The main problem will be storing the information, where I will need to ensure that

the packet sent from the program has both the information that needs storing (visitor name, room size, etc) and the information on where the information will need storing, or what the information corresponds to (similar to meta data). Adding rows will be important for when a new entity is needing to be represented, and a new row of spaces will need to be created to store this information. The prompt for creating this new row could be detected if the user is on the form for submitting a new entity, otherwise (if information is to be edited in the database, not add) then this ‘new row’ signal will not be sent, thus not requiring a new name for the entity, etc. In short, I will need to create two separate forms - one for inputting new information into the database, and one for editing already existing information into the database.

- Protect the data stored, by providing correct SQL precautions, and protecting sensitive data.

Shown through my stakeholder survey, security is an important issues that must be address. Whilst a large portion of security risks will be from the physical keycard mechanism (to keep people and items safe in rooms), another security risk will be the risk of a privacy breach through intercepting the database of visitor data. As the system is going to be based offline, this will make it much easier to contain any possible breaches of data. I will need to ensure that the ability to hack the database is not possible via SQL inputs within open input text boxes for the database. For example, if there is a text box for the user to openly enter any string of data, a hacker may be able to abuse this liberty and use SQL language to enter and obtain and change information from within the database. This is a highly sensitive security issue concerning privacy, and failure to protect a visitor’s data is illegal, hence there will need to be heavy validation on the input forms for the databases, however, the way I could restrict this is by limiting forms of input, such as steering away from using open input forms, and using a drop down box of preset inputs, thus eliminating the ability to maluse the input system to access the database. Other necessary protection will need to be instantiated to protect the database, such as a level of encryption or password protection upon entering the database. Perhaps using a log in once on the program will grant access to the database as long as the program remains open.

Ability to embed explanation

- Keypad scanner

Obtaining the equipment will likely be the difficult part of this problem, as it will require a simple scanner alongside multiple chipped key cards for use of reading. Researching the equipment further, I have discovered a reasonably cheap RFID kit for the raspberry pi. Thus to likely solve this problem it will require me to study the electronics of the raspberry pi, and ensure that a valid input can be read from the keycards. Upon scanning the keycard, an integer input will need to be retrieved (which will likely represent the ID of the keycard in the database). A comparison will then need to be performed as to whether or not the lock should deactivate in response to that key card being scanned, and this will be done through comparing if the key card's associated room is currently booked, where it will then presume that the user scanning is the appropriate visitor - and will open, make a successful beep sound, and display a welcome message on the LED screen.. Otherwise, if the key card room is not currently booked, or if the keycard is not the correct card for the door, it will remain locked. The outputs to show this failure to open is to make a 'negative' beeping sound, and an LED screen giving an error message, denying the card. This comparison will likely be made using an "IF" and "AND" statement, as in (in pseudo code):

```
If (keycardNo == DoorNo AND = Indate(DoorNo)==True) {Open,  
Display "Welcome"}  
Else {Display "Error"}
```

The above pseudocode infers a method called "Indate" which would produce a boolean output from the input of DoorNo, producing true for if the room is booked, or a false if the room is not booked.

- 'Beep' upon scanning

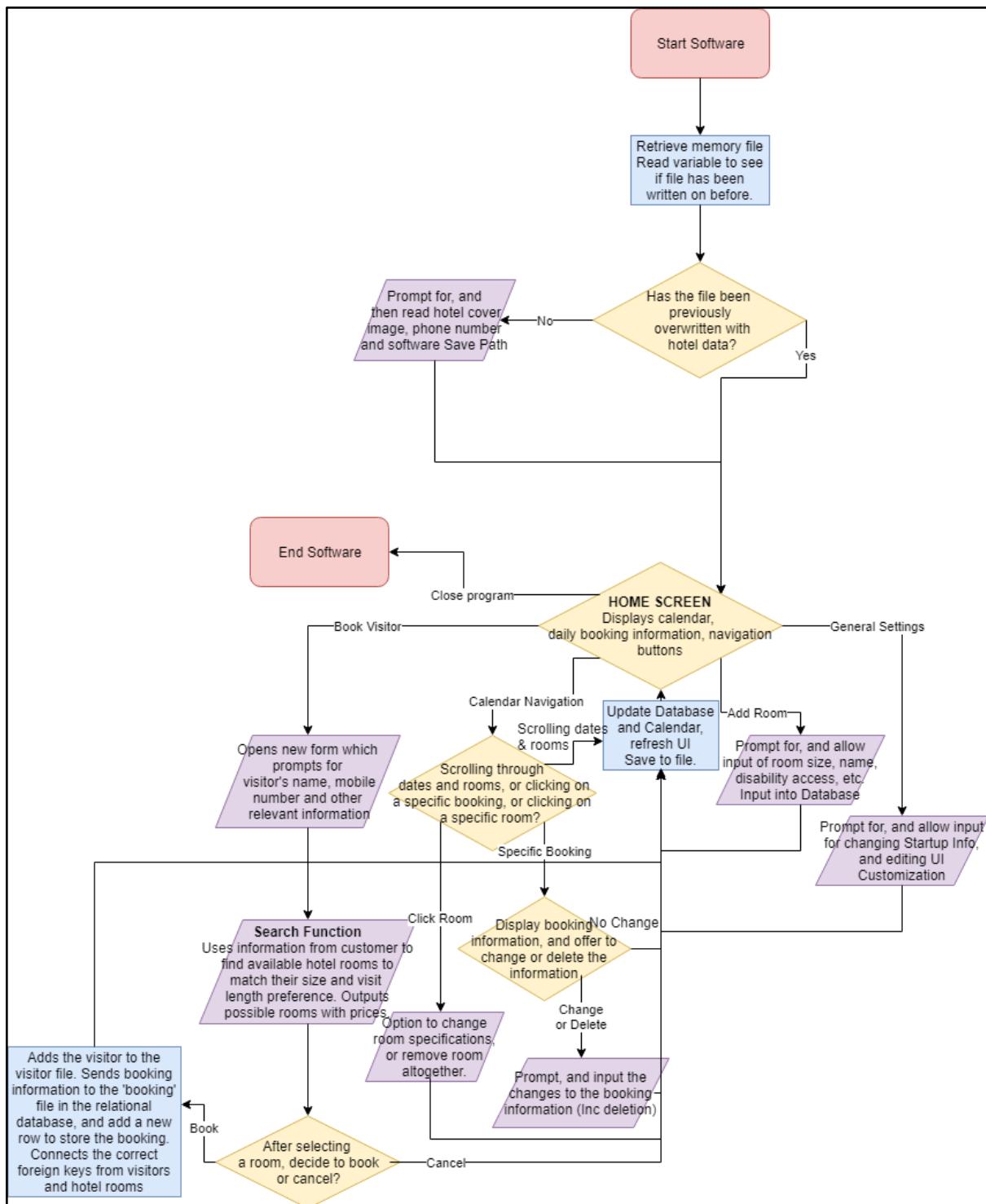
This will require a simple buzzer in the raspberry pi that will output two variations of sound depending on the output of the lock system. Shown above, it will link directly to the pseudo code alongside either the "Display 'Error'" or "Display 'Welcome'", and will play a corresponding function to the buzzer. For example, the function (Open) could initiate the sound for a successful 'beep', and the specific commands to use in

the function to make this sound would have to be created, perhaps the buzzer used will allow for different pitch emission via different instructions, thus allowing a difference to the successful ‘beep’, and the failure ‘beep’.

- Lock

The lock will likely be magnetic, and will deactivate this connection upon receiving the command to deactivate. It will be important that this deactivation be temporary every time it is called upon, otherwise the system may produce a permanent unlock if something goes wrong within the code, and I could ensure that this deactivation remains temporary (as to ensure room safety) in two ways. Firstly, I could use a timer once the deactivation is called upon, by containing the deactivation in a function where the function ends with a locking of the magnet. This ensures that the system cannot continue until the lock is activated, and the room is secured. The way I could add security onto this, is to place the command to ‘lock’ the door at frequently iterated areas through the code, just to ensure that the lock is locked.

Initial System Flowchart



Start Software

Starting the software stands for when the software is ran as a process from (likely) a .exe file. This means that the program's relevant files will all need to be installed in one packet to a computer. The files will be stored in a .zip file to allow easier organisation of files: as zip files allow for extractions to specific files to maintain order and clarity when placing

the program files, thus making it easier on a user which may happen to have less “computer navigation” experience. Zip files will also be used for their ability to compress files, which will make it easier to distribute the software (if need be) via the internet.

Retrieve memory file...

This will be the first function the program will perform, where it will read a file (most likely a basic .txt file) which was installed within the program files once extracted from the .zip, described above. This file will contain either a FALSE variable - if the user has never open the program before, or it will contain the word TRUE, alongside the startup information about the hotel (which is anomalous information which cannot be stored in the database) such as the save location of the hotel’s cover image, and the hotel’s name and number. This information will need to be loaded from the file for use on the home screen. Databases will not need to be checked at this stage, and will be expected to be loaded if the file reads TRUE - as TRUE would entail that the user has previously inputted hotel room information, hence the database for the calendar is not expected to be blank. Otherwise, if the file is empty and reads FALSE, the databases will not need to be loaded as they will be expected to be either blank, or nonexistent. Whatever information is read from the file, it will be stored in the code as a variable, and used in the decision below. The extra information if it is TRUE will be stored in string variables, for use when the home screen is loaded.

File previously overwritten?

Using the information taken from the memory file, there will be a decision made on whether or not the software is being ran for the first time. In pseudo code it would follow:

```
If (line1 == "TRUE")
{name = line2, contactno = line3, coverpath = line 3}
Else
{launch Startup}
(Home Page)
```

This shows that through either selection, both will end up at the (Home Page), which is important as that will serve as the main hub of the software. The system will then decide to either store the information in the memory file, if the first line is “TRUE” (which means the program will have previously stored information, able to be loaded). If the file does not contain TRUE on the first line, then the program will presume the program has not been ran previously, and will prompt the “Startup” function, which is shown below. Heavy validation will need to be made on the reading and writing of the word “TRUE” on the first line, as there will need to be no way for the program to accidentally stored the word alongside a “space” for example, where the system would then presume the program has not been ran before, despite it saving “TRUE”. There will need to be no way for this reading and writing system to go at fault by itself, other than an outside presence interrupting the memory file directly.

Hotel cover image, number, and save path

This is the first visual form a user will see, upon running the program for the first time. It will prompt the user for information about the hotel for the use of personalizing the home screen. There will be spaces for strings of information for both the hotel name, and phone number(as the number is being displayed as a string type), and these will need to be validated to ensure there are no conversion errors, or erroneous data which would crash the software. Save path will be important for users who may have changed their disk, or are using a backup where the path containing the software memory has changed. This input will allow a user to reconnect the software’s files manually, if need be. The purpose for this being safeguarding, to ensure that a use of a different drive does not determine the loss of the hotel’s data. Visual Studio should provide a file browser to allow the user to input the path with ease. Upon first launch of the software, this path input should already be filled in with the default save location of the databases and memory files, and the user may need to be warned not to change the input, if they are unsure of its function - to ensure no accidental data loss by selecting the wrong path. Also, the home screen will be able to function if there are no inputs made for the hotel’s phone number, name, and cover picture - meaning that these categories will be optional. This is to increase the efficiency of the

program, and for a quick simple start up if need be (as the start up data can be changed within the general settings.).

Home Screen

Loading the home screen will be the largest process for the software, as it will need to read from three different databases, and then display the information in the form of the calendar. It will also need to load all of the bookings and their attached information, to ensure that they are graphed correctly onto the calendar, and are colour coded. It will then need to process which visitors are expected to book in and book out, by taking the current date via the computer system. It then takes these expected bookers, and stores them in a table (Shown in UI design below). It will need to load the calendar in relation to the current day, to minimize confusion for the user, and eliminate the trouble of needing to find the correct date. As well, upon loading, the screen will need to read the user-set variable of colour template, and then change the UI accordingly (this colour variation is shown below in the UI presentation section.). Also, the cover photo and previous hotel information described to be retrieved upon startup, will be placed accordingly on the home screen, and will be set to a certain placement to ensure that they do not appear awkwardly on the form. This will be done by setting a fixed resolution for the cover image, to ensure that the image does not break a certain boundary. This will also be the first form that the user will be able to interact with multiple buttons for different outcomes, these outcomes either having a direct effect on the home screen- thus updating it, or opening a new form. It will serve as the main ‘hub’ of the software, giving the user the liberty of access all other parts of the software from it, whilst also displaying the vital information through the calendar and booking table.

Book Visitor

Visitor Input

It will prompt for the visitors information into certain input boxes. These input boxes will be the first to require validation, as they will be connecting to the database - which requires better privacy unlike the initial setup variables which only be stored in notepad (as they are not information based on privacy). Appropriate SQL validations will need to

be enforced. Once inputted, the information will not be sent to the database until the booking is complete (shown below alongside the search function) - to ensure that every value of booking has a visitor attached.

Search Function

The search function process will be integrated with the booking form, and will be a small input output system, with the end goal of having a certain free room selected. The initial input will be inputted via the booking form, and will consist of which type of room the visitor requires, and also the dates they will be staying for. Following the inputs, and submitting the information via a 'search' button, the search function processing will then begin. It will first search the database for every room with the correct size matching the visitor's preference, and then from this list it will search for the rooms which match the availability of the visitor's visit times. The rooms successfully matching the criteria of this "AND" statement will be displayed in a table, displaying all of the rooms additional information - such as floor or ease of disability access. This is justified as to when a receptionist is trying to sell the room, explaining additional details to the customer may be necessary to helping decide a specific room. With all the rooms outputted into the graph, the user will then be able to select a room - or change the search preference if no rooms are available matching the criteria. Once a room is selected, it will then allow the user to click the 'book' button, which will prompt the function below.

Book or Cancel

This will be a small form clarifying the booking information - to reduce mistakes by giving the user an opportunity to read a condensed version of the booking. Once the user has clarified the booking, they can then decide if they would like to book the information, thus storing it within the database, or to cancel the booking and change a piece of information - which will cancel the booking temporarily. Cancelling the booking will return to the booking screen prior, with the same variables that were previously entered for easy corrections if a mistake was made. This means that the variables will be stored in temporary storage when the user decides to book, so that they can either be loaded back into the form for updating, or to be submitted and copied into the database.

Storing Booking Information

This will occur if the user decides to book the booking, meaning that the information will now need to be copied from the temporary variables, and stored within the database. As it is a new booking, it is likely that a new ‘entity’ row will need to be created for this new specific booking to be stored in, which will need to occur within the database to store the booking information correctly. Once the information is stored correctly, the program will then update the home screen, before loading the home screen for the user’s access.

Update Database, Calendar, UI, and save

This process will be extremely important to the flow of the program, with this process being appropriately called on at the end of every selection and iteration of visiting the home screen. This process ensures that the home screen is kept up to date with the variables stored, such as new database entries or even a new UI configuration. This means that the home screen will reload itself; instead of just displaying an outdated version, every time upon being returned to. It will read all of the permanent variables ranging from database bookings (to update the calendar) to any UI changes, to make sure that the user’s changes actually occur as soon as possible. It will be important to call upon this function upon every return to the home screen, as it is likely that every return will include a change to either the calendar or the UI, meaning it will need to be updated. Also, the database and other relevant variables will be saved upon every visit to the home screen, to minimize data loss by automatically saving the data at these frequent periods.

Calendar Navigation

Navigating, selecting a room or selecting a booking

On the home screen, there will be three main ways to interact with the calendar. Either to change the information being displayed on the calendar (a different set of room bookings, or showing the bookings from a different month for example), and this navigation will be allowed through clicking navigation buttons, likely arrows to have an effect. Upon clicking one of these navigation buttons, the home screen will simply refresh to show this different intended area of the calendar.

If the user decides to click a room, it will initiate the specific room’s form (described below)

If the user decides to select a specific booking on the calendar, it will initiate the opening of the specific booking for (shown below).

Option to change room specifications

This option will open upon the user selecting a room on the calendar. It will be a small form, displaying all of the selected room's information retrieved from the database. It will allow the user to change this information about the room, as well as delete the room from the calendar entirely. Coherent warning will be displayed, and perhaps deletion will be denied if the room has pending bookings still to complete - simply to validate the system and to avoid confusion for the user. Upon deleting or changing information, it will reload the home screen, and update the calendar with the appropriate changes.

Change and delete booking, or make no change at all

By clicking on a specific booking on the calendar, this will enable the user to edit attributed information to the booking, via a new form opening. If the user decides not to make any changes to the form, they will close the form after seeing the information, which will prompt the usual home screen update and reloading. If the user decides to edit the booking, or even delete the booking, the below input will be available. The information in the input function will be loaded from the database, and if the information is null, a default will be loaded into the form.

Input booking changes (including deletion)

Once the form is open, with the available input functions alongside the relevant text (shown below in GUI form), the user will be able to input and edit the loaded information. Once the information has been changed, the user can then click a "submit" button which will validate the newly entered information, before either updating the form to show error with the information, or submitting the information to the database if correctly validated. There will also be a button input to "delete" the booking, which will display a quick warning upon being clicked - instead of deleting the information instantly upon clicking. This will be a reasonable security option, to protect the accidental loss of data- and reduces an arguable downside to computation over paper based system - the fact that information can be deleted much easier on a computer. By putting this deletion safeguard in, it will definitely reduce the risk of losing data, by sacrificing a slight section of time efficiency. Once either of these changes are made, the home screen will be refreshed and loaded.

Add Room

Allow input for room specifications

Clicking the “Add Room” button on the home screen will load a form similar to the room booking form, though it will not load any existing data. It will allow the input of room specifications, such as room size, and room number. There will be a submit button, which will validate every input upon being selected, this validation meaning that every input has some sort of value, for submission to the database. Once information is submitted, the home screen will be loaded and updated. Otherwise, if the information is unsuccessfully validated, there will be text alongside the unsuccessful validation, prompting the user with the problem with the input (a non numerical input for room number, for example.).

General Settings

Allow for editing of UI customization, and startup settings

Loading settings from the home screen will load a form displaying the more cosmetic customisations of the program. It will allow the user to change the initial settings that the program first prompts (such as hotel name, and an input for an image file to be used as the ‘logo’). It will also contain inputs for customization options not offered upon initial setup, such as the colour scheme customization (shown below in GUI form). The input for this will be a rgb or hex colour selector, and this changed from using templates as rgb and hex colour selectors come with Visual Studio, which will be quite easy to implement rather than colour schemes, and also allow more user customization by having a broader array of colours. Once the user has finished editing the customization, there will be a submit button - validation - and then submission to the form or to the database (depending on the information inputted). Once reloading the home screen, these changes should take effect.

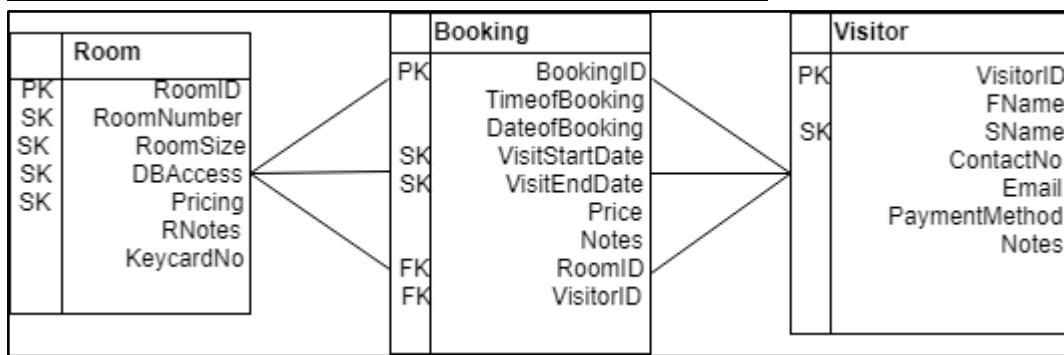
Close Program

End Software

This ending of the software will be designated upon closing the program from the home screen using the out of program GUI red close button. Upon clicking this button, the program should be able to prompt a save before closing, ensuring no data is lost - however, if a change is made in a menu, it should always be saved before entering the home screen, hence it should be simple to maintain data without any loss. Closing forms other than the home screen will not close the program, but will just load the home screen, as the home screen will be maintained in memory always as the program is opened, to serve as the ‘hub’ of the program,

to make sure there is no confusing or unexpected closings of the program to say, an inexperienced computer user. The goal of ending the software will be to ensure that it is definitely what the user desires, as well as ensuring no data is lost upon exiting the program. Also, to maintain variables correctly to ensure that the program is ready to begin successfully on the next launch.

Database Relationship Diagram



This initial database design shows how the database will be structured, and how the relationships will be structured. Firstly in terms of security, the visitor database will need to be the most validated and secured database, due to the legality of maintaining the security and purpose of private information, hence the access to this database will need to be minimal to ensure no data is publicly accessible.

The booking file maintains a sensible “one to many” relationship between the visitor and room files, as there will be only one visitor (or payer of the visit) to one booking, and as well there will be only one room attributed to one specific booking. If a visitor wishes to book multiple rooms, then this will be handled as multiple bookings. Hence, the booking file will take both VisitorID and RoomID foreign keys, so that the booking file may interact with both of them to take relevant information. The booking file will be the file used in the calendar to identify which room is booked at what time, and by which visitor. By serving as this link between a room which has an ‘infinite’ availability, the booking table limits the room’s availability finitely, thus allowing the information to be displayed to the user. It uses the booking end and start times as secondary keys, as these will be important details for SQL queries when indexing the bookings for the calendar. These are the two details which will be most commonly used for searching, whilst all the other

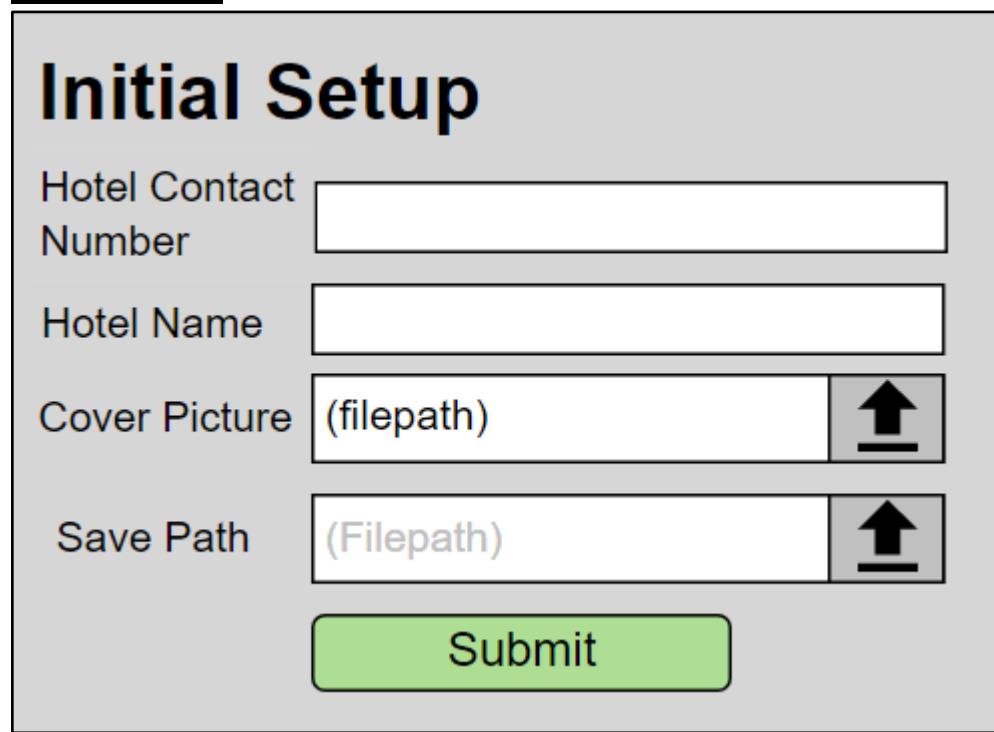
information will not be used for searches, but will still need to be retrieved to be displayed in the specific booking form. The primary identification is simply BookingID which will serve as the unique identification for the booking, however I imagine that this will not be used as it does not need to be passed into either the Visitor or Room table, due to its finite nature. The “price” field will be important to working out the price of the booking, and will need to be passed to the booking form upon the room being selected. The price will need to be in the double format, and will need to be validated or rounded to two decimal places, due to it being a price. The notes field will contain additional information about the booking, such as if a user requires disability access to the hotel - and this information will be loaded on the main menu in the daily booking table (shown below). The pricing field will be a multiplication of (Room price* length of stay in days) - with the ‘Room price’ being an attribute sent via the foreign key in RoomID.

The room file will store data added from the “add room” function, and will be used to associate a visitor in the booking table. The rooms will not have a foreign key, as it will be a table dedicated to taking and storing inputs of the hotel rooms. The pricing field will be important for passing into booking, as it will be used to be added or multiplied depending on the visit length, but the room table will not need to calculate anything like the booking table will, it only stores the price of the room (in validated two decimal place double). Room size will be an example of a field which takes only a restricted set of data; thus protecting the database from being intercepted via inputting SQL commands, an example of this would be that there will be a drop down box in the ‘add room’ form which allows the selection of a selection of preset sizes. I will need to apply a broad range of room sizes to accommodate for every available hotel room size, and this can be as large an array as possible, as the user will be inputting the room’s individual price manually. I am allowing manual input of room prices, as it may become an annoyance to the user if room prices are distinguished by myself then it will restrict the hotel’s specific room pricing requirements, hence it should be open for manual inputting. The visitor table will require the most validation for security, as it will take the input of the major personal details of the visitor such as phone number, full name, and email addresses. These will be stored together,

and will not have much relevance in being passed to the booking form, other than just the full name. The second name field will be a secondary key, as it will be best used as an index to search for a specific visitor.

GUI Plan

Initial Setup



The image shows a user interface for 'Initial Setup'. At the top, the title 'Initial Setup' is displayed in a large, bold, dark blue font. Below the title are four input fields arranged vertically. Each field consists of a label on the left, a text input box in the center, and a small icon on the right. The first field is labeled 'Hotel Contact Number'. The second field is labeled 'Hotel Name'. The third field is labeled 'Cover Picture' and contains the placeholder '(filepath)'. To the right of this field is a small icon of an upward-pointing arrow inside a square. The fourth field is labeled 'Save Path' and also contains the placeholder '(filepath)'. To the right of this field is another small icon of an upward-pointing arrow inside a square. At the bottom of the form is a large green button with the word 'Submit' written in white.

Shown is the different input fields for inputting information. It is a very simple, and specific design to be easily understood for a new user. The arrow by the cover picture will open a file dialog, to allow the user to select an image file to be uploaded, and will convey an indication in the form once the file has been uploaded (either successfully or unsuccessfully). The fields are split to equal pixel lengths simply for visual ergonomics. I have used the fonts as a placeholder, as I do not currently know the extent of availability of fonts Visual Studio, however, I feel as though the placeholder is a suitable, and simple font which matches the feel of the program, thus I do not deem it to change too much in terms of font during development.

Home Screen

HOTEL VICTORIA

NEWQUAY

01270 767339

▼ Visitor Name	▼ Booking in/out?	▼ Status	▼ Note
Kieran Turo	In	<input checked="" type="radio"/>	
Peter Johns	Out	<input type="radio"/>	
John Lemon	Out	<input checked="" type="radio"/>	
Wilson Hundura	In	<input type="radio"/>	
		<input type="radio"/>	

Save
Settings

Book		Add Rooms			
		January, 2017			
		28, Sat	29, Sun	30, Mon	31, Tue
		Turo, Kieran			
101					
102					
103					
104			Janiels, Jane		
105					
106					
107			Hundura, Wilson		
108					
109		Lemon, John			
110					
111		Johns, Peter			
112					

This will be the home screen of the program, and will open as a separate form to all the other forms, instead of just hiding the contents on an empty form. Shown in the top left is the use of the background image, which will be a fixed box which will store the image, and scale the image to the correct size of the box - to keep order in the organization of the form.

Below the background image, is the list of visitors expected to have a reception interaction on the current day, and it displays all the relevant information to track the entry and exit of visitors. This table will need to be saved and reloaded upon entry, as well as update depending on the day. It is designed for quick input from the user, to keep a track of visitors and the completion of their visit, or to highlight any anomalies which the hotel may need to follow up on by contacting the visitor.

Next is the calendar on the right, which is the main part of the home screen. It displays the axis of time / rooms. Specific slots will be coloured depending on the booking status, and the specific bookings can stretch across the time axis, but not the room axis. Red represents a booking which has not been booked in, green represents a successful booking in - yet is awaiting to be booked out. An orange booking shows a booking which has been booked in and booked out, thus showing a completed booking. As shown, this colour scheme will make it easier to track recent

bookings for the user. Both axis are chronological, either rooms in number order, or time in order. Also, the current day is coloured green for easy manual tracking. Currently, the idea of navigating the calendar is shown using sliders - however, this may be difficult when put into practise and is a volatile representation.

Around the home screen are the different buttons for outer-home navigation. They also contain pictures of their relevant functions, making it easier to learn and identify the use of different functions. An example of this is the 'save' function, which contains the save icon which is widely recognizable as 'save'. Also, the use of making the buttons white separates them from the rest of the program, thus making them visually available as buttons.

The Booking Screen for Hotel Victoria Newquay displays room bookings for January and February 2017. The interface includes a header with the hotel name, phone number, and a booking table. The main area shows a grid of room numbers (101 to 112) against dates, with specific bookings highlighted in color.

January, 2017				February, 2017				
	28, Sat	29, Sun	30, Mon	31, Tue	1, Wed	2, Thurs	3, Fri	4, Sat
101	Turo, Kieran							
102								
103								
104			Janiels, Jane					
105								
106								
107		Hundura, Wilson						
108								
109	Lemon, John							
110								
111	Johns, Peter							
112								

Left Panel:

- Visitor Name: Kieran Turo (In)
- Booking in/out?: In
- Status:
- Note:

Bottom Buttons:

- Save
- Settings

This is an example of how the colour scheme will work, as when the user updates their preference for background colour, it will change how the background of the form is coloured. Here it is shown that the new colour is not too demanding in form change, and is only a small visual difference which is not visually demeaning to the rest of the program's function.

Booking Screen

New Booking

First Name

Second Name

Mobile

Email

Notes

4/20/2017 To 4/22/2017

Room Size Options

Search

▼ Room	▼ Notes	▼ Price	▼
101	Balcony	£120	<input type="radio"/>
107	Bottom floor	£95	<input type="radio"/>
115	Bottom floor - Television Access	£100	<input type="radio"/>
125	Television Access	£90	<input checked="" type="radio"/>

Book

This is the booking form, which will open as a separate window upon clicking the booking button in the home screen. It is split into two sections, one for visitor input and one for booking input. The visitor input fields are spaced out evenly and are prepared to take an input. The booking half then requires two dates from the visitor, a start and end

date, and also a few search enquiries about the room (mainly size). The search function can then be performed inside the form, and shows the suitable and available rooms in a table form, along with their total price (price of room * length of stay in days). From here, the user can choose a room suitable for the visitor, and then book using the book button. The form is specifically intuitive, to be followed from the top down to complete the booking.

Add Room Form (With input validation)

The screenshot shows a 'New Room' form with the following fields:

- Room Number:** Input field contains "Nine". To its right, a red error message says "Please enter a valid room number."
- Price Per Night:** Input field contains "£".
- Room Size:** A dropdown menu is open, showing "Options" as the selected item.
- Notes:** A large text area containing "String Value".

At the bottom is a green "Submit" button.

This is the 'New Room' UI, and allows the user to add a room to the calendar. As shown, the user inputs room number, price, and room size from a selection of preset sizes. The notes section is also noticeably larger, as it is likely to contain a larger quantity of information, thus allowing the user to see the information they have inputted more clearly. This is also an example of how validation will work, which will be reciprocated in every other form which requires inputs. If the input has an unexpected data format (in the example, if room number contains a string value), it will not submit the information upon the submit button being clicked, but instead display a message alongside the relevant input, displaying what was validated in the input box. Whilst validation will be minimized by using drop down boxes, validation will still need to be incorporated when open inputs are required. The goal of validation is to not allow the user to update the database until all the fields are correctly coherent.

Specific Room Form

Room 101

Price Per Night	<input type="text"/> £
Room Size	Options <input type="button" value="▼"/>
Notes	String Value
<input type="button" value="Delete Room"/> <input type="button" value="Save Changes"/>	

This smaller form will contain information about pre existing rooms, and will allow the user to edit this information if need be - as well as delete the room entirely via the red and distinct “delete” button. Once again the notes section is larger to accommodate an inevitable larger quantity of information.

Specific Booking Form

Fred's Booking

Stay Length	<input type="text"/> 4/22/2012 <input type="button" value="▼"/> To <input type="text"/> 4/22/2012 <input type="button" value="▼"/>
Room	Room 101 <input type="button" value="▼"/>
Price	<input type="text"/> £
Notes	Fred will require access to the disability entrance to the Hotel.
<input type="button" value="Delete Booking"/> <input type="button" value="Save Changes"/>	

This screen allows for edits or viewing of pre-existing bookings, and is similar to the “new booking” form, but without the visitor section as the visitor input is already assumed. It is personalized in the title for easier understanding, by putting the visitor’s name in the title (see “Fred’s Booking”). It allows a manual inputting of price, which will be validated for a two decimal place double data value. Once again, it allows the deletion of the booking which will delete the information out of the database.

Settings Screen

General Settings

Hotel Contact Number	<input type="text"/>	Background Colour <input type="color" value="#c2c2ff"/>
Hotel Name	<input type="text"/>	Calendar Orientation <input style="width: 150px; height: 25px; border: none; background-color: #f0f0f0; border-bottom: 1px solid black; padding: 0 5px;" type="button" value="Date V Rooms"/>
Cover Picture	(filepath) <input type="text"/>	<input style="width: 25px; height: 25px; border: none; background-color: #f0f0f0; border-bottom: 1px solid black; border-left: none; padding: 0 5px;" type="button" value="Upload"/>
Save Path	(Filepath) <input type="text"/>	<input style="width: 25px; height: 25px; border: none; background-color: #f0f0f0; border-bottom: 1px solid black; border-left: none; padding: 0 5px;" type="button" value="Upload"/>
<input style="background-color: #ffcccc; border: 1px solid black; border-radius: 5px; padding: 5px 15px; margin-right: 10px;" type="button" value="Discard Changes"/> <input style="background-color: #90EE90; border: 1px solid black; border-radius: 5px; padding: 5px 15px;" type="button" value="Save Changes"/>		

The settings UI will be clearly different from the rest of the forms, by being rectangle. This shows a bit of separation from the rest of the program, clearly indicating to the user that these settings are less active in the role of managing rooms. All the fields are aligned, and allow dialogs in the correct places (cover picture, and background colour) for inputting of data. Also, it uses a drop down box to change the calendar's orientation. Once again, there is a green 'save changes' button to indicate that this will be the positive exit out of the form to save the information, as well as a negative 'red' button to leave the form without making changes, on the chance of the user wanting to leave the form without making changes.

Variable Table

Variable Name	Data Type	Purpose
SavePath	string	Stores the result of the save path dialog, when the user chooses a path to save the config file.
path	string	Stores the actual file name returned from when the user selects a picture file. Will be joined with SavePath to retrieve the image.
ContactNo	string	Used for validating the contact number. The value of the label will be passed as variable 'Contact'. It will then be used to check the phone numbers length, and quality.
HotelName	string	Used to store the value entered in the hotel's name spot. Will be used for

		validations.
Val	bool	Used for validations, to see at the end of the validation function whether or not the program is ready to continue. Will start as true, will change to false if a validation test is failed.
Writer	var	Used as a random variable to be associated with the .txt writer streamwriter function - to write to txt files. An instance of streamwriter is stored in Writer.
HomeScreen	Form2	Special to Visual Studio, I will call a data type 'Form2' which is an instance of a form GUI.
label	Label	An instance of a label type, which represents a single label in the form. The label can then be interacted with, once set to this variable.
Rich	RichText Box	Another instance of a richtextbox in the form, which can be set and interacted with - as well as its text value or colour.
DAnchor	DateTime	This would be the main date which the entire calendar will revolve around. By taking this date, the rest of dates to be displayed in the calendar can be found by adding an extra seven days to the anchor date.
BookingForm	BookingForm	An instance of the booking form, which can then be created once the user wants to view and interact with the booking form.
cmd	SqlConnection	This is an instance of an sql connection with a database, where the program can use specific methods linked to the SqlConnection class which will allow it to interact with the database in a number of ways - mainly to run SQL commands in the database.

Comm	SqlCommand	Branching from the above connection, the ‘SqlCommand’ data type will store the specific string SQL command to be ran in the cmd connection, as well as the database to become connected to. It takes these two things as separate parameters.
ConStr	string	This will contain the connection string to the database - relative to the computer accessing the database. It will be found using the ConnectionString() method - which will discover the string where the database is stored, and hence saving it to ConStr. It will then be used to form a physical connection with the database.
AlphNum	Regex	This will be used in validation, to compare whether or not a string contains alphanumeric characters only - to validate the first name field for example. It is regex, as regex represents a broad number of single characters. In this case, it will represent only the 26 alphabetical letters, and numbers 0-9.
end	bool	This will be used in the second set of validations in the booking form, to check if the booking information is secure and proper enough to be sent to the database. It will be set to false if it fails a validation.
Num	Regex	Another regex for validation, but this regex only contains number 0-9. This will be used for validation of the phone number inputted for storage - to ensure it comprises of only number characters.
reader	SqlDataReader	This is a variable to be used during a database connection (cmd). It will read each individual line in the database, and each record. It will then output every record’s data in the database, if set on a loop. This will allow me to retrieve everything from the database if needed.

OutputList	var	It will represent a list of rooms outputted from the database. It should be a list of data type 'Rooms' (a class made by me). It will then store all the individual pieces of data in an organized fashion, working alongside the reader to retrieve everything from the database.
tabnum	label	This will store a select label during a single iteration. It will store which label is being interacted with, when a loop is done to cycle through every label in the table - or a single column.
combo	string[]	Will hold the array string data to display in the combobox in the booking process, to select the type of designated room required. All the room types will be stored in this combo array, to then be stored in the combo box all in one cycle.
SearchList	string[]	This will represent the returned rooms from a booking search. It will be used to find the room selected by the user using the checkboxes, to see exactly which room is wanting to be booked. By having them in an array order - similar to how they are displayed - the room can then be associated with a checked checkbox location.
check	CheckBo x	To identify which checkbox is selected, it is required to loop through using a for loop through every possible checked checkbox. By putting a loop's instance as 'check', by selecting the physical checkbox being checked and storing it in this temporary variable, the decision as to whether or not it is checked can then be made.
VID	int	This will be a public variable, to store the retrieved visitor ID from the SQL command. As the visitor ID cannot be found from just the program - as it is listed

		in the SQL table, it must be retrieved from a very reclusive position in the code, meaning using a public variable would be simpler to extract the integer once retrieved.
VStart	Datetime	This will store the retrieved start date from the booking table in the sql command indentation. It will be then used to add an attribute to a new 'booking' object.
VEnd	Datetime	This will store the retrieved end date from the booking table in the sql command indentation. It will be then used to add an attribute to a new 'booking' object.
Booked	Bool	This will store the retrieved status of the room's booking from the room table in the sql command indentation. It will be then used to add an attribute to a new room object. It can then be used to check if a room is booked or not when another booking is trying to take that room, to ensure no room is booked during the same period of time.
Db	Bool	This will store the retrieved value of whether or not the room is suitable for disability access. It is boolean, as there are only two states this variable can be measured as (suitable, or unsuitable, 1 or 0). It will then be stored as an attribute in a room object.
Num	string	This will store the string value of the room number of the rooms. It will retrieve it from the database, and store it in this variable before being attributed to a room object. It is a string, and not an integer, as it will only be used in a sense of displaying, and will have no arithmetic use as it is just the room number - not representing a real number, just the organized value of the room. Hence, only displaying the room

		number in a label will mean it would be better suited as just a string consisting of numbers.
Size	string	This variable will be limited to those values stored in 'combo[]'. As it will retrieve the size of the room from the database as a suitable string datatype. It will then store the value of the size in the object attribute, when the class is instantiated.
Price	Double	This is used to store the variable of the price of the room or booking, and will store it either before sending it the data to the database, or retrieving it from the database to place in an object. It is double, as mathematics will need to be performed on the number to calculate any total prices, and a double allows integer data to two float points.
Notes	string	Once again, using this variable to store the notes of the booking or room to communicate between the program and the database.
Diff	Timespan	This will store the difference in time between the visit start, and visit end. It will then use the .days value in this timespan to calculate how much to multiply the booking's price by, to ensure the amount of nights are successfully taken into account when producing the booking price.
iv	integer	This will store a the constant displacement of the calendar navigation, when travelling in the y direction (moving the rooms up or down). It will be changed by either +1 or -1 depending on which button is pressed.
c	int	This is the for loop counter to count which selected booking is needing to be checked. It begins at 1, to skip the first row in the calendar.

MiniList	List<Booking>	This is a list of bookings which have a start or end date on the current date. It will be used to fill the side calendar.
----------	---------------	---

Function Table

Name/Type	Purpose	Pseudocode
UIUpdate() Activated upon entry to form.	Updates the home screen UI by setting the values of the UI controls to that of what is contained in the config file - saved during the setup, by reading the file with StreamReader.	<code>Using StreamReader{ BackgroundBox.Image = reader.ReadLine(); HotelNumber.Text = reader.ReadLine(); HotelName.Text = reader.ReadLine(); BackColor.Color = reader.ReadLine(); }</code>
CalRight() Activated upon user clicking 'right' button on calendar.	Navigates the calendar by a forward week of dates by changing the 'anchor date' and adding 7 days to it. The calendar is then updated.	<code>DateAnchor = DataAnchor.AddDays(7); CalUpdate();</code>

CalLeft() Activated upon user clicking 'left' button on calendar.	Once again navigates the dates by a week, but in the backwards direction. Does the same as above, but removes 7 days from the anchor.	<code>DateAnchor = DataAnchor.AddDays(-7); CalUpdate();</code>
BookingClick() activated when the user clicks the 'Book Room' button	Will open an instance of the booking form, and hide the home screen from view.	<code>HomeScreen.Hide; Bookingform = New BookingForm; BookingForm.Show;</code>
RoomClick() activated when the user clicks the 'Add Room' button	Will open an instance of the room form, and hide the home screen from view.	<code>HomeScreen.Hide; RoomForm= New RoomForm; RoomForm.Show;</code>
LinkString(), only called upon when the link string is required to access the database.	This will find the connection string to the database locally from any computer, and will return the string.	<code>String path = CurrentDirectory; Return "Data Source=(LocalDB)\\MSSQLLocalDB;Attach DbFilename='"+path+"Database1.mdf;Integrated Security=True"</code>

ClearTable(), called when the calendar updates.	To clear the calendar's controls, to ensure that it is blank upon being written to - so no overwriting occurs (leftover controls from a previous update). It will do this by iterating through every cell in the table and setting it to blank.	<pre>TextBox Rich; for(x;x<XLength;x++){ For (y;y<YLength;y++){ Rich = TextBox from (x,y); Rich.Text = ""; Rich.BackColor = White; } }</pre>
NavUp() Activated when the user clicks the 'up' arrow	To move the entire table up by one slot, to visualize a navigation through the y axis of the calendar.	<pre>Constant = Constant +1; UpdateCal()</pre>
NavDown() Activated when the user clicks the 'down' arrow	To move the entire table down by one slot, to visualize a navigation through the x axis of the calendar.	<pre>Constant = Constant - 1; UpdateCal()</pre>

UpdateCal() Called whenever the calendar needs refreshing.	To update the calendar's attributes and be used as a refresh function.	<pre> ClearTable(); //Refresh Dates for(i;i<XLength;i++){ Dates[i-1] DateAnchor.AddDays(i-1); Textbox in Pos[i,0].Value = Anchor.AddDays(i-1).ToString(); } //Refresh Dates Var Bookings = List of Database Bookings Var Rooms = List of Database Rooms. If (Rooms.Length>0){ Int count1 = 0; Int count2 = 1; Foreach(Room room in Rooms){ If (count1<=YLength+iv&&count1>=count){ Textbox in (0,count2) = Rooms[count1].RoomNumber; count2++; } } } </pre>
CalColour() Activated when updating the calendar.	To colour the calendar's cells appropriately.	<pre> For (int i = 1;i<YLength;i++){ If Rooms[i].StartDate.Days = Today{ Textbox[0,i] = color.Green} If Rooms[i].StartDate.Days > Today{ Textbox[0,i] = color.Amber} If Rooms[i].StartDate.Days < Today{ Textbox[0,i] = color.Red} } </pre>
HomeCall() Called only if the text file already exists for the booking system	To bring the user straight to the home screen on startup if they've ran the program before, by detecting if they have a startup settings file.	<pre> If (This.Directory.config.txt.Exists){ This.Hide(); Homescreen HomeScreen = new HomeScreen(); Homescreen.Show(); This.Close; } </pre>
UponLoad()	To change	Using (StreamReader() @

) This is called upon the any form being opened	the colour and visual settings of the form, by retrieving the user's choice settings from the config.txt file.	<pre>Directory/config.Txt) { ConvertFromARGB(ReadLine(4)) = thisforms.BackColor;</pre>
ColourPick (), called when the user wants to select the colour dialog.	Handles the user's effort to select a colour using the colour dialog. It will then handle a return to store this chosen colour, by converting it into a string and storing it in config.txt	<pre>DialogResult result = ColorDialog.ShowDialog(); if (result == DialogResult.Exists) { DisplayBox.Color = Result; Store ConverttoARGB(Result) @ config.txt Using streamwriter;</pre>
Val() Will be called when a validation is required.	Will take the inputs requiring to be validated and validate them, and returning a boolean value as to whether they are validated or not.	<pre>bool val = true; // ContactNumber if (NumbVal(ContactNo) == true && ContactNo.Length < 17) { NumbWarn.Visible = false; } Else { NumbWarn.Visible = true; val = false; } // HotelName if (HotelName.Length < 31)</pre>

		<pre> { label6.Visible = false; } Else { Val.Visible = true; val = false; } return val; </pre>
AddRoom() will be called when the program has enough information to make a new record in the database for a room.	Will store a new record in the database for a new room.	<pre> String Con = LinkString(); SqlConnection to (Con){ “INSERT INTO Rooms”{ RoomNum, RoomSize, DAccess, RoomPrice, RoomNotes,} Execute SQLQuery </pre>
SubmitClick() will be called when the submit button is pressed in the booking form.	Will set about validating the inputs, before making the decision as to whether or not to make a booking.	<pre> if(Val('Input Parameters')){ AddRoom('Parameters') } </pre>
RoomGrab() Will be called at the beginning	Will create a list of every room in the database, to easily search	<pre> Var List = new List<Rooms>; Using SqlConnection @ LinkString(){} Open DB(); Foreach(Row in RoomsTable){ List.Add(new room {ROOM DATA IN DATABASE}) } </pre>

of the form creation, to retrieve all the entries in the database.	through them.);
--	---------------	----

Key Stages Planning

I will be breaking my solution down into multiple key stages, to make organizing the workload slightly easier. By breaking it down, it will also allow me to specialize each key stage with a focus - to know when each key stage is complete. I can then grasp an idea of when the first iteration of this solution will truly be counted as 'complete'.

It will also allow me to easily backtrack if required, as each key stage will be its own solution and function in and of itself.

Each key stage will also have planned testing, to then show the green light as to when the solution is complete - When the tests are complete, I know the key stage is complete.

Design - Key Stage 1 - Start Up System

This key stage will focus on the first greeting of the program to the user - upon running the program for the first time. It will be the introduction to developing with forms, and will be a basis to developing a knowledge of working in Visual Studio. It will involve a new form, as well as populating the form with the correct input controls, and visual controls to communicate to the user.

It will also involve the validation of these inputs, as well as returning a visual cue as to when a validation fails - relevant to the validation that failed.

Outside of the form, it will then concern how a config file will be interacted with, to save the input data to this config file to be retrieved later on.

Mainly, this will serve as an introduction to the toolbox of the form.

The testing below is the main test I will perform at the end of the key stage to show that the stage is complete. It will involve testing of every input of the form, to see how the validations respond to the input:

Input	Expected Output	Output	Evaluation
-------	-----------------	--------	------------

Blank data for every field (except default save path) (Borderline data)	A blank config.txt file.		
Valid information for every field, but the .txt file already exists with already valid information inserted (Extreme data)	Config.txt's first three lines are overwritten		
16 character phone number and 30 character hotel name (Borderline Data)	Stores the information normally		
Valid data, but using a long picturebox.Location length (as the background image field is the only field not validated by length)	Config.txt stores the location as normal.		

Design - Key Stage 2 - Database Design & Home Screen Foundation

Key stage 2 focuses on the foundation of the database, ensuring the links are made within the database so that it can communicate internally, using foreign keys for example. This database must be at a level where the form can then communicate with it clearly.

To begin as well on the home screen is part of this key stage, to have a placeholder for the calendar and buttons on the home screen, to then link it to the other forms which will come later in key stages. This will allow me to access these other screens during testing.

The main testing of this intermediate stage will contain the basic navigation of the dates on the calendar (which is a part of this key stage), and the storage of text within certain table cells. If I am able to

access these cells in this fashion at this stage, it will be useful later on when I have actual database data to insert into the table.

Test on navigating the calendar's dates:

Input	Expected Output	Output	Evaluation
Selecting 'right' navigation 3 consecutive times	The table displays the week following exactly three weeks from testing		
Selecting 'left' navigation 3 consecutive times	The table displays the week three weeks prior to the current week		

Test on storing data in specific cells:

Input	Expected Output	Real Output	Evaluation
Normal Value "LB" initials in (1,1)	It stores the initials in a clear manner, in the correct cell (1,1)		
Erroneous data "ABCDEFGHIOL ELE"	The cell tries to bare the input, but does not change size to accommodate		
No data ""	The cell remains blank		
'Label.Colour' set to red.	The specific cell turns red		

Design - Key Stage 3 - Booking System with Database (Including Search Function)

Key Stage 3 contains the first link between form and database. It will begin with an access to the booking form and room addition form from the homescreen, and organizing the controls and tables on the forms. The 'add room' form must be able to add a room filled with validated data attributes and add it to the database created in key stage 2. To then retrieve data, it must then validate it similar to the style used in the first key stage, and returning either a success or failure when the data is submitted - with a success then submitting data to a database. The search function in the booking form must be able to retrieve data from the database specific to terms met from within the form, and store it in the table in a specific order. To be able to select a specific room must then be established, to associate a specific room with a booking to then book that room for the visitor. All the appropriate validations must be made in this form as well.

To test this, it must test the booking form's ability to retrieve and display rooms stored from the room form, shown in this test:

Input	Expected Output	Output	Evaluation
Booking a double room for £224 (VALID)	The room, with all the appropriate details according to the booking, and saved in the table.		
Booking two rooms at once (INVALID)	The booking does not continue, and a validation error is displayed.		
Booking a blank row and pressing 'Book' (INVALID)	Nothing occurs.		

Booking the same room, but set for different times	Both booking are saved, and the same RoomID is stored in the booking table.		
--	---	--	--

Design - Key Stage 4 - Calendar and Calendar Functions

After data is now able to be stored in the database, it can then be retrieved to test and develop the calendar. The calendar will build off of the previous navigation, by now having a navigation in the ‘room’ direction, and store the database’s rooms in the calendar. It will then be able to decide which cells and what dates correspond to the length of a booking to then colour the cells correctly. As well, the ‘current date’ calendar will need to be developed at this stage, to further add the feature of daily compatibility for the user. These tables will need to be developed, and able to be interacted with (delete or change bookings and rooms).

The first test will ensure the bookings respond well to being stored in the mini calendar:

Input	Expected Output	Output	Evaluation
Adding a booking which ends on today’s date	The booking’s name, price and notes are added to the mini list		
Adding a booking which starts on today’s date	The booking’s name, price and notes are added to the mini list		
Checking off the first booking	The entirety of the booking in the calendar turns orange, and the first booking is removed from		

	the mini list.		
Subsequently checking off the second booking	The entirety of the booking in the calendar turns red, and the booking is removed from the mini list.		

The second test corresponds to the items in the main calendar, and how it will respond to be interacted with:

Input	Expected Output	Output	Evaluation
Booking with no start or end date on the current day	The booking is shown in the calendar, but not shown in the mini list		
Booking with a start date on the current date.	The booking's full name is displayed in the mini list as well as "In"		
Booking with an end date on the current date.	The booking's full name is displayed in the mini list as well as "Out"		
Deleting a booking which is present in the mini list	The mini list updates, removing the relevant booking.		

Design - Key Stage 5 - Settings

This key stage will be the 'tidying up' of the project at the end, which will tie together to settings from the start with the rest of the form. It will add colour to every form, and allow the user to reenter the settings from the

home screen. As it is a smaller key stage, it will likely require very bespoke testing which is difficult to foresee. As long as the settings are able to be accessed and successfully changed alongside the other key stages, I know that this key stage will be complete.

Final Test Plan

After completing the last key stage, necessary tests will need to be performed to ensure and prove that the solution is relevant to the original problem. These tests will relate to the planned features of the system, and the results will correlate as to how complete the product is - and which areas may require more bug testing of additional features. The inputs will range across valid, invalid, and borderline inputs to check how the program reacts to each type, and if it handles it correctly.

Also, these tests will be broken down via colour to correlate with the different planned features:

FUNDAMENTAL

- **Initial Setup**
- **Ability to input room and customer information and indicate a price**
- **Ability to input room and customer information from a range of different input devices**
- **A Graphical User Interface**
- **Calendar Display**
- **Saving and Restoring Data**

ADVANCED

- **Keycard Scanning and unlocking upon validation**
- **Safeguarding (Storing Unlock Attempts)**
- **Responsive ‘Beeping’**
- **Colour scheme organization**
- **Search Function**

<u>Input</u>	<u>Expected Output</u>	<u>Output</u>	<u>Analysis</u>
Running the program for the first time and reaching the home screen.	The program notices it is the first time running, and displays the initial setup		

	form.		
Running the program for a second time, following the previous time.	The program displays the home screen instantly.		
Inputting VALID data into the initial setup form. (greg, 43242, -128 respectively)	The information is submitted to the config file in a specific order, and the home screen opens.		
Inputting INVALID data into the initial setup form. (001201026,"greg100 errorerrorproblem" respectively)	The initial setup screen remains open, and red validation messages appear relevant to the invalid inputs. No changes are made to the config file.		
Take 3 peers from my class and input them as visitors to bookings.	The three peers' details are stored as an abstract representation within the database - able to be retrieved and viewed.		
Attempt to manually change price of a booking, overriding the calculated price.	The inputted override price is submitted to the database, ignoring the calculated price.		

Book three rooms for different lengths, and check the booking prices are correct.	Each of the bookings are correctly calculated (Price of room * number of nights stayed)		
Attempt to input rooms prices for 4 different rooms, each to 4 different decimal places (0,1,2,3)	The first 3 rooms are accepted, but the 3 decimal place price is denied via validation.		
Attempt to book and delete a room using only a mouse	The program is navigable, and a room is able to be booked and deleted successfully.		
Attempt to book and delete a room using only a keyboard	The program is navigable, and a room is able to be booked and deleted successfully.		
Attempt to book and delete a room using only a touchscreen device	The program is navigable, and a room is able to be booked and deleted successfully.		
Attempt to book and delete a room using only a puff suck switch	The program is navigable, and a room is able to be booked and deleted successfully.		

Search to see if more than 3 prominent colours are used for the GUI	3 or more colours are successfully used to differentiate different visual elements in the program.		
Try and navigate the program through buttons - Is it a GUI?	The program qualifies as a GUI based program		
Are images used to help communicate button purpose?	Images are identified in the program for the purpose of aiding navigation		
Can bookings be presented in a tabular form?	The bookings are displayed in some sort of table form.		
Can an individual booking be selected, and interacted with?	A booking can be selected, and viewed.		
Add a room to the system. Check the calendar's status.	The calendar adds the rooms automatically to the calendar view.		
Add 3 bookings and close the program. Reopen the program and	The three bookings should be prevalent in being displayed on the home		

access the home screen.	screen - despite the program resetting		
Add 10 rooms and 20 bookings for those rooms, and access the calendar.	The loading speed of the program is not noticeable.		
ADVANCED	TESTING		
Can a keycard scanning be used to communicate with the program?	A keycard being scanned can be recognized by the program		
Does the scanner have different outputs for a successful and failed scanning?	The program recognizes a 'successful' scan and an 'unsuccessful' scan.		
Scanning a valid card at a valid time	The program outputs a 'success' reaction and records the time and location of the scanning.		
Scanning an unrecognized card	The program outputs a 'unsuccessful' reaction and records the time and location of the scanning.		
Scanning a valid card at a valid	A single 'beep' is emitted		

time			
Scanning an invalid card	Two consecutive 'beeps' are made.		
Changing the system's colour	The change is recorded, and every form now has that same colour.		
Setting the colour to something visually obtrusive.	The program counteracts this obtrusive change, to ensure every element is still viewable.		
Searching for a room which does not exist.	The program returns no results		
Searching for a room where multiple rooms meet the conditions	All the possible rooms are shown.		

User Satisfaction Survey Plan

Alongside feature testing, I will need to test that the program is acceptable to my stakeholder demographic. During post development evaluation, I will let my stakeholders interact with the program, and then give a survey asking questions which are contrasting to my success criteria. During evaluation, I can then identify which sections of my program were a success, and which features could be better tuned and improved for my stakeholders.

Each colour in the test plan represents a different section of the success criteria, to help section the results during post development.

Some of the questions I will record whilst they are navigating the program, such as questions concerning the usability and navigation of the program (EG, Navigate to the booking screen, how easy was that from 1-10?)

<u>Question</u>	<u>Result Type</u>	<u>Answer Aim Range</u>
What peripheral device did you use to navigate the program?	Open ended	A range of devices across the group
On a scale of 1-10, how easy was it to navigate the program using your peripheral device?	Scale of 1-10	Between 7-10
Navigate to (Certain section of the program). How simple was that to do, from 1-10?	Scale of 1-10	Between 7-10
Did the use of colour help you during this navigation?	Yes or No	Yes
Find (A Certain room). Is it available today?	A time in seconds.	0-5 seconds
Book yourself into a room. Are you able to do so?	Yes or No	100% efficiency 'yes'

Some of the success criteria does not require end user testing, just reflection. Hence, these tests will be dealt with in the 'features' testing section.

Overall, I will use both of these tests post development to inform reflection on how I could improve the end product, and to see if any certain problems 'fell through the cracks' during development.

Development

I will be splitting development into key stages, which will allow me to concentrate efforts efficiently by keeping smaller problems relevant to

their key stage. As they are relevant to one another, I will be able to evaluate and improve the stages much more efficiently.

The key stages will consist of images of the code, or the relevant part of the project, followed by a captioned response. Text will be relevant to the image it resides underneath, unless stated otherwise. The images will be chronological to how the code developed.

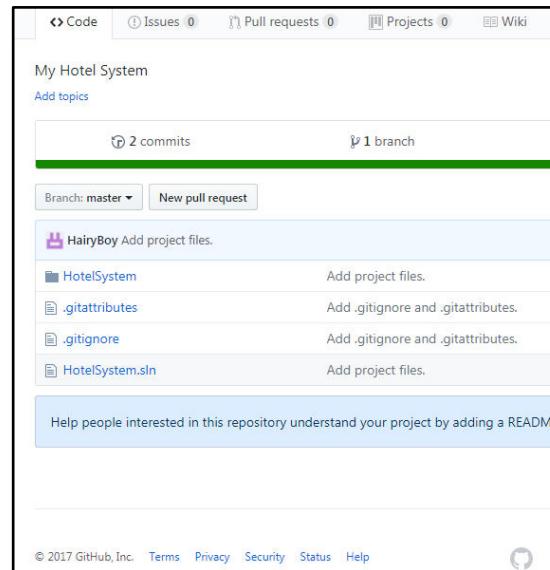
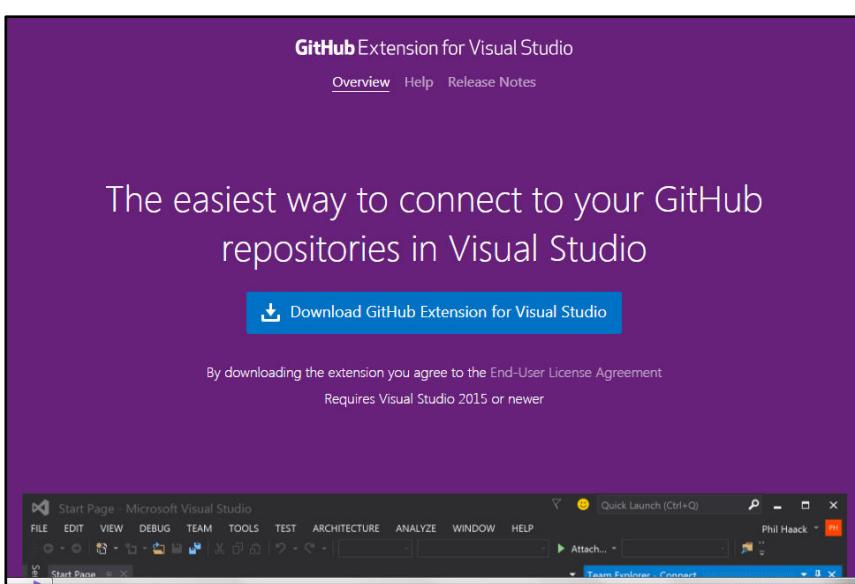
Paragraphs which are in **red** are instances where testing occurred for a *single* trial test - most likely where a problem occurs during development, but multiple tests cannot be made. Paragraphs following the red will be in **green**, which will contain the feedback from the tests, and how I used this feedback to fix the problem - alongside an evaluation of the situation. Black box testing will also be used, using tables - these will be used for a more extensive need of testing where multiple input datas may need to be tested. This **darker green** will also be used to show any improvements I have made upon earlier parts of the program once I have surpassed their key stage within development. This use of agile programming is to regulate any earlier improvements which may need to be necessary as I develop deeper into the code.

I chose the key stages in a certain order based on the chronology of the code's planned development, for example, the lock system will need the database to exist, and the database will need the home screen to exist to take input data, hence I will start with the home screen's development before the database and lock system, and so on.

Advanced features will be implemented during their relevant staging.

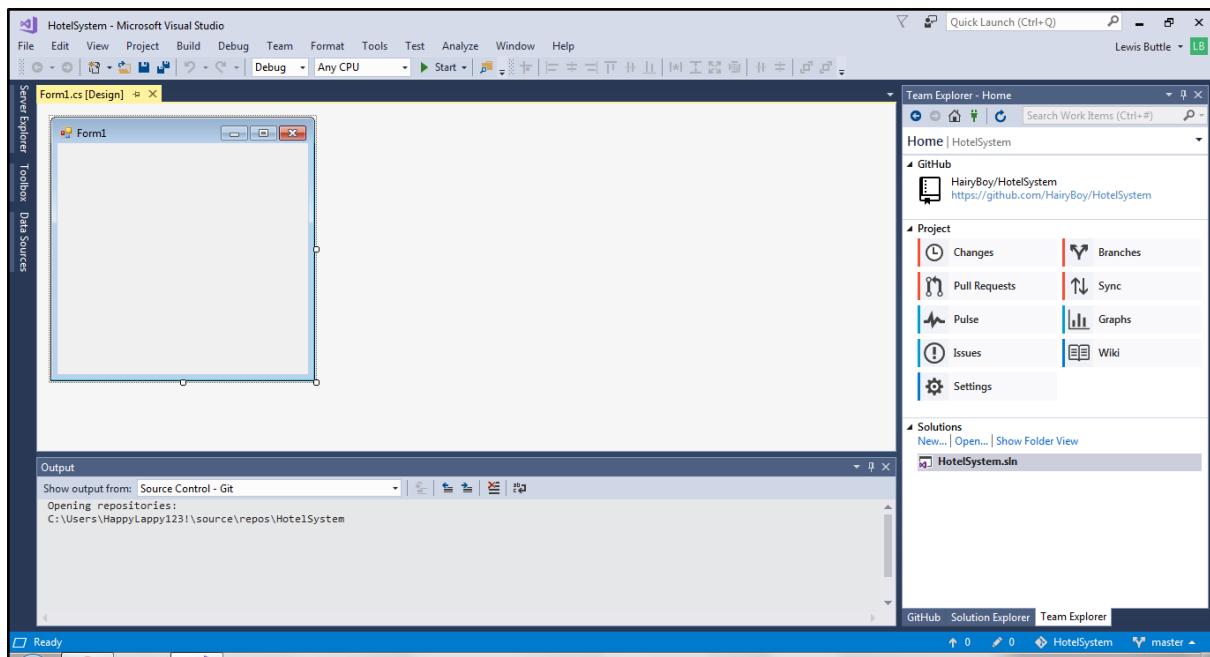
Key Stage 1 - Start up system

The first stage will be to get development started within the chosen IDE (Visual Studio) and ensure that a foundation is made for the future development of the project - such as preparing source control. As well, I will be developing the first form the user is greeted to upon firstly running the program, which will hopefully prove to be a foundation for later stages of development. This stage will be concluded when the setup form is successfully validated and able to store input information in permanent storage, as well as being a working independent solution, ready for the user to move onto the home screen.

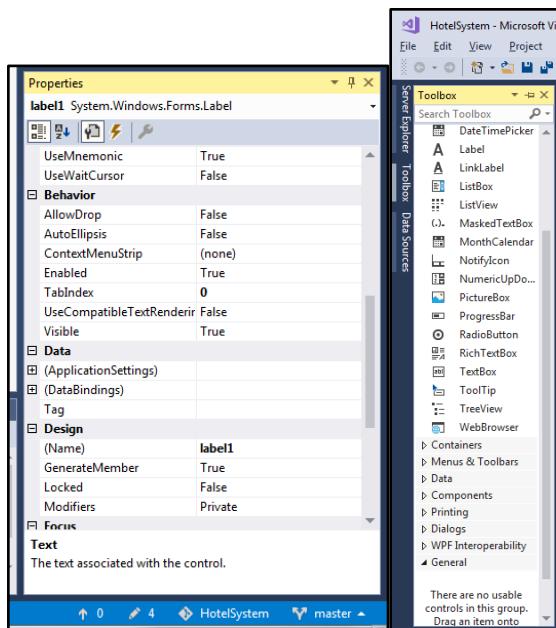


The first step of development was to prepare the IDE I would be working on. I downloaded Visual Studio with its C# components, as well as multiple optional components in the case that they may become needed during the database development.

To also improve development efficiency, I downloaded the GitHub extension for Visual Studio (shown above) which will allow me to identify problems in the code easier, as GitHub allows me to view different versions of the code, labelled by date when they were uploaded (or 'pushed'). It also shows me the differences that were made between development - hence allowing me to research when a specific problem appeared, to make it easier to identify problems. GitHub also backs up the code, and allows me to retrieve it online from any computer - thus keeping the code safe in case of data loss. By saving different versions of code, it also helps in screenshotting code development for captioning. An image of the different version availability is shown above.



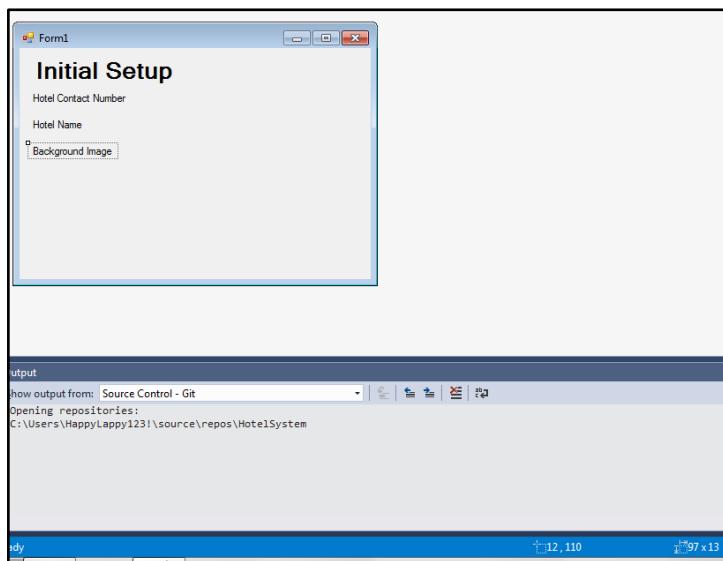
Once loading the Visual Studio, I created a new project based in the C# form appearance. Beginning a new project allows multiple files to be connected together, which is what will be required when the project requires to open a new form from another, or to access a different file. I used form instead of console, as the project is a graphical solution. Upon beginning a form project, Visual Studio opened a default form with interchangeable sizes, and a toolbox for graphical input or output functions.



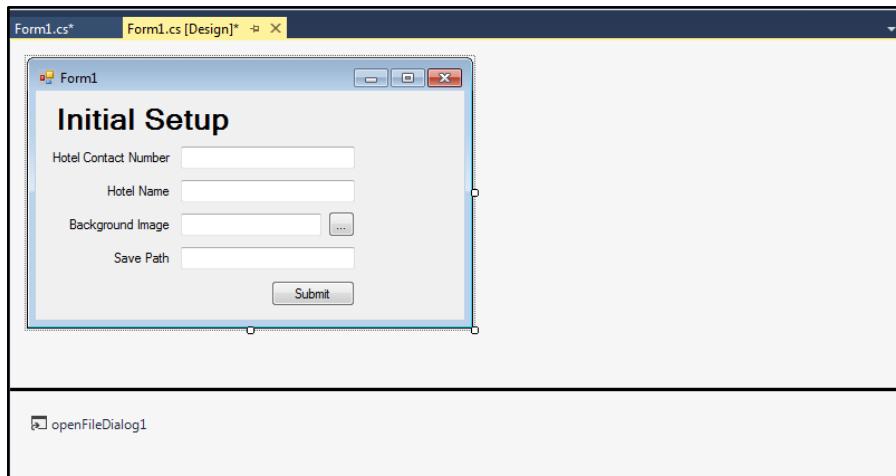
Shown above are the object properties and the toolbox. The toolbox in Visual Studio is great for more efficient development, as it allows me to place the functions of the program by hand. There are many functions

in the toolbox which will be useful for the the project, such as the date tool which allows an input of a date to be read, especially useful for the booking form.

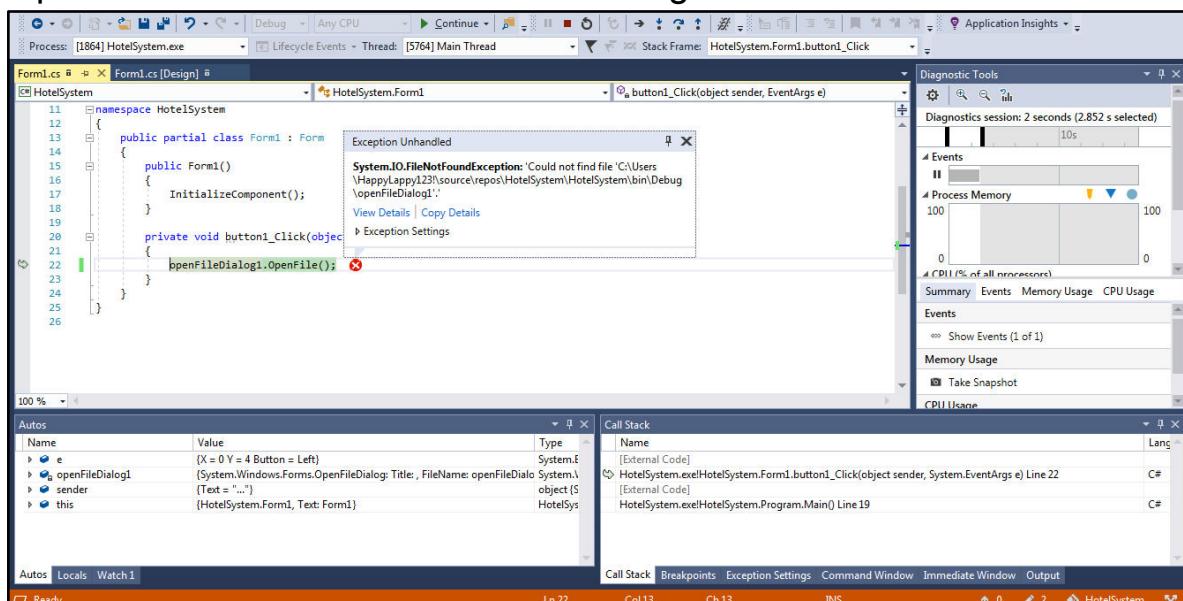
The object properties are used to change the values and properties of the functions, for example in the image, I am editing the properties of a label to change its graphical string value to display “Initial Startup”. I also used the properties to make the “Initial Startup” Bold, and to be displayed in a more titular font. (Shown below).



Shown is the construction of the initial setup interface, to match the initial design. I used the properties described above to create the “Initial Setup” label in bold, thus making it stand out as the appropriate title of the form. Also, advancing from the initial design, I decided to measure and give a real formality to the pixel lengths in between the labels. At the bottom of the above image, the pixel positions are shown in resolution format, thus allowing me to space the labels appropriately. I decided on the smaller labels being spaced by 30 pixels between each other, for the most visually ergonomic effect. After spacing the labels, I continued developing the layout of the form with input functions.

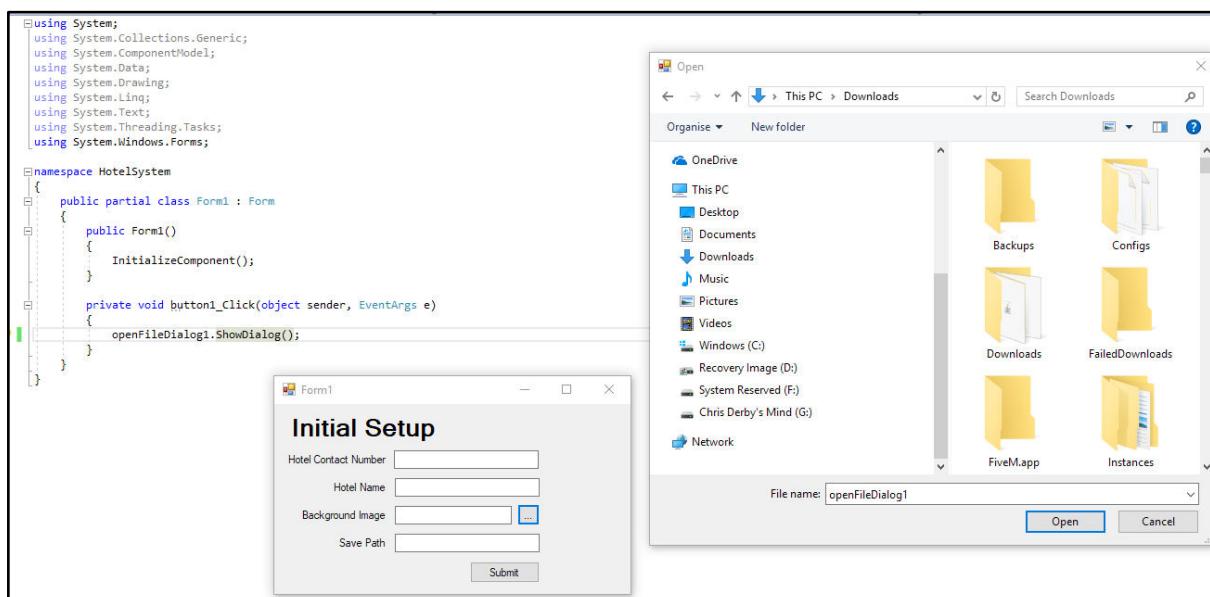


Shown above is the foundation design of the initial setup, and contains all the essential features for the initial setup - ready for coding. The `openFileDialog1` will be interesting to work out, as the function to open a file does not have any initial physicality in the code, it is just a function to stored a file location. To call upon opening this dialog, I set up a button which is labelled “...” which will open the dialog upon being clicked, hence allowing the user to select a file and store it in the text box. This input will need to be validated as an image file.



I began testing on the use of the `openFileDialog` to discover its full efficient uses. I used a range of sources, and my own intuition to experiment with the many different functions. Shown above is my first test output (with the testing cycle ending when pressing the ‘open file’ button opens a file explorer dialog). The first test was initiated when I looked at the different class functions for ‘`openFileDialog`’, and I saw the function called ‘`OpenFile`’ which I supposed sounded like a good start to

test. Upon inputting the class and function into the ‘upon clicking button1’ function; with ‘button1’ representing the input of a click of the browse files button, I ran the program and clicked the button to run the ‘OpenFile’ function - and the above result was shown. The program crashed, displaying the error shown above. The error shown was due to a supposed missing file, which lead me to discover that the initial test of using the .OpenFile function was incorrect in what I wanted the program to do - therefore; learning from this test, I began researching the different FileDialog functions to find the one which will open a file explorer for the user.

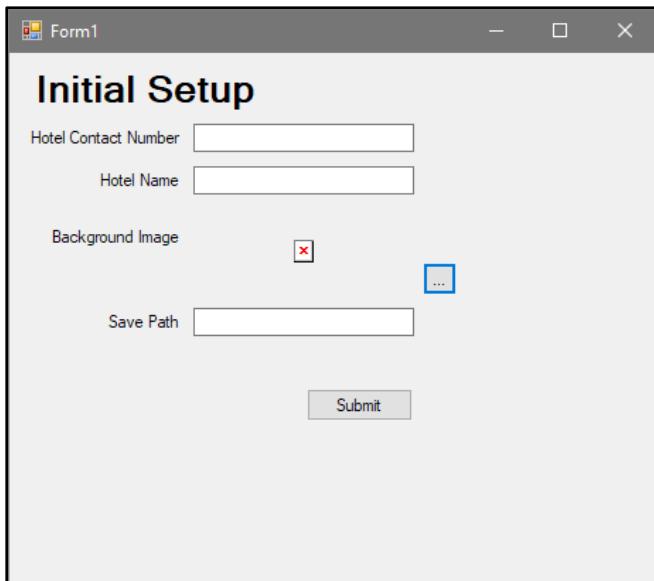


I then researched the different functions, and realized that event handling will need to be introduced into the program for the explorer to activate. I learnt this from testing, as I tested the different functions for the dialog command until I found the one which produces a successful output. This event handling is to coherently link the clicking of the button to the opening of the file explorer - and this is done through using the ShowDialog function (shown above). Once this test was complete, I then began experimenting with how this dialog data was read and inputted into the system, and I began working on how I would display the picture in a quick thumbnail format, for the user to ensure they uploaded the correct image.

```
namespace HotelSystem
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            imgupdate(openFileDialog1.ShowDialog());
        }
        private void imgupdate(System.Windows.Forms.DialogResult file)
        {
            pictureBox1.ImageLocation = file;
        }
    }
}
```

I added a picture box function onto the form, which takes the input of an image location and displays it in a certain space. Width and height can be adjusted, and they will need to ensure that the entire image is displayed - not just warped outside of the picture box. In this image above, it shows the method I created to automatically update this image, and the method is run upon a new file dialog being added by the user. This new file dialog returns a raw file output, which I attempted to pass straight through into the picture box shown above, however, there was a problem with the portability of the data types. I experimented with the different data types that the dialog was outputting (testing shown below).



There was a difficulty which required testing, where the picture box function required the file path of the image in string format, though the dialog was returning a completely bespoke data type. The problem shown is that the picture box was displaying a default error upon the upload button being pressed, showing that the pictureBox received an unexpected and unreadable data type. My thought process then was to find ways to convert the raw file data into its path - as a string. I began testing different ways to convert the data, and I first tested converting the dialog straight into string.

```
Convert.ToString(openFileDialog1.ShowDialog());
```

This was however, unsuccessful as I learnt that the return of 'ShowDialog' was the raw file, not the file path. Thus the conversion was totally inapplicable.

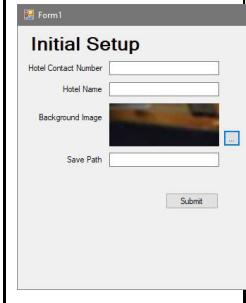
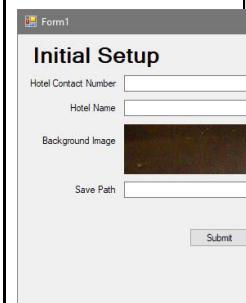
```
private void button1_Click(object sender, EventArgs e)
{
    System.Windows.Forms.DialogResult result = openFileDialog1.ShowDialog();
    string path = openFileDialog1.FileName;

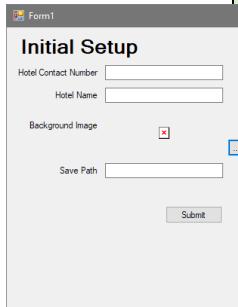
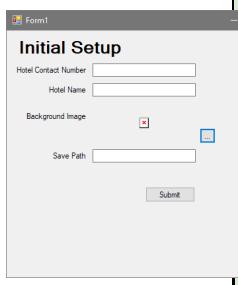
    imgupdate(path);
}
private void imgupdate(string file)
{
    pictureBox1.ImageLocation = file;
}
```

Upon researching this problem, I found out that there was a special function to retrieve the file path of the raw file (shown below), which I then took and attributed it to a new string variable called 'path'. I could then pass 'path' into the picture box function, where it could then

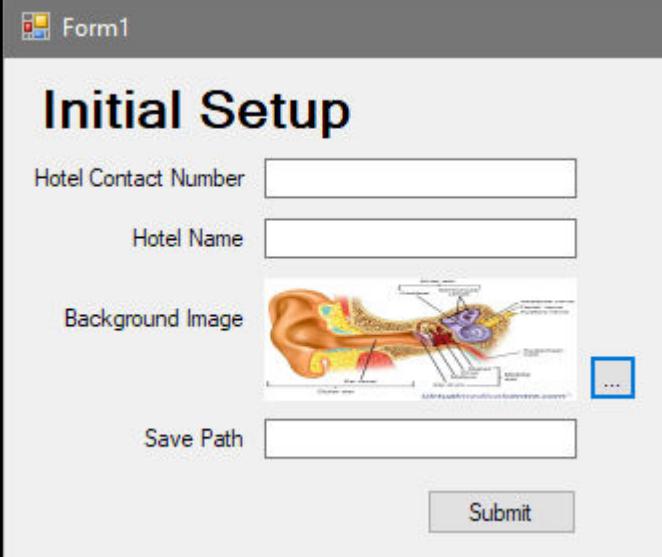
be read and displayed. Shown above is how I created the ‘DialogResult’ variable to try and read the file path from something like ‘result.path’, however, I discovered that this could only be done straight from the ‘ openFileDialog1.FileName’, which outputs a string. I could see that the ‘ShowDialog()’ only needs to be called upon once, for its output to be available and relevant for the method it was called in. The string is then shown to be passed into ‘imgupdate’, and the pictureboxes’ value is then updated.

Shown below is a quick black box test of the newly implemented picturebox.

Input	Expected Output	Output	As Predicted?	Improvement?
Image file	Display the uploaded image		Yes, successfully displayed the image.	Size and resolution validations need to occur, as well as scaling for the rectangle.
Image file followed by another image file	Display the first uploaded image, and then replace the first with the second uploaded image.		Yes, successfully replaced the image without any noticeable error.	Same as above.

Non-Image file (erroneous data)	Provide some sort of default error		Yes, a default 'x' error appeared, obviously showing the error. This could be further validated	Display relevant error information to help the user choose a coherent file.
No image submitted	Provide a default error, or remain the same.		Yes, A default 'x' appeared thus showing a likewise reaction to inputting 'erroneous data'.	Same as above, just with a note such as 'ERROR: No file uploaded.'

```
private void imgupdate(string file)
{
    pictureBox1.ImageLocation = file;
    pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
}
```



The screenshot shows the 'Initial Setup' window with the following fields:

- Hotel Contact Number: [empty]
- Hotel Name: [empty]
- Background Image: [checkbox checked] Anatomical diagram of the human ear.
- Save Path: [empty]
- Submit: [button]

Shown above is how I began to remediate the results of the black box testing, specifically on the point that the image must be scaled to correctly fit the box - and some sort of validation must take place. I did this by using the 'StretchImage' function in the PBoxSizeMode options, which are options specifically designed to changing the size of the picture box depending on the picture, or vice versa. By uploading the image to the picturebox, I can then set for the image to be scaled to the picture box's preset locked width and height - through the 'StretchImage'

function, which is shown to be a variation of the picture box. By setting the unscaled picture box to the new scaled picture box, this then updates the form with the proper image.

Specific image heights and widths will need to be validated later on, depending on how they will fit into the home screen - which is currently yet to be developed. Thus for the moment, as long as there is a placeholder of the full image being displayed and scaled, it can be edited later on.

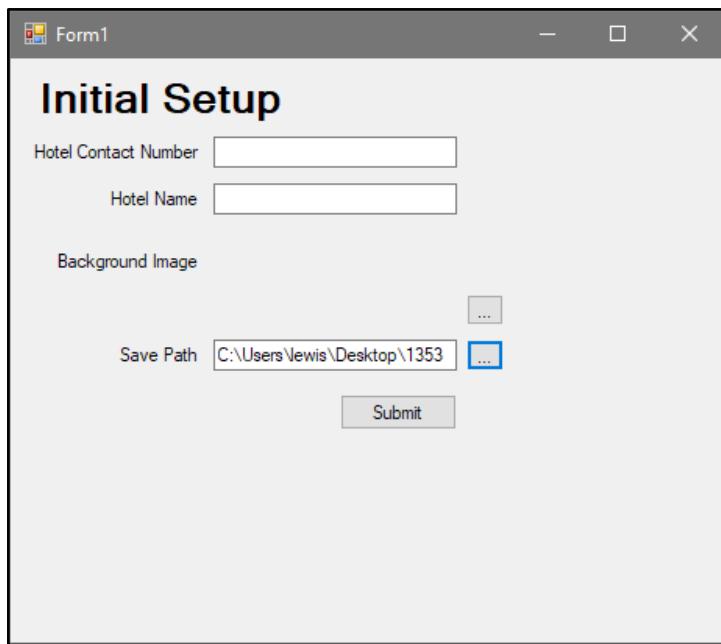
Design	
(Name)	SavePathDialog
GenerateMember	True
Modifiers	Private
Folder Browsing	
Description	
RootFolder	Desktop
SelectedPath	
ShowNewFolderButton	True

With this newly acquired dialog knowledge, I decided to create the function which allows the user to choose a folder for where information (mainly the database, and saved options) will be saved. It will also allow the user to choose a folder to retrieve saved data, if the user expects the program to load pre-saved data.

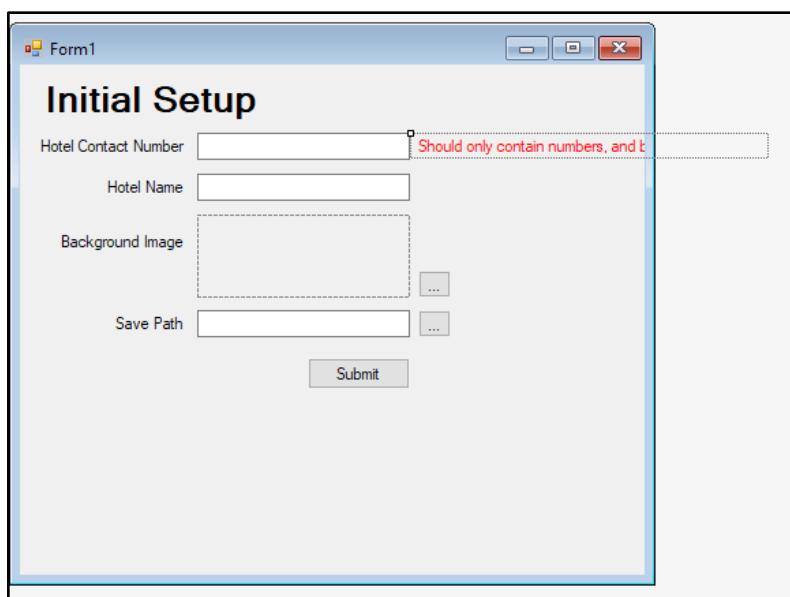
The first difference to the picture retrieval, is that I used a 'Folder Dialog' instead of a file dialog, which has the difference that the function will output a folder location selected by the user, not a raw file. I then plan on using this folder location to try and load a file, and if there is no file, it will create a new text file and write to it - as well as read or create a new database depending on the save path's contents.

Shown in the image is the properties of the folder dialog, and the importance of the 'root folder'. This will be the first prompted folder the dialog loads, and I will need to set this as the default folder of the program when I create the structure of how the files in the program folder are connected. By having this default folder, it will make the finding and saving of data less confusing and more safe for a user.

```
{\n    SavePathDialog.ShowDialog();\n    string SavePath = SavePathDialog.SelectedPath;\n    SavePathText.Text = SavePath;\n}
```

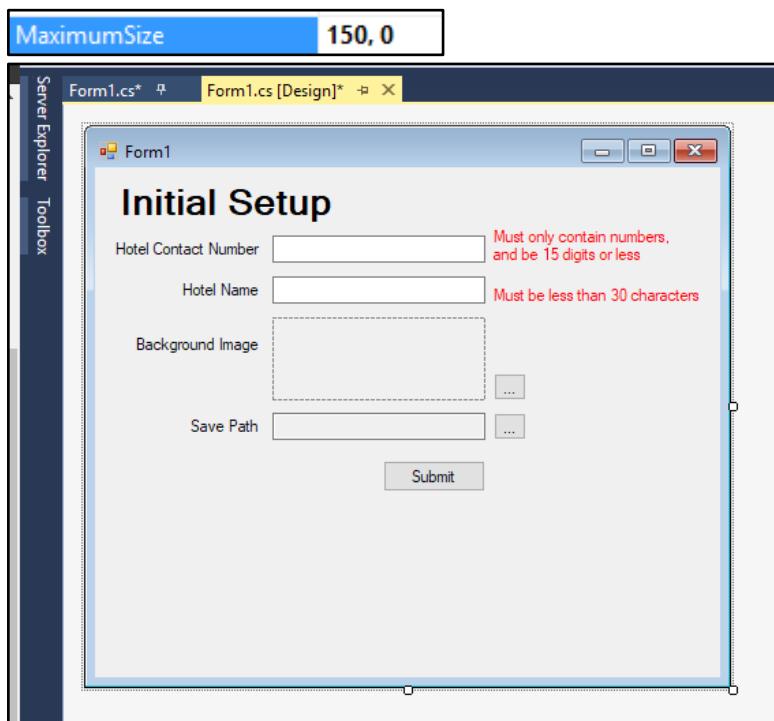


Shown here is the outcome of displaying a save folder path from the user. Similar to the file dialog, using the `.SelectedPath` function once again retrieves the string data of the user's folder selection - thus allowing me to save the string as the 'StringPath' variable. By then changing the text property in the `SavePathText` textbox via the `.Text` function, I am able to display the user's chosen path in the text box for the purpose of clarification. Also, by storing the path as a string in a variable, it will make it much easier to copy the information to a config file once submitted.



I started on validating the setup, which I initially planned to have a method with many 'if' statements to go through each field, and if one of them is not correct to my policy of validation, it will 'unhide' a label which

displays the error with the input information. Shown above is my attempt at placing these hidden labels - however I did have the problem of the text not wrapping. I thought of using two labels, though I deemed this to be a lazy solution and would make it far more confusing to code, so I began looking at ways to limit the label's lengths in the properties. There was a 'maximum size' field which I reduced from infinite, and experimented with a few numbers to see which size would be suitable for the form. The result; with all the hidden labels finished and sized correctly, is shown below, along with the property I changed to allow the text to wrap.

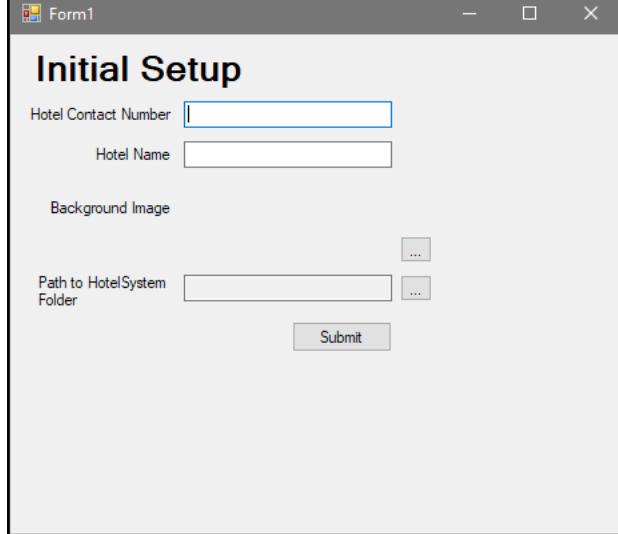


I chose to not require validation text for the Background Image and Save Path inputs, as I found ways to validate them within their functions. Firstly, I set the 'Save Path' text box to read only, meaning the only input the box could retrieve would be from the Folder Dialog function, which is going to give a suitable output no matter what is chosen - hence no validation is required.

```
{  
    OpenFileDialog Background = new OpenFileDialog();  
    Background.Filter = "Image Files (JPG,PNG,GIF)|*.JPG;*.PNG;*.GIF";  
    Background.ShowDialog();  
    string path = Background.FileName;  
    imgupdate(path);  
}
```

For background image, I change the FileDialog code above and use the .Filter function , which allowed me to only allow the dialog to output files

with appropriate image file extensions (JPEGs, PNGs and GIFs). I also rearranged the code, by assigning a named variable (Background) to the occasion of the OpenFileDialog, which meant I could attribute the functions easier with more clarity. Shown in the code, I instantiate the 'Background' variable and a OpenFileDialog data type, before I then put the filter on 'Background' before I the OpenDialog is requested. 'OpenDialog()' is then initiated, opening the explorer window for the user - with the file extension filter attached, before then using the path from the selection to send to the pictureBox function.



At this stage, I planned mentally how the saving system would work - as I began to doubt the use of allowing a user to choose their own save folder. For example, if a user chose their own savepath somewhere on their computer, the location of this savepath would need to be stored somewhere where the system can find it every time the program is opened, hence the save path would eventually need to be saved in a config file within the main program folder - so that negates the use of allowing a user to make their own save folder. Thus, I chose to restrict all saving of permanent information to the HotelSystem folder, and changed the 'Save Path' label, to prompt the location of the location of the HotelSystem folder which will store everything the program requires. This path option was and still is a means of limiting data loss, if the HotelSystem folder is moved but the program still tries to access its old location. This input is now a means of updating this location, just in case. However, this entire function may prove obsolete later on if I discover a method of the program finding its own location which I feel is very likely, hence this feature may be removed later on in development.

```

48     private bool NumbVal (string St)
49     {
50         for (int i=0;i<St.Length;i++)
51         {
52             if (Char.IsNumber(St[i]) == true) { }
53             else if (St[i] == Convert.ToChar(" ")) { }
54             else { return false; }
55         }
56     return true;
57 }
```

Above is the part of the validation for the contact number box, which validates that every character in the input string is either an integer, or a null “ ” space value. Was it not for requiring to check this space value, I could have simply parsed the string as integer or string without the use of the method above. I would rather the code allow contact numbers with spaces, as it helps for users that prefer to split their phone numbers via a space, allowing fluency when communicating the number to a visitor (for example). By allowing spaces in the number, it also makes the contact number field totally optional, as if the string is blank, it will validate the null value successfully. The point of this validation is to ensure that no non-numeric, erroneous data gets display on the home screen.

The method in question work by returning a boolean data type (true if there are only integers and spaces in the string array, false if there is a non-numeric character in the string array.). It uses an iterative ‘for’ loop, by having the ‘i’ integer begin at 0, and increment upon each iteration, only ending the loop once ‘i’ increments to the same value as the length of the string being validated, hence every character in the string will be checked. Upon every iteration, there is an ‘if’ statement which checks **‘Char.IsNumber(St[i]) == true’**

Which uses the char function ‘IsNumber’ to check if a specific char value is a number or not (outputs true if it is a number, false if it is not a number). It then compares the result to ‘true’ via an ‘==’, and if it is true then it will pursue a blank statement, thus continuing to the next iteration to check if **[i+1]** is a character or not.

If it outputs false, at this point we will know that the character is not a number, and the next ‘else if’ will check to see if the value is a “ ” value. By comparing the same “**St[i]**” value as before to see if it is equal (**==**) to the value of “**Convert.ToChar(“ ”)**”, which is just the same as “ ” though it just needed converting from a string to a char, so that the comparison

could be legible, as any input between “ ” is automatically considered a string.

After this, I learnt that using single quotation marks is a way to instantly instantiate a char value, hence the code was slightly improved:

```
else if (St[i] == ' ') {}
```

Thus removing the conversion from string, and reducing the amount of processes required to be performed per iteration.

If the character “**St[i]**” has outputted false for both the previous if statements, then it is certain that the character is non numerical and not a space, therefore the method returns ‘false’ on this final statement, as it will know the string has at least one anomalous data type. Otherwise, if the for loop finishes for every value of the string array, the method will ‘**return true**’, thus clarifying that there will be no need to refuse the information, and the next stage of validation can occur.

```
private void button2_Click(object sender, EventArgs e)
{
    if (NumbVal(ContactNo.Text) == true && ContactNo.Text.Length<17)
    {
        NumbWarn.Visible = false;
    }
    else { NumbWarn.Visible = true; }
}
```

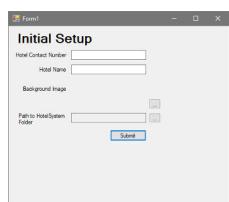
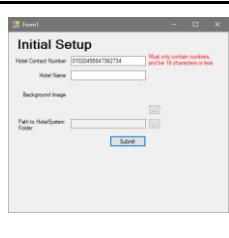
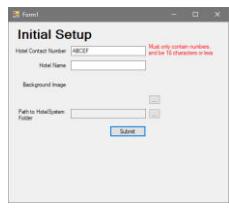
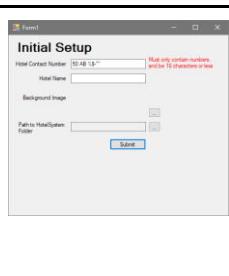
Above is the ‘NumbVal’ method in action, initially implemented into the code which runs when the ‘Submit’ button is pressed.

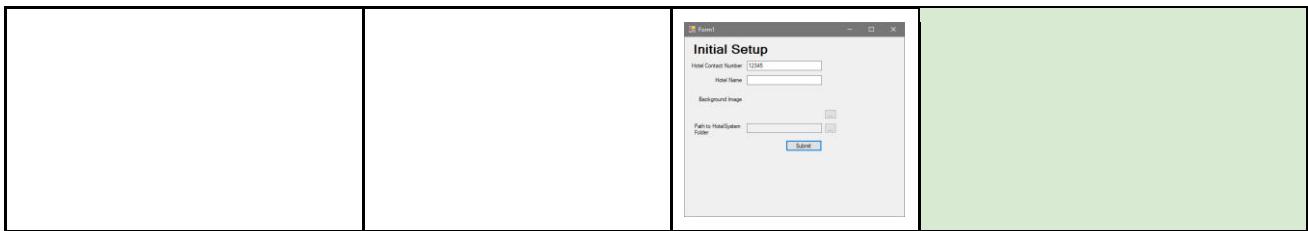
Upon submission, the validation of the contact number field will begin, as an if statement will compare two items using the ‘**&&**’ operator:

1. That the result of ‘NumbVal’ is true, which would mean that the inputted information comprises of only numbers and spaces.
2. That the length of the string is less than 17, which is a number I chose due to the telephone regulation which states that telephone numbers must be 15 digits or less, and having 16 allows for the maximum amount as well as a space.

If these conditions are both met, then the NumbWarn label (The red label used for validation) will have its visibility set to ‘false’, and this is to ensure that if it becomes visible, then it can become invisible once more if the user inputs correct information. If one of the conditions are not met, the ‘else’ statement will ensue which makes the red validation label

visible, showing the user the requirements for the input. I then did black box testing on the validation of this input:

Input	Expected Output upon submission	Real Output	Evaluation
“0155330 29545357” Exactly 15 numbers and 1 space	No validation message appears		Worked as expected, no validation message appeared. No repair required.
“” No data, not even a space	No validation message appears		Worked as expected, no validation message appeared. No repair required.
“010204958473627 34” Exactly 17 numbers	Validation message appears		Worked as expected, validation message appeared, and deemed the input inappropriate.
“ABCEF” Non numeric data	Validation message appears		Worked as expected, validation message appeared, and deemed the input inappropriate.
“50 AB %\$-™ ” A mixture of unicode characters	Validation message appears		Worked as expected, validation message appeared, and deemed the input inappropriate, without error.
“ERROR” submitted, followed by “12345” submitted.	To show if the message becomes visible, then invisible.		Successfully displayed the message, and then retracted the message once the coherent data was inputted.



```
private bool V(string ContactNo ,string HotelName)
{
    bool val = true;
    if (NumbVal(ContactNo) == true && ContactNo.Length < 17)
    {
        NumbWarn.Visible = false;
    }
    else
    {
        NumbWarn.Visible = true;
        val = false;
    }
    if (HotelName.Length < 31)
    {
        label6.Visible = false;
    }
    else
    {
        label6.Visible = true;
        val = false;
    }
    return val;
}
```

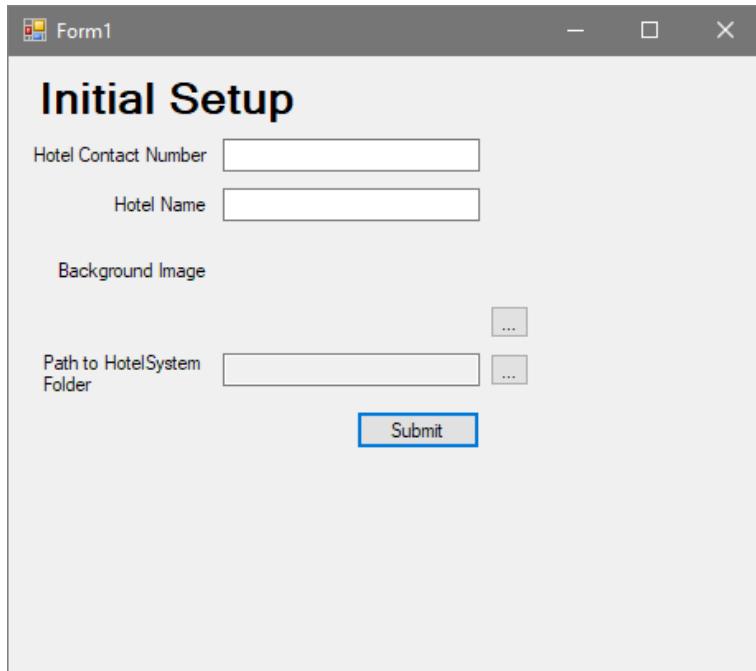
I then finished the validation of both text boxes, though the hotel textbox was much easier to validate due to the input only requiring one term (that it is 30 characters or less).

I had the idea to use a single method to represent the validation, with the output being boolean with 'false' for validation failure, and returning 'true' for validation success. Using this method by passing in the direct text value of the boxes makes the code much more concise and easier to comprehend, by passing only the useful part of the text box.

Also, I had to implement the system which enabled the function to output a boolean answer, hence I initiate a boolean variable called 'val' and give it a 'true' value. From this, there are two opportunities which can return a 'false' if either of the fields are improperly entered, and that is all that is required to ensure the program does not continue if the information is not correct - this means I set 'val' to false in the else statements where the fields are found to not be correctly configured.

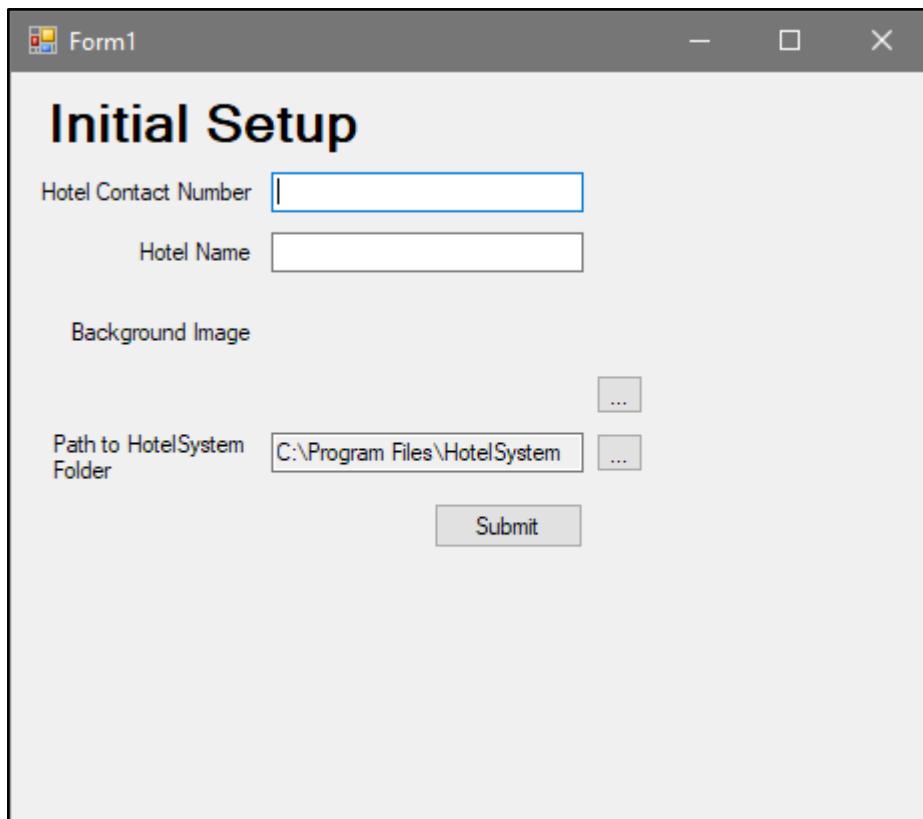
The reason I could not put all three of the validation comparisons into one 'if' statement is because then there would be no way to switch off and on the visibility of the individual labels, they would either have to be both off or both on and would require at least another comparison as there would only be one comparison to make two decisions of the visibility of two labels, hence it would be a problem placing them both in the same if statement.

An important factor of success for this system is that the fields can both acceptably be left blank, which I tested below:



This picture was taken after pressing submit, with both fields blank. This meaning that the initial form is successful in taking a breadth of information, including nothing - which is key to maintaining simplicity in the system.

In response to this test, I learnt that the submission will also work with a blank 'HotelSystem path' field, which should not occur as the system will need a place to retrieve and save the information, hence I added a default value to the field so that leaving the field blank will still technically submit - just with a default value with a default location of the data folder. (This default field is shown below.)



This will be the format of the form from the instance the program is ran, with the 'path' field being already filled with a default value. This default value is simply an imminent folder location which is certain to exist if the user is running the Windows operating system, as C: is the default OS drive for the OS. Whilst 'Program Files' is default to exist, 'HotelSystem' is not, and will need to be created as a folder if required - though this will need to occur once the submit button is selected and ran through a 'if this location exists' function using an if statement most likely. The pseudocode will be as follows:

```
If (FileLocation exists) {}  
Else {Create path 'FileLocation'}
```

This pseudocode means that if the path does not exist (which I imagine there to be a function in Visual Studio to read this), it will create the necessary folders to ensure that this location exists, as it will then be passed onto the next part of the code sequence, which will likely be the creation of files using that location - and if it does not exist then an error is going to occur.

To finish validating this initial setup screen, I then added the above pseudocode into the validation function (V).

```
// HotelSystem Domain
string path = (Path);
if (Directory){ }
```

A quick problem I had with implementing this domain creation, is that the function of ‘Directory’ which I had researched was not usable within the code. Shown above, any use of ‘Directory’ resulted in an error of not recognizing the term’s use.

I resolved this problem by discovering that ‘Directory’ is a sub function of ‘System.IO’, meaning that this had to be called first for ‘Directory’ to be understood.

```
// HotelSystem Domain
System.IO.Directory.CreateDirectory(Path);
```

Shown above is the first implementation of the path validation, whereas it quickly evolved from the pseudocode by completely removing the ‘if’ statement. This development was a necessary improvement, as the function of ‘CreateDirectory’ operates using an in-built if statement already. Normally, the function will create a directory along the ‘path’ variable, though if this directory already exists, it will choose to skip the creation altogether, as there would be nothing to create. Hence, it is much simpler to ignore using an if statement in this case.

I then tested this function:

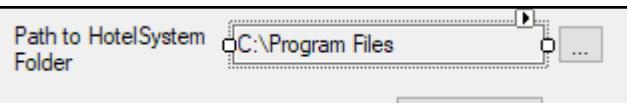
Input	Expected Output	Real Output	Success? Improvement?
Default root directory, but it <i>does not</i> already exist	C:/ProgramFiles/HotelSystem is created		Produced an error, claims that the directory cannot be accessed due to it not existing (See below for evaluation on this result)
Default root directory, but it <i>does</i>	No change, but C:/ProgramFiles/HotelSystem	No Change	Whilst this matches the expected output,

already exist.	remains existing		it is likely not due to making the correct comparisons or working correctly.
Manual root directory, but it <i>does not</i> already exist	The directory path inputted is created	Using the desktop as the manual path: No Change	It was not created, as 'HotelSystem' was not added onto the end of the manual path input, therefore it simply ignored the statement as it was being told to create a directory which already exists - hence it ignores the statement.
Manual root directory, but the directory is deleted before submission.	Both the HotelSystem directory and the deleted directory are created	<p>Path to HotelSystem Folder G:\Test Folder</p>  <p>Using a 'test folder', selecting it in the dialog, and then deleting the folder before submission:</p> <p>Test Folder was created in the deleted location.</p>	It did successfully recreate the deleted directory, however, it did not create the 'HotelSystem' directory, though this is now understandable as I have not given the manual 'HotelSystem' on the end of the manual location unlike the default

location. Hence it will need this ‘HotelSystem’ place added to the end.

Test Improvements

Initially from the testing, it is easy to see where I misunderstood the ‘.CreateDirectory’ function. Firstly, I will improve the system by changing the default path to not include ‘HotelSystem’, and it will instead be added nonetheless to the path via string manipulation shown below:



Changing the default value of the text box.

```
SavePathText.Text = SavePath + "\\HotelSystem";
```

Attempting to manipulate the string using ‘+’, though it will not allow the ‘\’ unicode due to confusion as ‘\’ has another meaning with string manipulation, hence I tried to initiate it using two sets of speech marks:

```
string SavePath = (SavePathDialog.SelectedPath + "\\\"HotelSystem\"");
```

However, the problem with this now is that it ends without closing the string, hence the ‘;’ is in string format. I then worked out that ‘\’ is for excluding speech marks and other confusing string aspects, hence I then tried using two ‘\’ to allow just one to appear:

```
string SavePath = (SavePathDialog.SelectedPath + "\\\\HotelSystem");
```

I quickly tested if this would output either ‘\HotelSystem’ or ‘\\HotelSystem’, by using the stepping tool in Visual Studio, which interrupts the sequence of code flow temporarily, and allows me to see the value of certain variables during this. Hence, I place a step just after ‘SavePath’ is initiated, and checked the value of ‘SavePath’.

37		SavePathDialog.ShowDialog();
38		string SavePath = (SavePathDialog.SelectedPath + "\\\\HotelSystem");
39		SavePathText.Text = SavePath;

The red dot indicates the use of the stepping tool.

SavePath	"C:\\\\Users\\\\lewis\\\\Desktop\\\\HotelSystem"
----------	--

The result of the test shows that it does in fact use ‘\\’, though whilst I initially deemed this unsuccessful, it does appear that the raw form of ‘SelectedPath’ uses ‘\\’, hence it actually does end up working. I thought that it used one ‘\’ as that is how it is displayed in the textbox, but this

would be certain to occur as it needs to display the value as a string, so it would remove all the extra '\' characters. In retrospect, the test was successful.

This string manipulation now ensures the inputted or default path always ends with '\HotelSystem', thus making it successfully create the directory.

At this point, I actually discovered that returning nothing from the directory dialog would remove the default value and just replace it with '\\HotelSystem', so I added a quick 'if' statement to restore the default value, if the dialog returned a null value.

```
private void SavePathClick_Click(object sender, EventArgs e)
{
    SavePathDialog.ShowDialog();
    string SavePath;
    if (SavePathDialog.SelectedPath == "")
    {
        SavePath = "C:\\Program Files\\\\HotelSystem";
    }
    else
    {
        SavePath = (SavePathDialog.SelectedPath + "\\\\HotelSystem");
    }
    SavePathText.Text = (SavePath);
}
```

Above was what I felt was the optimized system for taking the save path location, as it simply replaces SavePath with the default value, if the dialog is returned blank via the 'if' statement, and using the 'else' statement, the SavePath becomes the directory that the user chose. Both paths end with adding '\HotelSystem' onto the end of the string, and both join up to display SavePath into the text field.

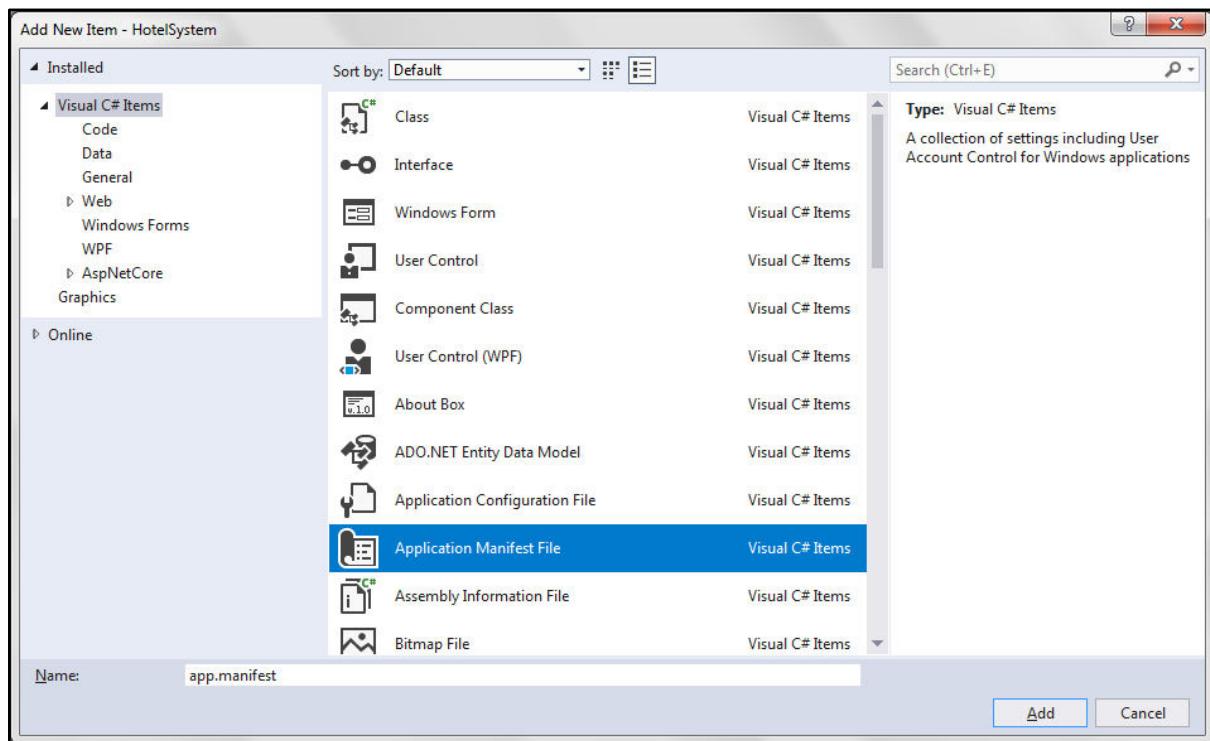
Upon reflection, this entire process may become obsolete during further development, as if a user decides to link to a directory which is already their HotelSystem directory, then it would end up creating a 'HotelSystem\\HotelSystem' directory, which is entirely confusing. From this, I decided it will need to be certain that the initial screen is skipped entirely if the user has already saved data to the program, and this will need to be imminent and certain that the screen is skipped. In reality from this, I will need to add a config file always associated and locatable by the program, which will contain the a boolean variable depending on if the user has ran the program before, where it will then bypass the initial setup. It may be much easier just to keep the default position of the program in 'ProgramFiles', which would eliminate this entire confusion of

giving the user a choice of directory - which would mean shelving this DirectoryDialog system for later in the development cycle, perhaps in the settings. I decided to black box test the directory system once more, and then move onto the creation of the config file as the future risk and volatility of the directory system is not worth spending much more time on - if it may be removed entirely.

Input	Expected Output	Real Output	Success? Improvement?
Default root directory, but it <i>does not</i> already exist	C:/ProgramFiles/HotelSystem is created	 A screenshot of a Windows File Explorer window. Inside the main pane, there is a single folder icon labeled "HotelSystem". Below the folder, the status bar shows the path "C:\Program Files\HotelSystem" and the date and time "26/10/2017 16:41".	Successfully created a folder in the correct location. No errors.
Default root directory, but it <i>does</i> already exist.	No change, but C:/ProgramFiles/HotelSystem remains existing	No change	Successfully ignored creation, as it already exists.
Manual root directory, but it <i>does not</i> already exist	The directory path inputted is created	 A screenshot of a Windows File Explorer window. Inside the main pane, there is a single folder icon labeled "HotelSystem". Below the folder, the status bar shows the path "C:\Program Files\HotelSystem" and the date and time "27/10/2017 15:18".	Successfully created folder in the correct directed position.
Manual root directory, but the directory is deleted before submission.	Both the HotelSystem directory and the deleted directory are created	 A screenshot of a Windows File Explorer window. Inside the main pane, there is a folder icon labeled "Test Folder". Below the folder, the status bar shows the path "C:\Program Files\Test Folder" and the date and time "27/10/2017 15:18".	Surprisingly created both missing directories to ensure the entire path was correct.

At this point, I had to give a quick fix to the directory system as I attempted to run the program on another computer - where the 'AddDirectory' system failed due to lacking permissions to access the 'ProgramFiles'. Researching this problem, I found I needed to add to a

manifest file alongside the program, to either gain the admin permissions from the user, or close the program. This would subdue any errors regarding permission.



Shown is how I added the application manifest file to the program, allowing me to change the permissions required by my program.

```
<requestedExecutionLevel level="asInvoker" uiAccess="false" />
</requestedPrivileges>
```

This was the default line within the manifest file, which has default permissions of 'asInvoker', which does not allow the program to access 'ProgramFiles'. I changed the level of permissions to 'asAdministrator' which then hopefully should allow access the 'ProgramFiles'.

```
-->
<requestedExecutionLevel level="requireAdministrator" uiAccess="false" />
</requestedPrivileges>
```

Upon running the program, Visual Studio restarted and ran in administrator mode, to make up for the permissions.

I then needed to finish the use of the validation (V) function upon clicking the submit button, so that it would either continue to work on the config file, or ignore the config file to validate the information. I first tried using a 'while' loop shown below:

```
private void button2_Click(object sender, EventArgs e)
{
    while(V(ContactNo.Text, HotelName.Text, SavePathText.Text) == true) { }
}
```

Though upon testing this quick function without filling it, I found that the program was sent into an infinite loop - crashing the program. I initially planned to use this while loop (which operates as 'while Validation(passes all the strings requiring validation) is true, run this') as a means of making a decision, but as it crashes because it gets stuck in a permanent iteration if all the variables are correctly validated, it would be more suitable just to use an 'if' statement:

```
if (V(ContactNo.Text, HotelName.Text, SavePathText.Text) == true)
{
    |
}
```

This ensures that validation text is ran 100% of the time upon submit being clicked, but the decision of continuing to write these variables to the config file must be made if the validation is cleared, hence if validation is 'true' then the saving will be made.

```
if (V(ContactNo.Text, HotelName.Text) == true)
{
    //Directory Creation
    System.IO.Directory.CreateDirectory(Path);
    //
```

I also moved the directory creation code from the validation method, to within the 'if validation is true' statement, and I did this to prevent the directory being created every time the submit button is pressed as there was no decision stopping it from creating it every time. Whilst this doesn't directly prevent errors, it just organizes the code and processing a lot more efficiently - to only create this directory once when necessary.

With the directory function successfully working, I began working on the configuration file which would store all this inputted information, to be retrieved on startup whenever the program is ran.

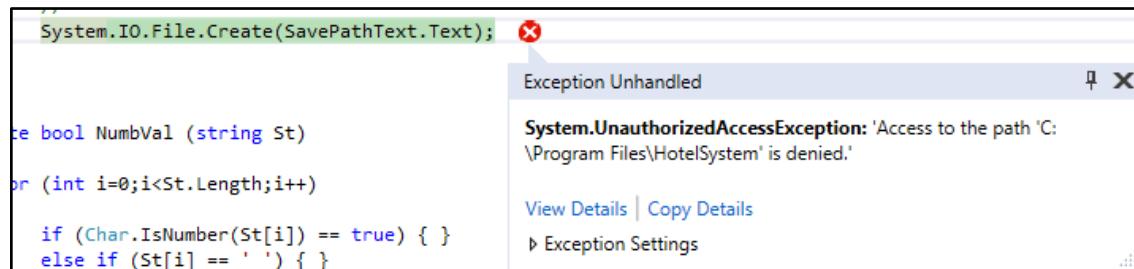
My first idea was to use the streamwriter tool to write to this config file, which would have to be stored with the program files. For matters of simplicity, I am using the default value of the directory - (C:\ProgramFiles\HotelSystem\Config.txt) to store the config file, and any location variance will have to be handled later on. This will give be a solid file which the program will be able to recognize every time upon

startup. Using streamwriter to relate different lines in the file to different parts of information - to organize the information all the better for retrieval. For example, line 1 in the file would be designated to storing the hotel's contact numbers, and line 2 would be designated to store the hotel's name, and so on.

First of all, the config file will need to be created in the position of the save path, and this can be done using the streamwriter functions:

```
if (V(ContactNo.Text, HotelName.Text) == true)
{
    //Directory Creation
    System.IO.Directory.CreateDirectory(SavePathText.Text);
    //
    System.IO.File.Create(SavePathText.Text);
}
```

I began trying to create the file in the savepath position given using 'File.Create(path)', and I quickly tested as to what sort of file would be created, and if a file would even be made. I would need the file to be both writable and readable, so a .txt file would be a suitable choice- though this choice of file is not given through 'File.Create'. I did a quick test, and pressed submit with the default save path to see what the file would become.



For the first test, it created an error upon pressing 'submit' and denying me access to the save path. This was confusing, as I had changed the permissions of the program to requiring admin privileges, hence it would be unlikely that it is denied access due to privilege. I then thought that maybe the path would require an actual filename and file extension, thus solving the '.txt.' problem. Shown is how I added this location to the path using string manipulation:

```
System.IO.File.Create(SavePathText.Text+"\\config.txt");
```

Once again, I had to use the double '\' to ensure that one did appear, and to tell the code that it is to be treated as a string value. By doing this, it allowed me to maintain the save path as a single variable, without interfering with it by only adding 'config.txt' as a one time solution to creating the file.

Upon pressing submit again, there was no error and this was the result:

This PC > Windows (C:) > Program Files > HotelSystem			
System			
Name	Date modified	Type	Size
config.txt	02/11/2017 17:34	Text Document	0 KB

The creation of the 'config.txt' file was successful, as well as being created in the correct directory as well. Evaluating this process, it appears that the 'File.Create' method requires more than just a directory to store a file, it requires the actual filename and extension - which is more understandable now upon reflection. From this testing and development, it has solved the previously discussed problem of changing the file's extension, and now allows me to use StreamWriter to write to the file.

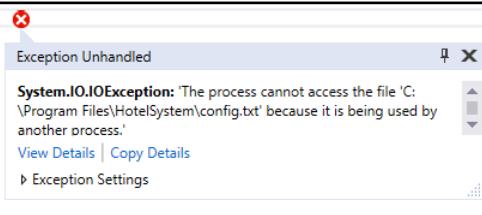
It is useful from this point that it is guaranteed that the config.txt file will not exist prior to this, as the initial setup will only be opened if the user has not opened the program before, meaning the config file will not exist. This means I will not have to validate to see if the file exists or not before writing to it, though this will need to occur to remove error when creating the settings of the program later on in development.

```
if (V>ContactNo.Text, HotelName.Text) == true)
{
    //Directory Creation
    System.IO.Directory.CreateDirectory(SavePathText.Text);
    //
    System.IO.File.Create(SavePathText.Text+"\\config.txt");

    using (var writer=new System.IO.StreamWriter(@SavePathText.Text + "\\config.txt")) {
        Console.WriteLine(HotelName.Text);
        Console.WriteLine(ContactNo.Text);
        Console.WriteLine(pictureBox1.ImageLocation);
    }
}
```

Above is my first attempt at using StreamWriter to write to this config file. I used the 'using' statement to instantiate the StreamWriter, which would then allow me to use 'Console.WriteLine' in the context of the specific file, not using the visual studio console. StreamWriter is the tool function used to write to files, hence calling on it and naming a new variable an instance of StreamWriter (**var writer = new StreamWriter**) allows me to have this instance of StreamWriter editing the config.txt file, which is successfully paved using 'SavePathText.Text'.

After calling on StreamWriter, I then systematically write out the information I want to store via 'Console.WriteLine' which ensures the information is set out on separate lines. After making this quick attempt, I tested to see if this layout of code would work:

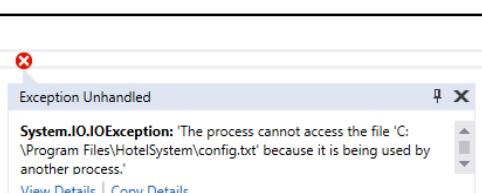


```
using (var writer=new System.IO.StreamWriter(@SavePathText.Text + "\\config.txt")) {  
    Console.WriteLine(HotelName.Text);  
    Console.WriteLine(ContactNo.Text);  
    Console.WriteLine(pictureBox1.ImageLocation);  
}  
  
bool NumbVal (string St)  
(int i=0;i<St.Length;i++)
```

Shown above is the result upon running the code and pressing submit, with successfully validated information. As shown, the code was unable to use StreamWriter to write anything to the file, due to a problem with the config.txt being used by another process. I thought this to be strange at first, especially if the config.txt file had literally just been brought into existence in the previous line of code. I repeated the test, ensuring that a minimal amount of program were open to ensure that no program was using or displaying config.txt - adhering to the same result with no characters being added to config.txt at all.

Testing the next logical step, I tried saving to a different directory which may have less restrictions or usages, but this also displayed the same result - the reflection had now communicated that the error was directly within the code.

I believed that the error must mean that maybe I call upon config.txt to be opened twice, causing this error, and maybe by creating config.txt in the same method as calling on it means it is called twice - so I tried placing the entire StreamWriter command into a new method, and passing the saved parameters through.



```
private void SaveConfig(string Name, string Num, string Loc)  
{  
    using (var writer = new System.IO.StreamWriter(SavePathText.Text + "\\config.txt")) {  
        Console.WriteLine(Name);  
        Console.WriteLine(Num);  
        Console.WriteLine(Loc);  
    }  
}
```

Once again, it was met with the same response.

I tried removing the line which creates the file 'config.txt' to see if StreamWriter would react differently:

HotelSystem			
Name	Date modified	Type	Size
config.txt	02/11/2017 23:53	Text Document	0 KB

Strangely, just calling on the path via StreamWriter would create 'config.txt'. The above result came from removing the line 'File.Create' and submitting valid information. Whilst config.txt was created successfully, and without error this time (most likely due to the fact by using File.Create, this made it so that two instances of config.txt were being used in memory, hence it would already be being processed and an error would occur.) - though it was quite strange that StreamWriter did not write any information to the config.txt file. The file was left blank, despite it being an error.

After discovering this error fix however, I moved StreamWriter back out of its method, and back into its original position.

To evaluate my current position in testing StreamWriter - I have got the solution down to having no imminent errors, though it does not successfully write information to the file.

```
using (var writer=new System.IO.StreamWriter(SavePathText.Text + "\\config.txt")) {
    writer.WriteLine(HotelName.Text);
    writer.WriteLine(ContactNo.Text);
    writer.WriteLine(pictureBox1.ImageLocation);
}
```

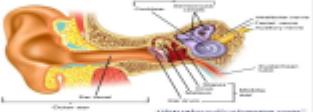
The fix to the problem was quite obvious, as I had used 'Console' instead of 'writer', meaning that the WriteLine commands were being made to a console which does not even exist, as it is a form solution. This threw me off, as using Console as a function produced no errors as it is legible code, hence it was hard to find where I had gone wrong. I tested this immediately after, with successful results:

Form1

Initial Setup

Hotel Contact Number

Hotel Name

Background Image 

Path to HotelSystem Folder

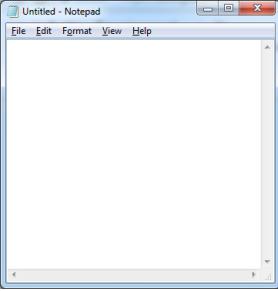
This is the state of the form as I submitted it.

config.txt - Notepad

File Edit Format View Help

Hiya Hotel
0121 7676
G:\6a470ed2e9963912d52dd1c071fbcd2--human-anatomy-and-physiology-ap-human-anatomy.jpg

This is how the form outputted this information, successfully storing the correct information in the correct order. After doing this pseudo test to see that the code worked, I then thoroughly black box tested the stage 1 function in its entirety.

Input	Expected Output	Output	Evaluation
Blank data for every field (except default save path) (Borderline data)	A blank config.txt file.		Successfully created a config file which was blank, with three blank lines of data.

From testing this system, I deemed that this key stage was complete - as it was working as an independent solution, and would now require

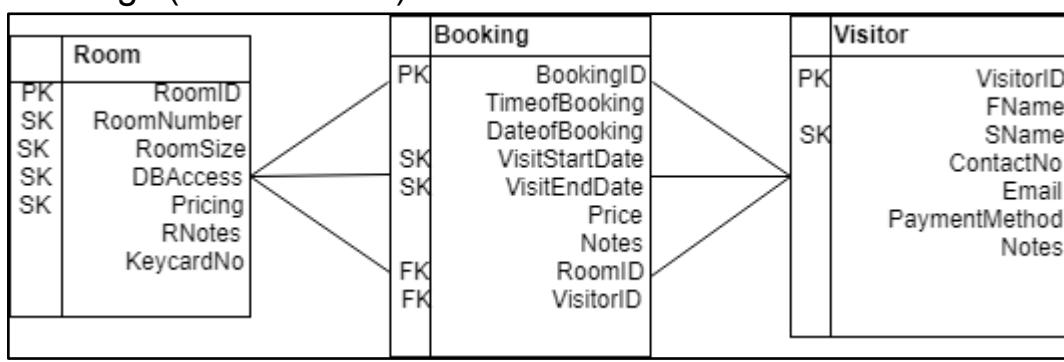
moving onto the foundation for the home screen and database, so that they can then be built upon.

Key Stage 2 - Database Design & Home Screen Foundation

This section will include the beginning development of the home screen and the database.

The home screen will need to have the appearance finished, including the placements of the calendar and the booking graph (though not needing to be functioning, just ready to input and output information), as well as having the buttons correctly placed with their appropriate icons (. This section in terms of the home screen will also need to display the saved hotel name, contact number and background image previously taken from the initial setup. The home screen stage will also include the system of the previously described ‘loading’ system, where the decision is made via the config file as to whether or not to load the home screen straight away (if the program has been ran before and will have retrievable data) or to run the initial setup made in stage 1 (if the program has not been ran before).

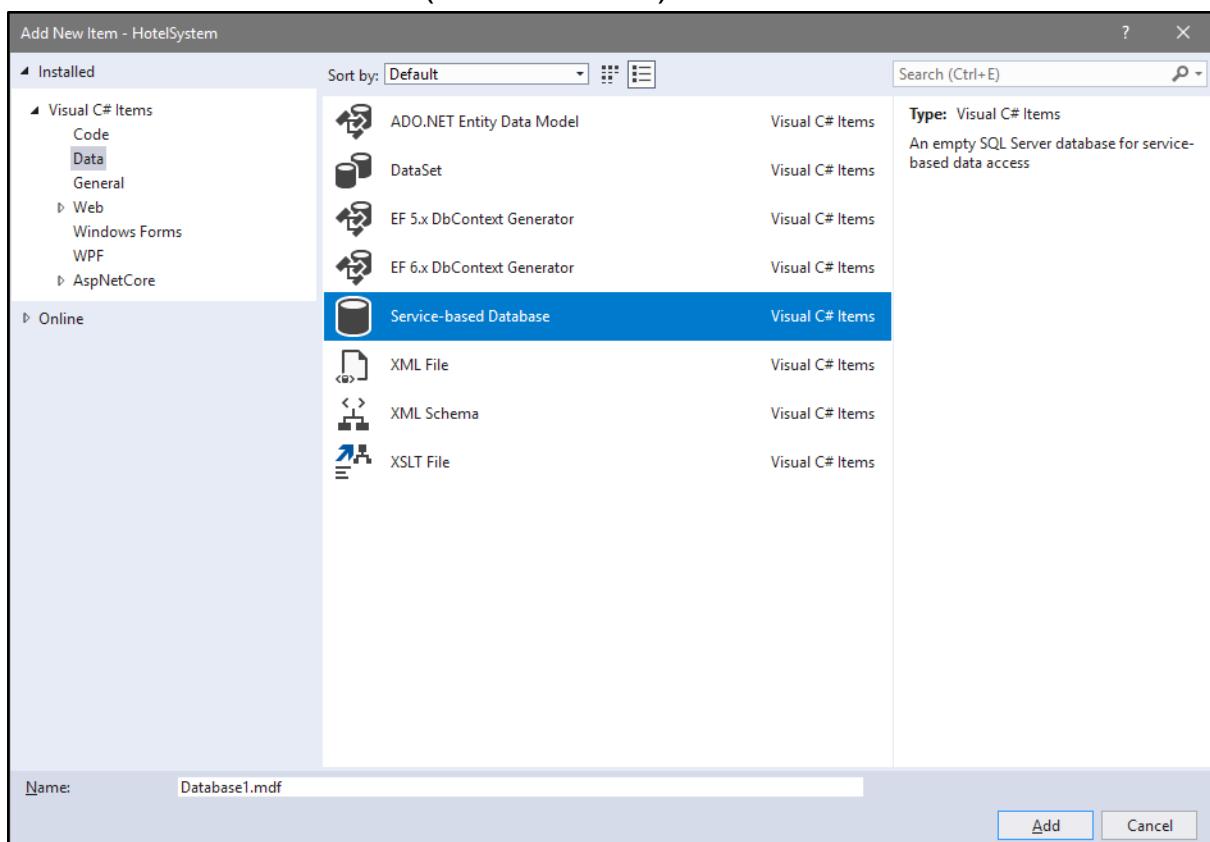
The database is what I will chose to develop first within this stage because of it being much more solid in terms of planned development than the home screen (the home screen’s design would change if the database required it during development, but not vice versa). The database will need to be relational between rooms, customers, and bookings (shown below).



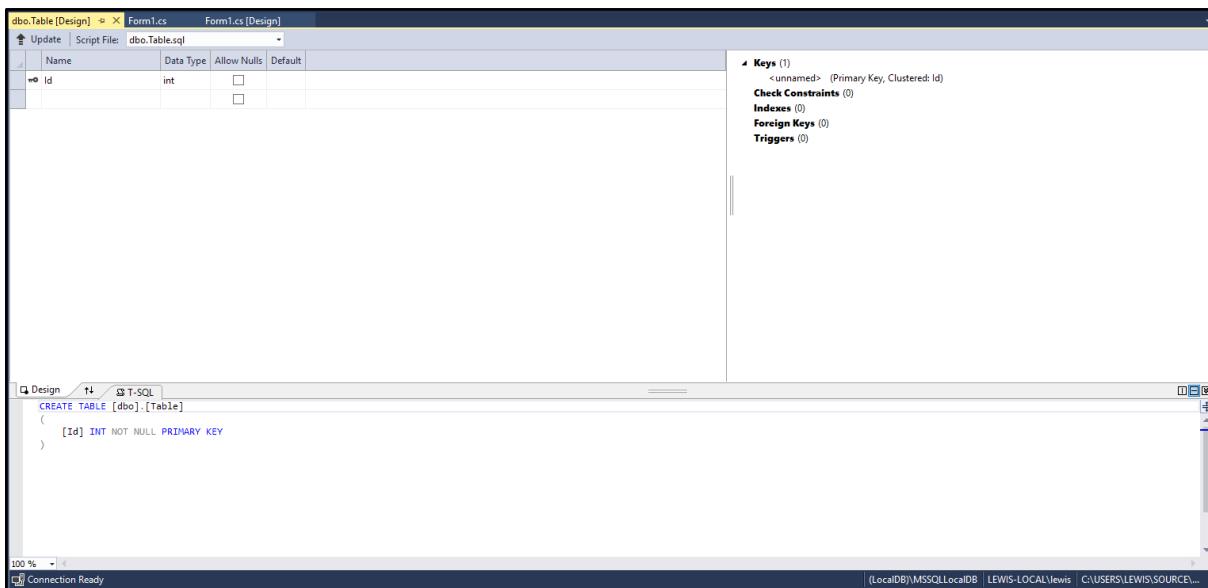
I will then need to added the appropriate fields and entities within these tables. Upon making the tables, connecting their relations via the foreign keys will be the next step in development. Validation within the SQL will then need to occur to every field with an input, and know how to handle any possible invalid data.

As I am quite inexperienced with SQL code and databases, this stage of development will likely contain much more ‘trial and error’ development throughout. Via the SQL, I will also test a few SQL searches within the fields to ensure it is compatible with a search function (for easier development in stage 3). I am also going to explore the method of hashing passwords during storage - as an example of securing data legally for the success criteria and in response to stakeholder’s security concerns.

This stage will be concluded upon both the home screen and database being completed to a level where the next step would be to connect the two solutions together. Once I have developed both solutions to this point, I know they will be ready to be used in solutions which require their combined functions (ie search tool).



I first added the database to the project, by adding the ‘Service-based Database’ item to the solution. I first had to install the necessary SQL components so that I could add the item.



Upon creating the item and adding a table to the database, above was the first screen that came up. Firstly, I recognized the tables in the top left and understood how they will need to be manipulated to hold certain data types in the 'Data Type' column - as well as the 'Allow nulls' column referencing as to whether or not the row will accept null values (total blank values). I also noticed that the first row had a key displayed in its far left column, most likely signalling that this was the primary key. On the right of the screen it appeared to display additional information about the file, as well as info on specific fields. Lastly - the bottom of the screen gave a place to code in SQL to search the database.

After working out these main components, I began fleshing out the fields with the details of the 'rooms' flat file.

	Name	Data Type	Allow Nulls	Default	
1	RoomID	int	<input type="checkbox"/>		
	RoomNumber	nchar(10)	<input checked="" type="checkbox"/>		
	RoomSize	nchar(10)	<input checked="" type="checkbox"/>		
	DBAccess	nchar(10)	<input checked="" type="checkbox"/>		
	Price	nchar(10)	<input checked="" type="checkbox"/>		
	Notes	nchar(10)	<input checked="" type="checkbox"/>		
	KeycardNo	nchar(10)	<input checked="" type="checkbox"/>		
			<input type="checkbox"/>		

Above is how I first filled in the 'name' column of the 'rooms' table, and it seems to default 'Data Type' to 'nchar(10)' which I presumed was a 10 character long string. (After researching, it appears that the 'n' stands for unicode, meaning 'nchar(10)' means a 10 character long unicode string).

```
CONSTRAINT [FK_Bookings_Rooms] FOREIGN KEY ([RoomID]) REFERENCES [Rooms]([RoomID]),
CONSTRAINT [FK_Bookings_Visitors] FOREIGN KEY ([VisitorID]) REFERENCES [Visitors]([VisitorID])
```

Shown above is how I created the foreign keys in the ‘bookings’ table, allowing the table to communicate directly with both the ‘visitors’ and ‘rooms’ tables. In SQL, I used the ‘CONSTRAINT’ command followed by the name of the foreign key (for example in the first case, I called it ‘[FK_Bookings_Rooms]’). It is similar to making a variable in another language, but naming this foreign key as a ‘constraint’ type within the database. After calling it as a constraint, I then named it a foreign key to the row ‘[RoomID]’ thus linking it directly to the RoomID row in the *Bookings* table, but I have not yet referenced it as linking to the other table - this was done through the ‘REFERENCES’ command. By calling ‘REFERENCES [Rooms] ([RoomID])’, this then references another table in the database (Rooms) and then connects to a specific row in that other table (RoomID). Basically, these statements are saying:

The constraint ‘FK_Bookings_Rooms’ is a foreign key, which makes the row ‘RoomID’ in the bookings table equal to the ‘RoomID’ in the ‘Rooms’ table.

```
CREATE INDEX [IX_Bookings_VisitStart] ON [dbo].[Bookings] ([VisitStart])
GO

CREATE INDEX [IX_Bookings_VisitEnd] ON [dbo].[Bookings] ([VisitEnd])
GO
```

Above is a sample of how I allowed certain rows in the tables to be indexed, as it is not a constraint; just an index for searching. I used a simple ‘CREATE INDEX’ to create the index by the name of ‘IX_Bookings_VisitStart’ (IX standing for index), and then used ‘ON’ to represent that this index variable will be upon the row ‘VisitStart’ in the ‘Bookings’ table. The [dbo] references the database, as it begins to scope into the specific row - similar to a directory scoping in like ‘dbo\Bookings\VisitStart’. By creating these indexes, I should be able to use these for SQL queries later on in development.

	Name	Data Type	Allow Nulls	Default
•	VisitorID	int	<input type="checkbox"/>	
	FName	nvarchar(50)	<input type="checkbox"/>	
	SName	nvarchar(50)	<input type="checkbox"/>	
	ContactNo	nvarchar(50)	<input checked="" type="checkbox"/>	
	Email	nvarchar(50)	<input checked="" type="checkbox"/>	
	PaymentMethod	nchar(10)	<input type="checkbox"/>	
	Notes	text	<input checked="" type="checkbox"/>	
			<input type="checkbox"/>	

This is the visitor table, which will store the details of the visitor when the user is making a booking. It will store the contact details, payment method, and any additional notes about the visitor. It also has its unique visitor ID as the primary key.

	Name	Data Type	Allow Nulls	Default
•	BookingID	int	<input type="checkbox"/>	
	VisitStart	date	<input type="checkbox"/>	
	VisitEnd	date	<input type="checkbox"/>	
	Price	money	<input type="checkbox"/>	
	Notes	text	<input checked="" type="checkbox"/>	
	RoomID	int	<input type="checkbox"/>	
	VisitorID	int	<input type="checkbox"/>	
			<input type="checkbox"/>	

This is the booking table which stores the information about a specific booking, and is information which is for a temporary event - unlike the rooms and visitors table which represents more 'permanent' information. It stores more temporary data, as it stores the start and end date of the planned visit. It will also calculate a price through a multiplication of **(Length of visit in days * Price of room)**. It will take the information of the room price via the RoomID foreign key, which should hopefully allow the booking table to access all of the field entries in the Rooms table - though I am quite skeptical at this point in development as I believe that inputting 'RoomID' as a foreign key will only allow access to that specific row in the rooms table, not the entirety of the booking table. This will be tested later on, however.

	Name	Data Type	Allow Nulls	Default
RoomID	int	<input type="checkbox"/>		
RoomNumber	nchar(10)	<input type="checkbox"/>		
RoomSize	nchar(10)	<input type="checkbox"/>		
DBAccess	bit	<input checked="" type="checkbox"/>		
Price	money	<input type="checkbox"/>		
Notes	text	<input checked="" type="checkbox"/>		
KeycardNo	nchar(10)	<input checked="" type="checkbox"/>		
		<input type="checkbox"/>		

This is the room table, which will store information about every room in the hotel, such as its size and if it is suitable for a visitor with a physical disability. Appropriately, I made these room factors the index keys of this table - as they will be the most important information for comparison when searching for a room (for example, a user won't want to search for rooms depending on just their number, as it is irrelevant to visitor's needs). This will be the main table used for the search function, as the search function will need to SQL search every room entry, and then also check the booking field as to the room's availability. Price will be a fixed value inputted by the user upon creating all the hotel's rooms, as to how much the room will cost per night. Additionally for the lock system, the table will store the key cards which are able to open the door.

Name	Data Type	Data Type Justification	Allow Nulls	'Allow Nulls' Justification
VisitorID	int	Must be unique, and incrementable to ensure uniqueness - hence a number value can satisfy this.	No	As it is unique, by default it would never be able to be blank.
FName	nvarchar(50)	I chose 50 unicode characters as the next lowest down maximum was 10, which is quite risky in terms of name lengths as a visitor's name may exceed 10	No	A visitor must provide some sort of identity to ensure it is correctly recorded, as it will not be an

		characters. Hence 50 is a safe choice.		optional field.
SName	nvarchar(50)	Same reason as above.	No	Same reason as above.
ContactNo	nvarchar(50)	I did not believe 'int' to be suitable despite it being a number, as it is not going to be worked arithmetically - it will simply need to be stored and shown. Also, the number may require the use of '' or '+' values, which would create an error if the value was integer. Therefore, 50 unicode characters is the most error-resistant data type for this situation	Yes	I believe that as there is a chance of a user not using a contact number, then the method of contact should be optional - as there is a chance that no data may be inputted.
Email	nvarchar(50)	As an email will require many different character types, unicode to a lengthy 50 characters will be suitable to take in the entire range of possible email addresses.	Yes	Similar to above, a visitor may not have an email - therefore there is a chance null will be expected.
Notes	text	The 'text' data type allows a large block of unicode characters to be stored, which is suitable store extensive notes on a visitor which could vary in a large amount of length. However, it will need to be validated to ensure it does not contain volatile unicode characters to	Yes	It is not vital to the system that notes are stored, as it is an optional input - therefore a null value may be expected.

		protect the database.		
Payment Method	nchar(10)	This will likely be a validated field with a drop down function - with the field being limited to 'Cash', 'Card' or any other modern methods of payment.	No	As there will always be a method of payment, even if it is discounted to nothing, there should be a piece of information to represent this transaction.
BookingID	int	Must be unique, and incrementable to ensure uniqueness - hence a number value can satisfy this.	No	As it is unique, by default it would never be able to be blank.
VisitStart	date	A date value is helpful in storing dates, which will be necessary to storing the date of expected entry from the visitor - thus working with the VisitEnd date value to work out the amount of nights being booked in the hotel, thus allowing the calculation of price. The difficult may be using arithmetic to work out the difference between these two dates, as I may need to create a function to work this out especially.	No	A date must be given so that a price may be calculated for the booking, thus it is a mandatory input and null cannot be accepted.
VisitEnd	date	See above.	No	See above.
Price	money	The money data type is useful in taking double values, but rounding them	No	The system will always need a value

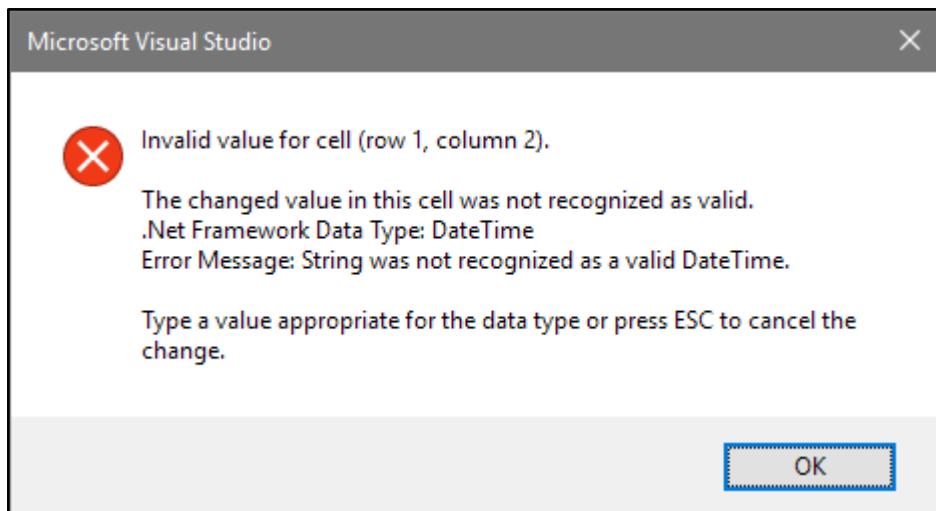
		upon 2 decimal places - thus maintaining the same form as money. It also works for a broad range of money unicode ('£', '\$', etc), thus giving portability to the system. This will make working with the arithmetics of the money quite simpler, except that the database will only accept values to two decimal places, meaning they may need to be validated before being sent to the database through the form.		for money for the system to function properly, as to prevent accidental free payments. Even if a payment is manually set to be '0.00' this will still be an expected value, instead of just null.
Notes	text	The 'text' data type allows a large block of unicode characters to be stored, which is suitable store extensive notes on a visitor which could vary in a large amount of length. However, it will need to be validated to ensure it does not contain volatile unicode characters to protect the database.	Yes	It is not vital to the system that notes are stored, as it is an optional input - therefore a null value may be expected.
RoomID	int	Must be unique, and incrementable to ensure uniqueness - hence a number value can satisfy this.	No	As it is unique, by default it would never be able to be blank.
VisitorID	int	Must be unique, and incrementable to ensure uniqueness - hence a number value can satisfy this.	No	As it is unique, by default it would never be able to be blank.

RoomID	int	Must be unique, and incrementable to ensure uniqueness - hence a number value can satisfy this.	No	As it is unique, by default it would never be able to be blank.
RoomNumber	nchar(10)	I do not imagine a room number to be more than 10 characters long, hence I have chosen for it to be represented in unicode within 10 characters. Once again it will not be a number requiring arithmetic functions, so storing it as an int value would not be suitable - though a validation to check if it is a number will be required.	No	A room number must be attributed to a booking, hence it will not be an optional piece of data, a null value should be rejected.
RoomSize	nchar(10)	This data will only need to be communicated simply, maybe even on a mnemonic level so 10 characters would be plenty to communicate this. It will be a validated drop down field which has preset values to choose from (Double, Single, Luxury, etc).	No	It is once again a valuable piece of data to the booking and the search function, by attributing necessary information to the room to be searched for - otherwise it would be a useless entity to a user.
DBAccess	bit	A bit value will store a single boolean expression (1,0), which suits this field as a room can either be suitable (1, true) or	No	As it has a boolean data type, there will be a 100% chance that

		unsuitable (0,false).		the field can be satisfied with just the expression, hence null will not be necessary to represent anything in the field.
Price	money	The money data type is useful in taking double values, but rounding them upon 2 decimal places - thus maintaining the same form as money. It also works for a broad range of money unicode ('£', '\$', etc), thus giving portability to the system. This will make working with the arithmetics of the money quite simpler, except that the database will only accept values to two decimal places, meaning they may need to be validated before being sent to the database through the form.	No	The system will always need a value for money for the system to function properly, as to prevent accidental free payments. Even if a payment is manually set to be '0.00' this will still be an expected value, instead of just null. Every room should have a specified price, or else there will be errors in the final price arithmetics.
Notes	text	The 'text' data type allows a large block of unicode characters to be stored, which is suitable store extensive notes on a	Yes	It is not vital to the system that notes are stored, as it is an optional

		visitor which could vary in a large amount of length. However, it will need to be validated to ensure it does not contain volatile unicode characters to protect the database.		input - therefore a null value may be expected.
KeycardNo	nchar(10)	The RFID chip in the keycard should have a unique piece of data attributed within it, and having 10 characters to keep this unique should be quite easy.	Yes	This should expect a null in the case that the hotel using the system maybe does not require keycards (a bed & breakfast for example). This means that not every room requires a keycard.

Upon finishing the tables, I then ran a quick test to check if they were able to successfully store data. I selected the ‘dbo.Bookings[Data]’ file which is the physical file which contains and stored the data, using the template from the [Design] file. The first record of the data file was filled with a default “NULL” for every field, and upon changing a few values (some specifically not to match the datatype) - I was met with this response:



This means it successfully caught the error via the data type and did not store the data permanently. This means that when I come to connect the form and the database together, I must either:

1. Heavily validate the information in the form to ensure no errors can occur within the database, by disallowing it access to erroneous data - and if successfully done there will not need to be any error catching necessary within the database (validating in the form).
2. Make sure the database knows how to handle errors, and return the storage request back to the C# form to then produce an appropriate error in the form. (validating in the database).

As there are three methods I will be entering the database from the form, I have organized as to which validation will be suitable to which method:

- Retrieving data - may require no validation if data is expectedly stored, though any retrievals should be able to be validated via method 1.
- Storing data - This will need to be validated via method 1, as it would be much more professional and organized to use classes to validate the information before storing - thus providing no errors.
- SQL Searching data - This may need validation from both methods, as a search of a large string would be difficult to limit within validation - as a huge range of input data could be expected. Thus, I could lightly validate it in the form, and use the database validation to validate as to whether or not a record exists to match the search parameters, returning an error to the form via method 2 if the search result is a failure.

	BookingID	VisitStart	VisitEnd	Price
	1	NULL	NULL	NULL

Shown is how the database is saved via commits - meaning that this commit will need to be called by my form after every change. As to whether this is automatic or not, that will be developed in stage 3. At this stage, I believe that the database is developed far enough for its next development to require the homepage, as it is able to store information successfully. Hence, I moved onto the development of the home screen UI.

The initial problem with creating this new UI was the method in which I would switch between the forms. I was reluctant to use the method of morphing the same form to look and act as many different forms, as I feel this would inherently be difficult to maintain organization with, and also likely to create more errors by needing to hide and resize the form's elements. I was also reluctant to create a new file to store this form, thus loading this file every time the system required it - as this would use quite a lot of memory and unnecessary processing to maintain needing to switch between these forms so often, especially on the booking form which would be opened quite often.

Through research I found a tool called tab control, which allows the user switch between tabs in the form - though this method will not be useful for switching to the home page from the initial setup, it may be useful later on for simpler access to the booking page.

In the end, the method of changing forms I chose to use was to create a form class, which inherits the properties from the original 'Form' the same way which the initial form uses - allowing two forms to be compiled simultaneously.



Shown is how I mean to say that I added a new form. I added this new form 'Form2' with only as [Design] file, unlike Form1 which has both [Design] and .cs (code) files. Form2's class will need to be created manually within Form1.cs's code, as the [Design] form for Form2 literally doesn't exist without assigning it to the form class - which Form1 is assigned to automatically.

```
private void HomeCall()
{
    Form2 HomeScreen = new Form2();
    HomeScreen.Show();
}
```

The HomeCall() method shown above is a method which hopefully is the gateway to accessing the home screen, and any route to this method should ensure that the program has the correct input information to progress without error. I make the variable 'HomeScreen' of datatype Form2, which is an object represented through the Form class. By pronouncing it as a new form, and representing the form I created previously (Form2), it allows me to call on the 'Show()' method to make the form appear. I then worked on linking the correct stages of the initial setup to this method.

```
private void SavePathClick_Click(object sender, EventArgs e)
{
    SavePathDialog.ShowDialog();
    string SavePath;
    if (SavePathDialog.SelectedPath == "")
    {
        SavePath = "C:\\Program Files\\HotelSystem";
    }
    else
    {
        SavePath = (SavePathDialog.SelectedPath + "\\HotelSystem");
    }
    if (System.IO.File.Exists(SavePath + "\\config.txt"))
    {
        HomeCall();
    }
    SavePathText.Text = (SavePath);
}
```

Shown on the selected line, it calls on HomeCall() to be ran if 'config.txt' exists in the user's selected directory input. This means that if a user selects a directory where the config.txt file has already been made, the program will break sequence and not overwrite this config file - instead choosing to go straight to the home screen using the previously saved data. If a config file already exists at this point in the code, this would mean that the user has ran the program before - hence the program will

be able to go to the home screen and expect a database to have retrievable data for startup.

```
private void Form1_Load(object sender, EventArgs e)
{
    if (System.IO.File.Exists("C:\\Program Files\\HotelSystem\\config.txt"))
    {
        HomeCall();
    }
}
```

This will be the mainly used function to check if the user is eligible to enter the home screen. Through the 'Form1_Load' method, this method is ran upon the form being loaded - meaning this should be the first manual piece of code checked as the program is first loaded. It makes an if statement checking the default location for the config file, and if this file exists then it will know to run 'HomeCall()'. Otherwise if the user is using a location different to the default path, they will have the opportunity to enter this location and have it checked for access to the home screen.

```
private void button2_Click(object sender, EventArgs e)
{
    if (V(ContactNo.Text, HotelName.Text) == true)
    {
        //Directory Creation
        System.IO.Directory.CreateDirectory(SavePathText.Text);
        //
        using (var writer=new System.IO.StreamWriter(SavePathText.Text + "\\config.txt"))
        {
            writer.WriteLine(HotelName.Text);
            writer.WriteLine(ContactNo.Text);
            writer.WriteLine(pictureBox1.ImageLocation);
        }
        HomeCall();
    }
}
```

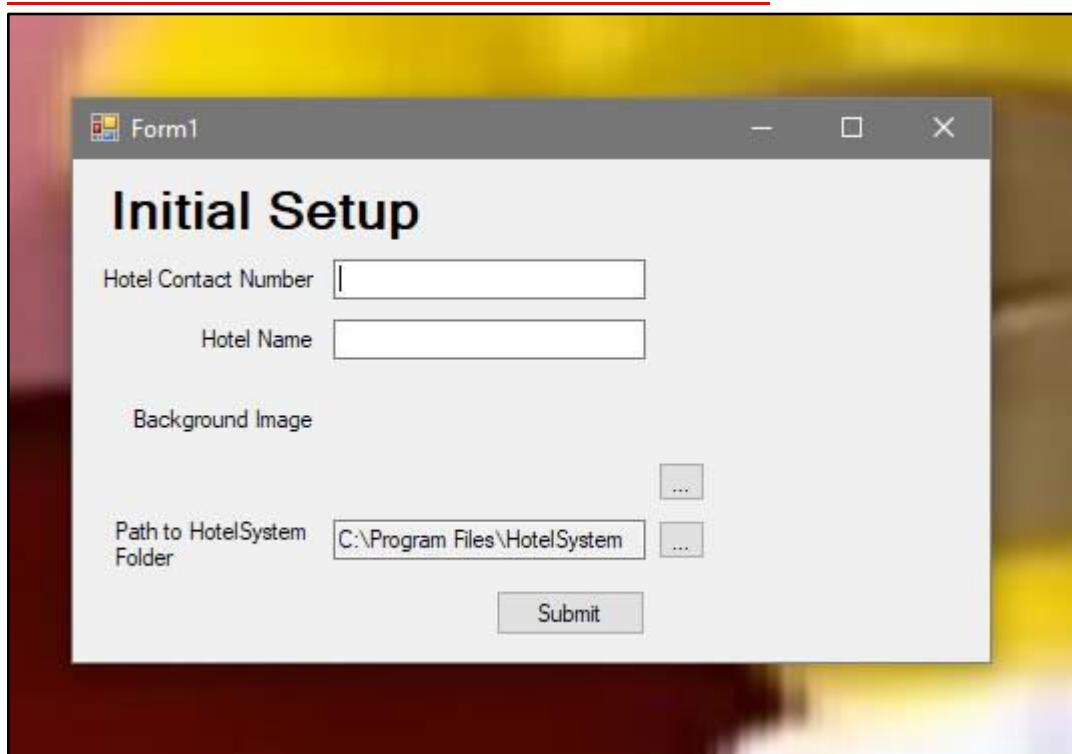
This final access method of the home screen will be the method a first time user accesses the home screen, as it is called upon within the 'submit button' being pressed, and once the validation of the input values is completed and have been written to the config.

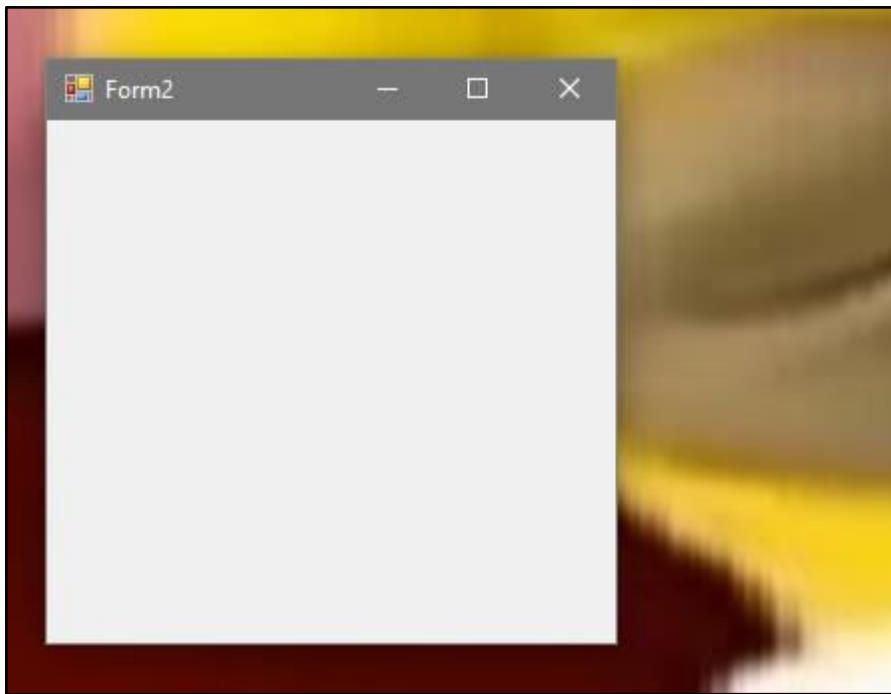
A problem I was now encountering was the fashion in which I would hide Form1 and then show Form2 to represent the transition within HomeCall().

```
private void HomeCall()
{
    Hide();
    Form2 HomeScreen = new Form2();
    HomeScreen.Show();
}
```

Shown is how I used 'Hide()' in relation to 'this.Hide()' to attempt to hide 'form1' as 'this' would represent the current form - which is Form1. Following this, creating the 'Form2' variable and then showing it allows the transition from the initial setup to the home screen. This is how I initially planned it to work, for the setup to disappear and then be replaced by the home screen. Upon testing, it seemed the function worked differently depending on how HomeCall was accessed. Shown are the different results from accessing HomeScreen from multiple sequences in the code.

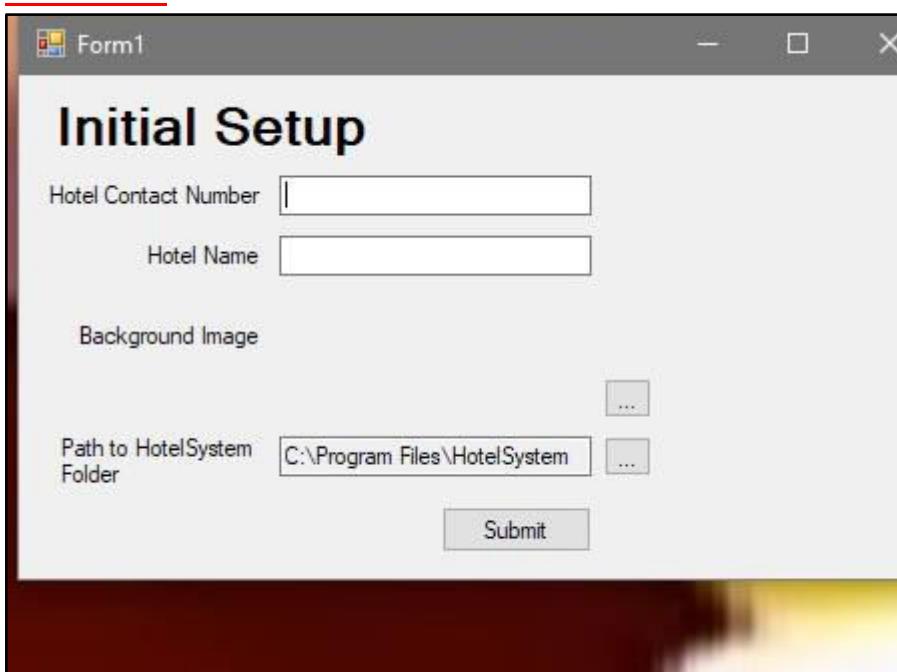
When HomeCall is called from after Validation

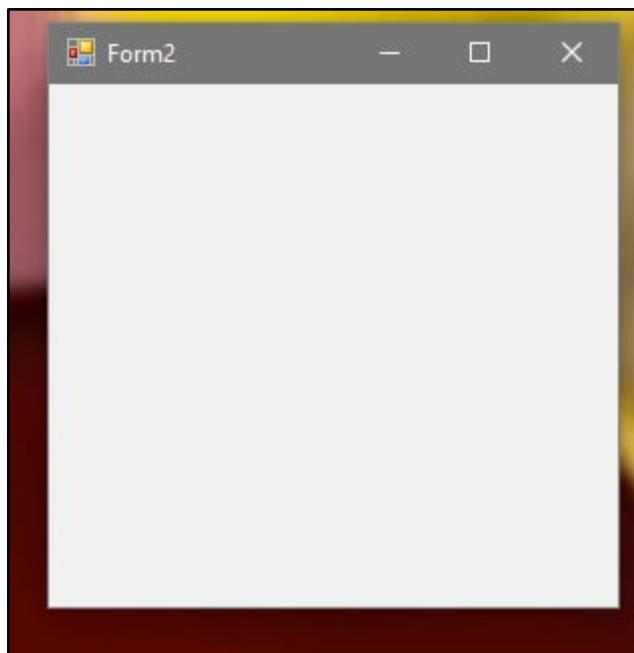




Hide and show works as expected, showing only the initial setup - and then hiding this setup form and displaying the home form after submitting. This is successful and expected.

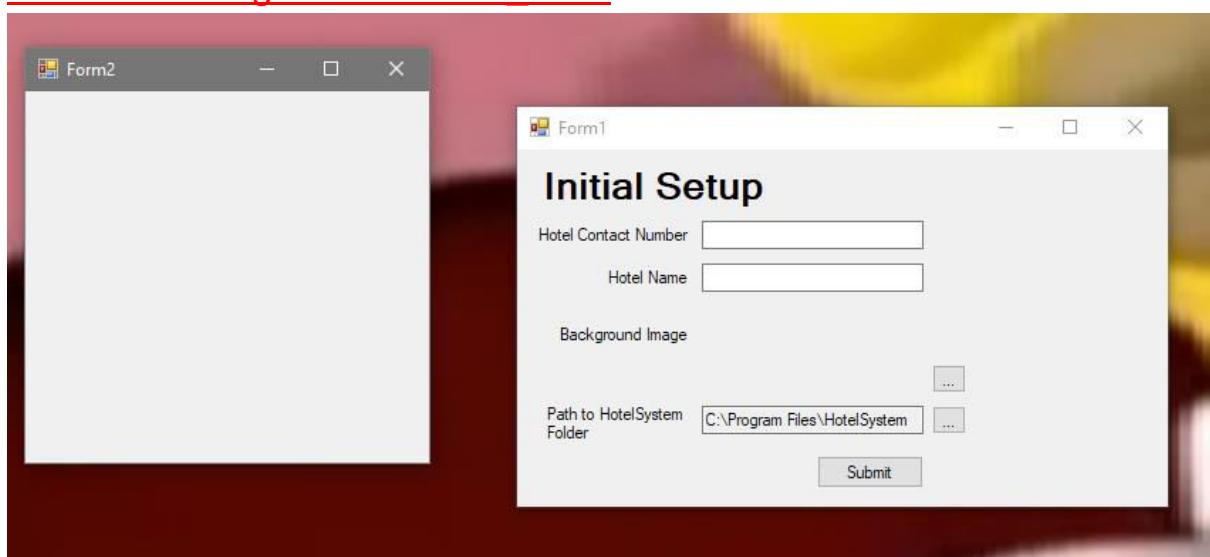
Accessing from after closing the directory Dialog - before the Config file is created

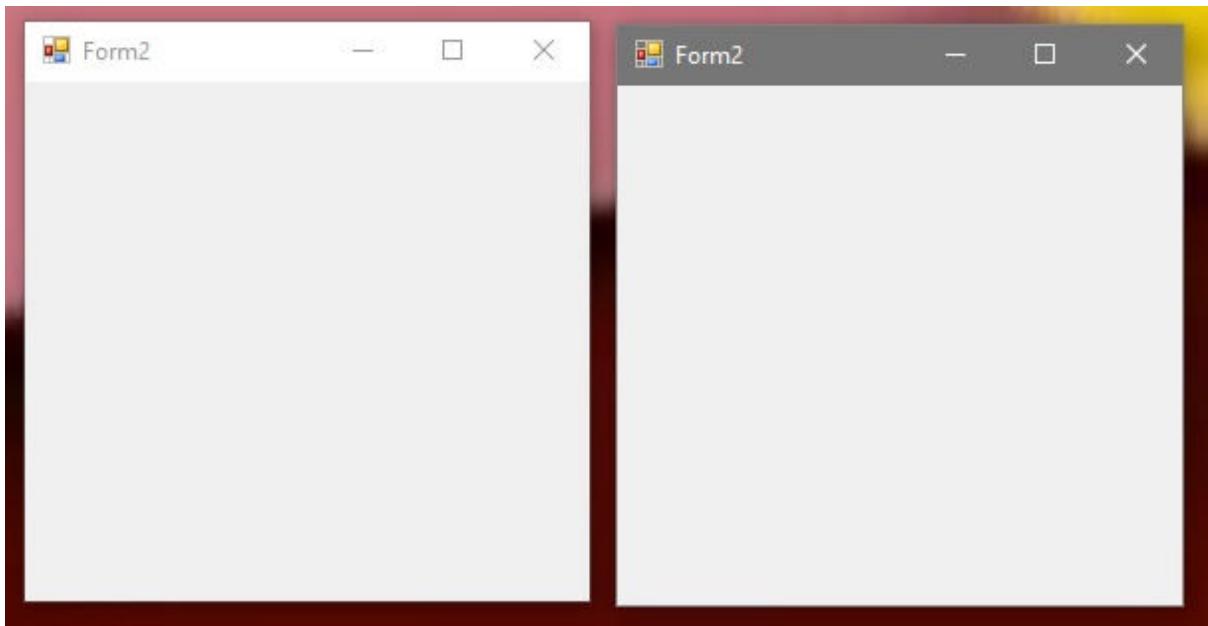




Same output as above - successful in being tested.

HomeCall being ran on Form1_Load



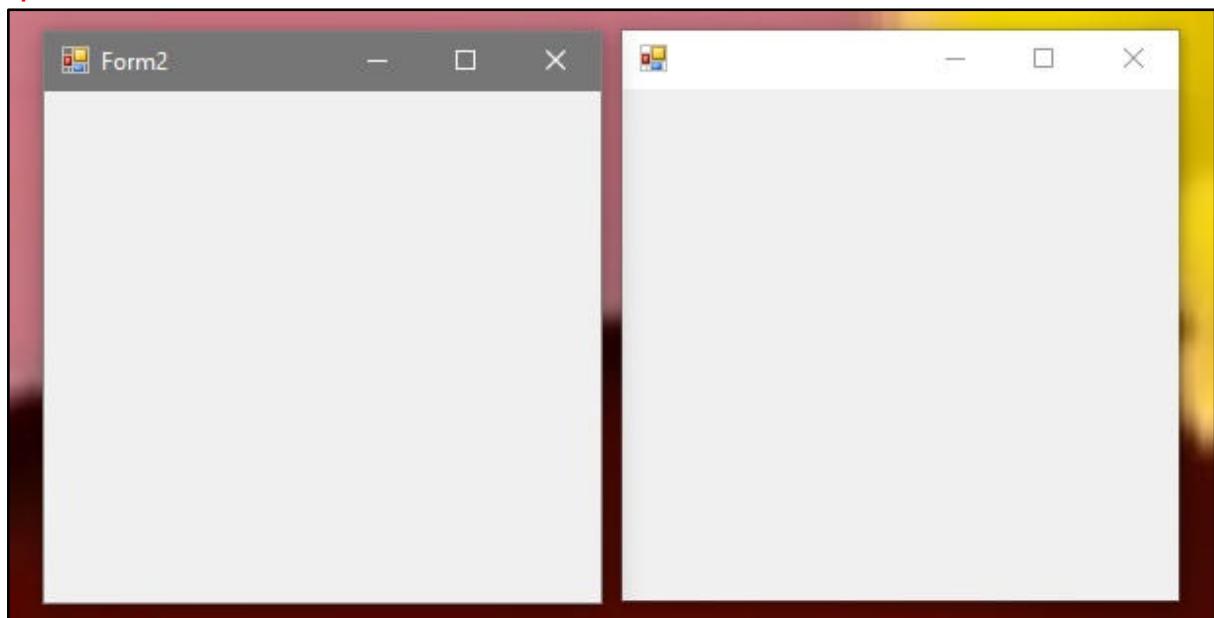


This produces an instant anomalous output, as it displays both the home form and the setup form once the program is ran - indicating error as only one form should be shown at a time. Also, once setup is submitted, it displays the bottom image which is a very strange error, displaying two clones of the home form. This is likely because the command is being ran to show Form2 upon startup, but there is a problem with Form1 being hidden - perhaps it does not yet exist for it to be hidden within the 'Form1_Load' method, meaning it tries to hide something that does not exist, before then loading the form which should be hidden. By then skipping this 'hide' function, it allows the HomeCall function to run twice, only with the 'hide' working on the second iteration but the 'Form2.Show' being displayed twice with two Form2 variables being created.

From this testing, it can be deduced that there is a problem with calling on the Form1 to be hidden, with it perhaps not existing within 'Form1_Load' when the HomeCall is called, thus leading me to instead move the decision into an 'if' statement outside of 'Form1_Load' to make a choice of skipping the creation of Form1 altogether, outside of 'Form1_Load'. This if statement is shown:

```
public Form1()
{
    if (System.IO.File.Exists("C:\\Program Files\\HotelSystem\\config.txt"))
    {
        Form2 HomeScreen = new Form2();
        HomeScreen.Show();
    }
    else
    {
        InitializeComponent();
    }
}
```

I tried placing it on the logic that 'if the file exists' it would display the home screen, and bypass initializing the Form1 component altogether, thus not needing to load and hide Form1. However, the results were quite anomalous:



The program created two similar home screens - though with one not being labelled 'Form2' on its upper bar. I deemed this quite strange, as I don't believe the program had an opportunity to loop to show two forms - however it must seem that the Form1 component was being initialized after all.

I tried stepping via the IDE to check if it iterates through HomeCall() at all, though it ran just as expected yet called on HomeCall() only once, yet two forms appeared once again.

I tried multiple methods in place of 'Hide()', as I instead tried to 'Close()' the form - though this simply ended the program hence not being a viable solution.

The only combination of methods I found to be viable was to minimize and hide the window on the user's computer, completely hiding the form whilst keeping it in main memory (a major disadvantage to this solution).

```
private void HomeCall()
{
    this.WindowState = FormWindowState.Minimized;
    this.ShowInTaskbar = false;
    Form2 HomeScreen = new Form2();
    HomeScreen.Show();
    Hide();
}
```

Using 'this.' and then editing the Windowstate to minimize, and then disable its ability to show itself in the taskbar did give the output which I originally hoped for, though I felt it was not the most solution.

A second, much more efficient solution which I came to use came from a better of understand of the 'this.' function:

```
private void HomeCall()
{
    this.Hide();
    Form2 HomeScreen = new Form2();
    HomeScreen.ShowDialog();
    this.Close();
}
```

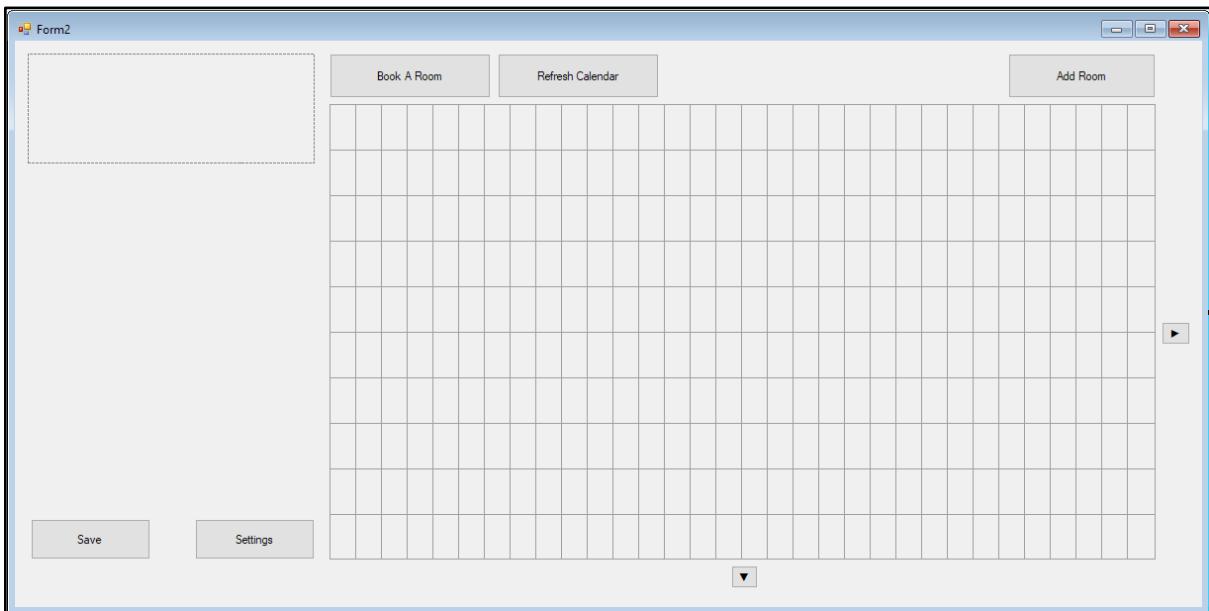
This came from an understanding that form1 must be kept open until it has its replacement, as it is the main 'anchor' form which can call on other forms, therefore the program cannot function if it is closed prematurely: though it can still function whilst hidden.

From this, I hid the form and then created the Homescreen variable and showing itself (now allowing the replacement of Form1 to now proceed). I could then 'this.Close();' successfully to kill the Initial Setup window process.

This is an improvement to the previous solution, as it will save space in the main memory when executing the program by keeping it clear of running the initial setup form, when it is not going to be used, additionally being a quicker program to maintain for user's with little computer memory.

Despite logical thinking, I could not just use 'this.Close' by itself, as it (for some anomalous reason) did not close the setup page when running the program for the first time, though hiding the setup and *then* closing it worked for all cases of the program, so I kept this combination.

After knowing that the home screen would appear to the user at the correct times, I began working on the UI of the home screen. I used the same methods as the initial setup page by using tools from the Visual Studio toolbox.



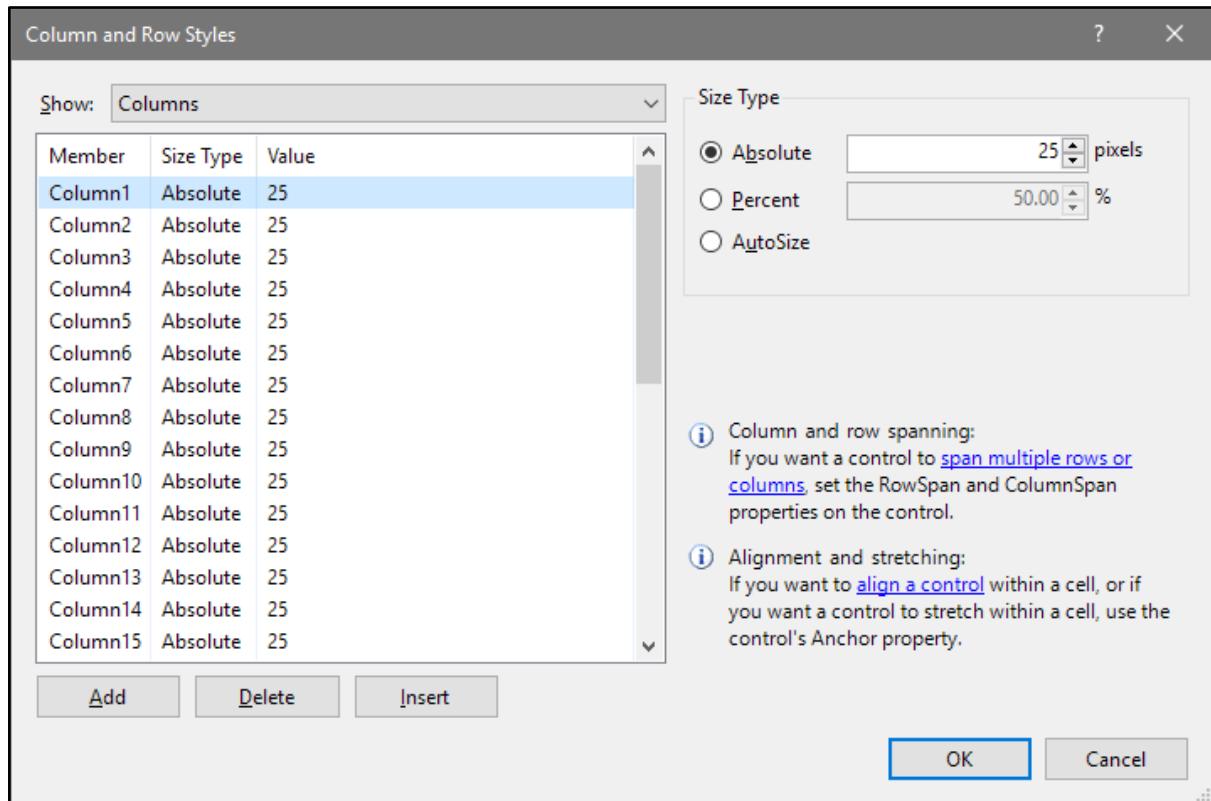
This was the foundational layout in terms of the original design, though quite noticeably it is lacking the table on the left which is supposed to display the day's expected visitors. I chose to currently leave this table out of the form's design for the moment, as I feel as though it is a volatile feature which may change through the development of the calendar table. I felt as though the table would be a complex addition to the form, hence I will first learn how to operate the table with the calendar before deciding to implement a table for the daily visitors.

In terms of form description, I have placed a picture box in the top left of the form to hold the picture that the user originally uploaded in the initial setup - followed by two labels below the picture box (shown blank in the image above) to withhold the hotel's name and telephone number - just like the design entailed.

Surrounding the outline of the form are the buttons, currently without any image designs attributed with them as originally planned - though I plan to introduce their functionality before finishing on their design due to the order of importance.

As well, the calendar has the arrow buttons which will be used to navigate either by month (the table specifically has 32 columns to accommodate every date in a month, as well as the name of a room.) or

by room, by being able to scroll every room in the hotel in blocks of 10. These buttons will decide which section of the 2D array will be displayed in the calendar.



The table shown in the form needed to be precise in how it was laid out, as I need it to be able to have 32 columns but with a suitable spacing within each one, to ensure they could display a reasonable amount of recognizable information. Shown above is how I managed the table via the VS toolbox, as the table's options allowed me to enter specific rows and columns at certain sizes - and I chose to use absolute 25 pixel measurements per column, to ensure that the table does not autosize and distort depending on what is inputted. This was the size I felt was the border between too large and visually inefficient, and to concise and unable to communicate information. I feel that at this size, I will be able to store the initials of the visitor - prompting the user to hover over the specific column to gather more information.

Alongside the sizing, I needed to insert a 'control' into each table's space - a control meaning the actual function of the field. Therefore, I needed to insert a label control into every space within the table. Fortunately, the table sizes the labels automatically to the size of the box they are contained in, so they were simple to organize within the table. A problem I thought would occur from having so many labels would be the state of

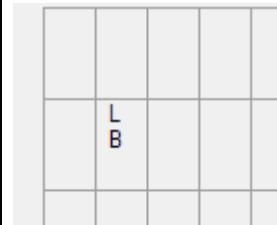
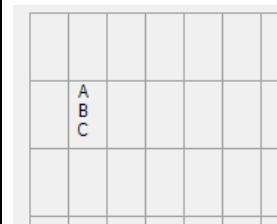
organizing them as a 2D array to call on the appropriately, though the table successfully labelled each label to their position in the table via 2D coordinates - which will be extremely useful later on (and hopefully functional).

I did a black box test on the inputting of information into this table, to see how the label controls would handle different length strings of information.

```
private void Form2_Load(object sender, EventArgs e)
{
    Label label = (Label)Calendar.GetControlFromPosition(1, 1);
}
```

The first step was working out the method in which I would communicate and select a certain label in the table using its coordinates. Using ‘GetControlFromPosition(x,y)’ returns the specific control which is located in the certain (x,y) cell - I then had to convert this Control data type to the Label data type by using (Label) before calling the control. I then gave the variable ‘label’ the link to this control, allowing me to change this cell.

I then tested inputting a range of values into these labels to see how they reacted in the table:

Input	Expected Output	Real Output	Evaluation
Normal Value “LB” initials in (1,1)	It stores the initials in a clear manner, in the correct cell (1,1)		It successfully displayed the initials in a clear pattern, allowing them to be identified by staff more efficiently
Erroneous data “ABCDEFGHIOL ELE”	The cell tries to bare the input, but does not change size to accommodate		It successfully tried to store the letters, and did not resize when the characters broke the limit. From this test, I can see that

			validation will need to occur in the labels to ensure only 3 letters are stored.
No data “”	The cell remains blank		This is successful, meaning that the cells which do not have a booking can be left blank.
‘Label.Colour’ set to red.	The specific cell turns red		It successfully displayed the colour, just not filling the box like I had anticipated. From this, I learnt that the labels cannot be filled fully with colour - just the text. Hence I will need to replace the labels with the text box control to allow this functionality.

From these results, I felt as though the ergonomics of the project would benefit if I chose to display only a week's worth of visits, not an entire month's worth. This would allow the program to display more information, and feel less visually compact to cram a minimal amount of information into each cell. Hence, upon revising the table, I changed the value of columns to 8 from 32, with one column for the rooms and then a column for every day of the week. The rows however, remained the same.

After replacing the table with a new table populated with text boxes instead of labels, I did the final test using the new updated code:

```
private void Form2_Load(object sender, EventArgs e)
{
    RichTextBox Rich = (RichTextBox)Calendar.GetControlFromPosition(1, 1);
    Rich.BackColor = Color.Red;
}
```

This is identical to the label code, with the difference being that the label data type has now changed to a RichTextBox data type.

The final test of colour change was retested below:

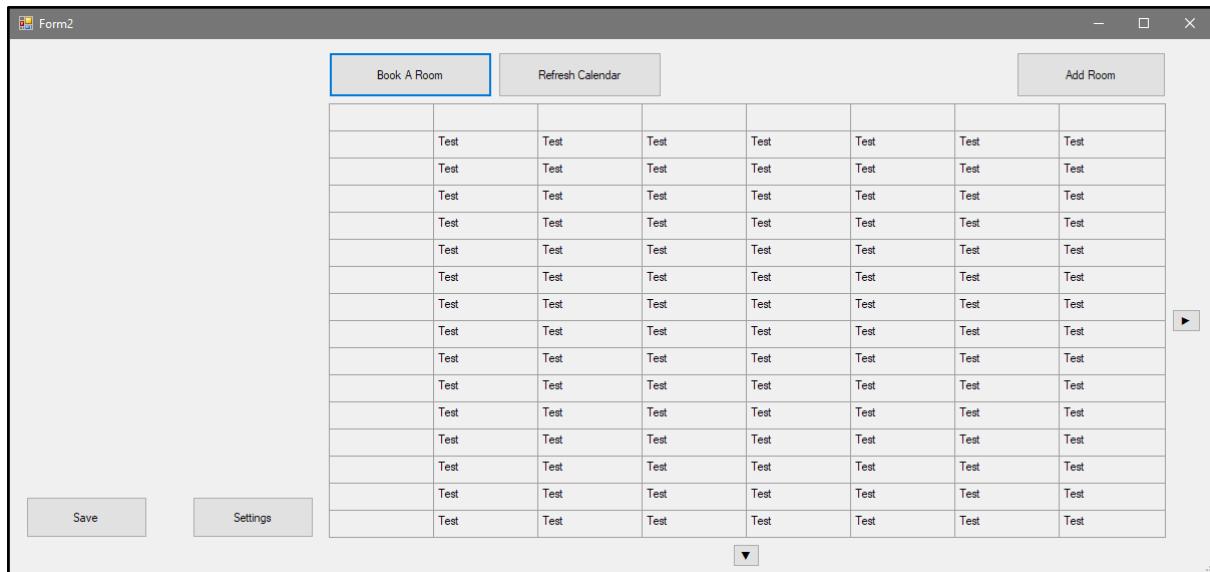
Input	Expected Output	Real Output	Evaluation
'Label.Colour' set to red.	The specific cell turns red		The entirety of the cell is covered with the colour - following the design a lot better.

To prove the table's functionality, I then created a quick test of how I would iterate through the table later on when it came to writing to 'update()' function, to update the table with corresponding bookings.

Using two for loops (for each axis) this is what I came up with:

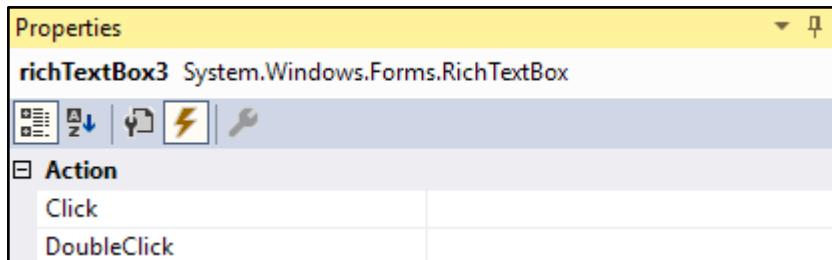
```
private void Form2_Load(object sender, EventArgs e)
{
    RichTextBox Rich;
    for (int x = 1; x < 8; x++)
    {
        for (int y = 1; y < 16; y++)
        {
            Rich = (RichTextBox)Calendar.GetControlFromPosition(x,y);
            Rich.Text = "Test";
        }
    }
}
```

Which upon running the code resulted in this:



This iterative code specifically leaving out the first row and column is beneficial, as it would allow me to treat these first lines as separate 1D arrays to organize rooms and dates much easier, instead of dealing with them all as one single 2D array. This is an example of problem decomposition, by splitting the calendar into different components I will need to work on separately - allowing a more organized development. Whilst this quick iteration code shows I can interact with every cell quite painlessly, it does not show how I will be able to differentiate each cell via their column or row. My first thoughts on how I could do this is that I will need to update the rooms and times imperatively, before then storing the information from the database as a temporary variable, and extracting the important dates and rooms. By then comparing each of the cells to this extract information upon every iteration, I can then highlight which cells need changing as I refresh the calendar - where the effective changes can then occur to the physical calendar. By using abstraction to take the important details from the database and storing them in a temporary class in the C# code upon every refresh of the calendar, this will eliminate the slowing nature of repeatedly accessing the database upon refreshing, and instead just accessing a large amount of data at one instance.

For the user to then interact with each cell, I found that there is a method alongside the event handler for “richTextBox_Click” which is a function ran if the user clicks, or double clicks a certain text box - allowing me to code a specific form to load with the booking details on a specific cell in the calendar.



These are the events found in the richTextBox variable properties, which I can then assign to any named method in the code.

I could not advance onto the navigation of rooms via the calendar (up and down arrows) as I had not yet got a method to store rooms, therefore I would not be able to develop this function.

However, I started to look at ways I could change the dates at the top of the calendar to advance or retrogress by a week upon clicking their respective left or right buttons.

I first had the idea of storing an 'anchor' variable which would be the current date - and then iterate through each column by adding one day to this anchor and then displaying them in their column - and if this anchor changed by a week, this code would still function correctly. This would mean that the first day of the column is always the current date which was one of my plans for easier navigation for the completion of the success criteria. This anchor would also need to be of a 'date' data type, which is usable in C#, and adding days to this data type may be difficult if it rejects arithmetic operators.

```
private void ResetDates(int preset)
{
    displace = displace + (preset);
    DateTime DAnchor = DateTime.Now.AddDays(7*displace);

    for (int i = 1; i < 8; i++)
    {
        ((RichTextBox)Calendar.GetControlFromPosition(i, 0)).Text = DAnchor.AddDays(i-1).ToString("dd.MM.yyyy");
    }
}

private void button1_Click(object sender, EventArgs e)
    //RIGHT DATE
{
    ResetDates(1);
}

private void button3_Click(object sender, EventArgs e)
    //LEFT DATE
{
    ResetDates(-1);
}
```

The method 'ResetDates()' is the main method used to communicate with the table's dates, and is efficiently designed to only need this one

method to both instantiate the current dates, and also change them via the button presses.

Ignoring the displacement part of the method, I used an iterative for loop to loop through the first row of the table, where the dates will reside. By looping through them, with a starting ‘i’ value of 1 (so that it ignores the top left corner cell, a useless cell), I am able to select the first cell at (1,0), and set the .Text value to the ‘DAnchor’ variable, which is the variable set to the current time earlier in the method.

By using the (AddDays(i-1)) method, this allows me to add days to a certain datetime variable, meaning on the first iteration it would take the original date and have ‘i’ set to a value of 1, meaning it would set the .Text in (1,0) to DAnchor if you added 0 days (hence i-1). I then needed to use .ToString to convert the datetime data type into a string data type, in a specific format : ‘dd.MM.dddd’ was what I researched to return the date’s day number, followed by month number, followed by the name of the day represented by 4 d’s.

When i increments and the for loop loops again, it will select the cell next to the previous cell by using the (i,0) coordinates, and as i has incremented to 2, it will now be one cell to the right of the original. As well, it will then add another day to the ‘DAnchor’ date via (AddDays(i-1)). This loop will break when i reaches 8, when it has successfully filled in every text box in the top row of the table: excluding the useless first cell.

In terms of the navigation buttons, I labelled them with a comment of “//RIGHT DATE” and “//LEFT DATE” to intend which direction the dates would be moved (a week ahead to the ‘right’ or a week backwards to the ‘left’). The methods will be called depending on which button is pressed. Depending on which button is pressed, they call the ‘ResetDates’ method but will pass an integer parameter to affect how the dates will display. With this parameter labelled ‘preset’, this will be added to the ‘displace’ variable upon ResetDates being ran, and the displace variable is a public integer variable which is preset to 0 - to begin neutral. Depending on either the ‘left’ or ‘right’ navigation button being pressed, the parameter ‘-1’ or ‘1’ will be passed respectively, and this will affect the displace value by either incrementing the integer by 1, or taking 1 away. This is useful if the user clicks right to navigate multiple weeks

ahead in the calendar, the displace value will keep on adding or taking away 1 without resetting each time.

This displace value is then used to create 'DAnchor'. When DAnchor is created, it has days added depending on the result of ($\text{displace} * 7$), meaning that the anchor date will advance or move backwards weeks (7 days) multiplied by the integer given by displace, which could be any integer value depending on how many times the user clicks the navigation arrows. This DAnchor is then used to calculate the remaining dates in the row, hence giving a notion of efficiency to the date creating method. When the form is loaded, ResetDates is ran when displace is set to 0, meaning the original dates will be displayed every time the form is ran.

Testing is as follows:

Input	Expected Output	Output	Evaluation
Selecting 'right' navigation 3 consecutive times	The table displays the week following exactly three weeks from testing		Successfully displayed the week following 3 weeks after the current date.
Selecting 'left' navigation 3 consecutive times	The table displays the week three weeks prior to the current week		Successfully displayed the week following 3 weeks before the current date.

After completing this component of the calendar, I felt it was necessary to finish the other components of the home screen, as the calendar was at a stage where it was ready to require database access to continue in development.

I chose to work on linking the picture box and other hotel information to the labels on the home screen by reading them from the config file upon loading. I chose to maintain this component of updating the hotel image, name and number within one function - to be called upon once the home screen form is loaded.

```
private void UIUpdate()
{
    using (System.IO.StreamReader reader = new System.IO.StreamReader("C:\\\\Program Files\\\\HotelSystem\\\\config.txt"))
    {
        HotelName.Text = reader.ReadLine();
        HotelNumber.Text = reader.ReadLine();
        CoverPic.ImageLocation = reader.ReadLine();
    }
}
```

This method is shown above. Creating an instance of StreamReader for the default file location of the config file then allows the program to read each individual line of the config via 'reader.ReadLine();'. By then equalling the appropriate control in the form to its specific stored line in the config, I can then quickly set the three controls within the reader function without setting any variables. Using this method of storing and reading this config file means a user will only need to setup these options once, before the program automatically retrieves them every instance of executing the program thereafter. Also, as it can read and skip a blank entry, it means that every setup field remains optional, and will not create an error if the tables are blank, just a blank line entry.

An example of this code running successfully is shown below:

First time running program

Initial Setup

Hotel Contact Number	7647 294010
Hotel Name	Boracay Summer Palace
Background Image	 <input type="button" value="..."/>
Path to HotelSystem Folder	C:\Program Files\HotelSystem <input type="button" value="..."/>

Submit

Running program second time

From this quick test, I do not believe that this small component of uploading this information to this form could be improved on other than graphical issues such as fonts and label size - though no issues in terms of immediate code or display problems.

I feel at this point that the forms and the database are ready to be bridged together, hence I concluded stage 2.

Key Stage 3 - Booking System with Database **(Including Search Function)**

This key stage will involve bridging the gap between the forms and the database, and allowing them to communicate through multiple ways. In terms of roadmapping this stage, the rough plan I imagine I will follow is as follows:

1. Create ‘Add Room’ form and validate the entries, and then pass and store the entries into the database ‘Rooms’ table.
2. Create the bookings form with the visitor entries. Validate all entries, before then being able to store entries within the ‘visitor’ table in the database.
3. Develop a search entry to SQLsearch the Rooms table, and be able to return items to the form in an organized fashion which are available, and match the search criteria. Then be able to select a specific room from those displayed.
4. Correctly store all the information as a record in the booking file.

The main priorities during this key stage will be research development due to my personal inexperience with linking the database to the form, and also testing and validation to ensure the system is structured as it is being developed.

I began by creating the new form which would hold the booking component of the program, and named it ‘BookingForm.CS’.

I then had to devise how the program would need to be communicated with to create an instance of this new form - so I created the method to be ran upon the event of the ‘Book a Room’ button control being clicked, and began experimenting with how I could ‘hide’ the home screen temporarily whilst the booking screen is displayed and interacted with. Unlike the transition from initial setup to home screen, I want to maintain the home screen in memory as the user will inevitably need to return to this form - hence closing and reopening an instance of this form every time it is accessed would be inefficient.

```
private void BookRoom_Click(object sender, EventArgs e)
{
    this.Hide();
    BookingForm BookingForm = new BookingForm();
    BookingForm.ShowDialog();
}
```

Using how I previously changed form, I manipulated the code and reworded it so that it would hide the home screen, and then create an instance of the booking form class, and then hence show this instance. It would not close the home screen however, meaning I could call for ‘Form2.ShowDialog’ when the booking screen is closed, hence moving the user back to the home screen.

However, I foresaw that if a user would be returning to the home screen and could press the booking button once more, this would then perhaps call for another instance of ‘booking’ to be created - perhaps producing an error. I did not have the required staging of development to test this error yet however, though I will be prepared to test if this error is going to occur once I create the link of going back and forth between these forms.

Now I had created system which meant the booking form would open when requested, I then began ensuring that the home screen would appear once the booking screen was finished with.

As there is an event handler which deals with the event of the form closing, I knew I could then have a method called when the form closed - which would involved the opening of original home screen thus finishing this loop. As this would then be a reusable component, I could implement this same piece of code to create this loop between the other menus accessible from the home screen.

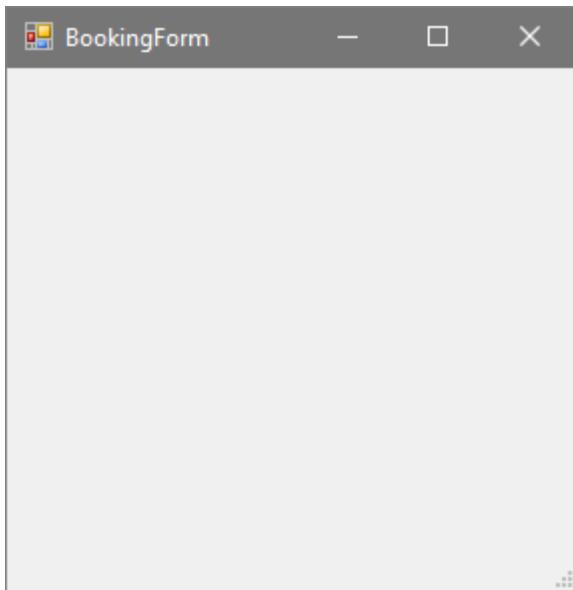
```
private void BookingForm_FormClosed(object sender, FormClosedEventArgs e)
{
    Form2 HomeScreen = Application.OpenForms["Form2"] as Form2;
    HomeScreen.Show();
}
```

This was how I originally planned to reopen the home page after the Booking form had closed, as I used the ‘FormClosedEvent’ argument, to run the method once the form had closed. By then creating a new variable and reinstancing it as the already existing form I created in HomeScreen (By using Application.OpenForms and then selecting the name of a form which should already exist, albeit hidden, it allows me to reinstantiate the variable but in the booking form’s code, as it is currently private to only where I instanced the form.) I also used ‘as Form2’ to recognize the re-instance as the member of the Form2 data type.

By then showing the reinstanced home screen when the form closes, I imagined this would work when I tried opening and closing the booking form.

Before testing, I expected the booking form would open when I click on the button, and then once I closed it, the home screen would reappear and ‘unhide’.

The results are as follows:



The booking form opens when I click the button, and the home screen hides. The test is successful up to this point.

By then closing this form, the entire program ends - without showing the home screen.

The first attempted solution to this problem lied in how I had used the event handler, as I had used the event 'Closed' and not 'Closing', and as it was only running when the form had closed, this meant the program could not execute this method as it is instructed to cease the program's existence once all the forms are hidden or closed. I then changed this event dialog:

```
private void BookingForm_FormClosing(object sender, FormClosingEventArgs e)
{
    Form2 HomeScreen = Application.OpenForms["Form2"] as Form2;
    HomeScreen.Show();
}
```

However, the program reacted the exact same way as it did when I tested it previously, meaning that there was still an inconsistency in the application closing and ignoring the method, and the event handling. I then attempted to see if the program was completely ignorant of the event handling, so I tried overriding the event using 'e.Cancel = true' which basically uses the .Cancel function to deny the user closing the program manually.

```
private void BookingForm_Closing(object sender, FormClosingEventArgs e)
{
    e.Cancel = true;
    Form2 HomeScreen = Application.OpenForms["Form2"] as Form2;
    HomeScreen.Show();
}
```

However upon testing, once again the method was completely ignored.

From this testing result, I began to think that maybe I had to ensure that the method with the handling was being made to exist when the form was loaded, and upon researching this problem it became clear that this was the solution - and this was what I had to run when the form was first loaded:

```
private void BookingForm_Load(object sender, EventArgs e)
{
    this.FormClosing += new FormClosingEventHandler(BookingForm_Closing);
}
```

By executing the above line of code when the form first loads, this ensures that the program will know what to do in the event of the form closing (sort of like an override for the event). As it will now run the method it had previously been ignoring, I tested the program once more.

```
private void BookingForm_Closing(object sender, FormClosingEventArgs e)
{
    Form2 HomeScreen = Application.OpenForms["Form2"] as Form2;
    HomeScreen.Show(); X
    this.Close();
}
```

Exception Unhandled
System.StackOverflowException

This was the result upon closing the booking form, as it threw a StackOverflow exception, this showed that the error likely came from a loop in the code.

Using white box testing logic, I went through the code line by line in this method and worked out that it was showing the home screen, and then closing the booking form - *which then runs the event of closing the form once more*, hence throwing the code into a loop of showing the home screen repeatedly.

o, as the method is only ran if the form is already going to close, then removing the 'this.Close()' would remove this loop from the code, whilst still maintain the functionality - and upon testing it, it worked just as expected.

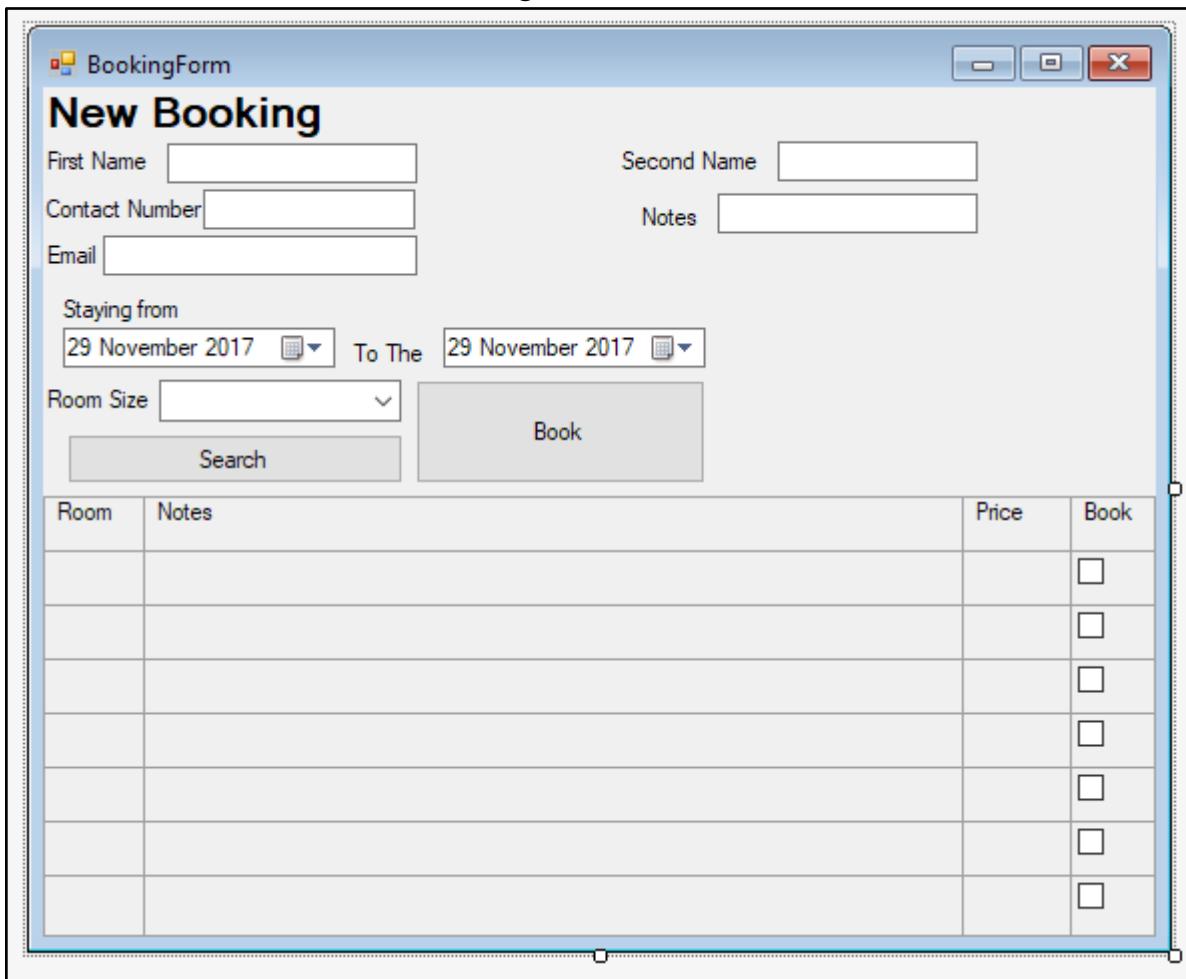
```

private void BookingForm_Closing(object sender, FormClosingEventArgs e)
{
    Form2 HomeScreen = Application.OpenForms["Form2"] as Form2;
    HomeScreen.Show();
}

```

This simple component would now be reused later on for the development of other child forms of the home screen - to return successfully.

I then began adding the basic controls to the design of the booking form, in accordance to the initial design.



This is how the booking form turned out to initially look like whilst populated with the necessary controls. I left it purposefully to have plenty of space, as it will need to display many kinds of validation messages later on in development, which will fill those blank spaces in the form. Also, the 'Book' button will be initially invisible when the form loads, and will only appear when one of the boxes in the 'book' column have been ticked, and this will aid in helping direct a user clearly by running them through the form in its specific order.

I have also used a table control once more to organize the information which will be retrieved from the database, and displaying room information in tabular form would make it much easier to differentiate the rooms when the user selects one.

As shown, I have provided an input for every major field the visitor database will require, as well as the search parameters (availability of rooms via ‘DateTime’ data type input, and the size of the room, which will be selected from a drop down box of a selection of preset room sizes.).

After finishing this design, I began looking at how I would deal with inserting these queries as a record in the Visitor database (I would only deal with the Visitor database for the moment, as that is the only database I could make a record for without there being an opportunity to attach a room to the bookings - as the rooms form does not yet exist). Adjoining the database SQL with this form involved using the functions within the “System.Data” functions included with C#. Instead of needing to navigate down to the ‘data’ directory upon every requirement of the data type needing to be called (shown below), I found that calling this directory at the beginning of the file allowed me to have instant access to this directory - and as I would be accessing this directory a lot during this adjoining, calling the data at the beginning of the file would simplify code.

```
private void button2_Click(object sender, EventArgs e)
{
    System.Data.SqlClient.SqlConnection cmd = new System.Data.SqlClient.SqlConnection();
}
```

Above is how I originally would have to call ‘SqlConnection’ every time I required the function.

```
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;
using System.Data.SqlClient;
```

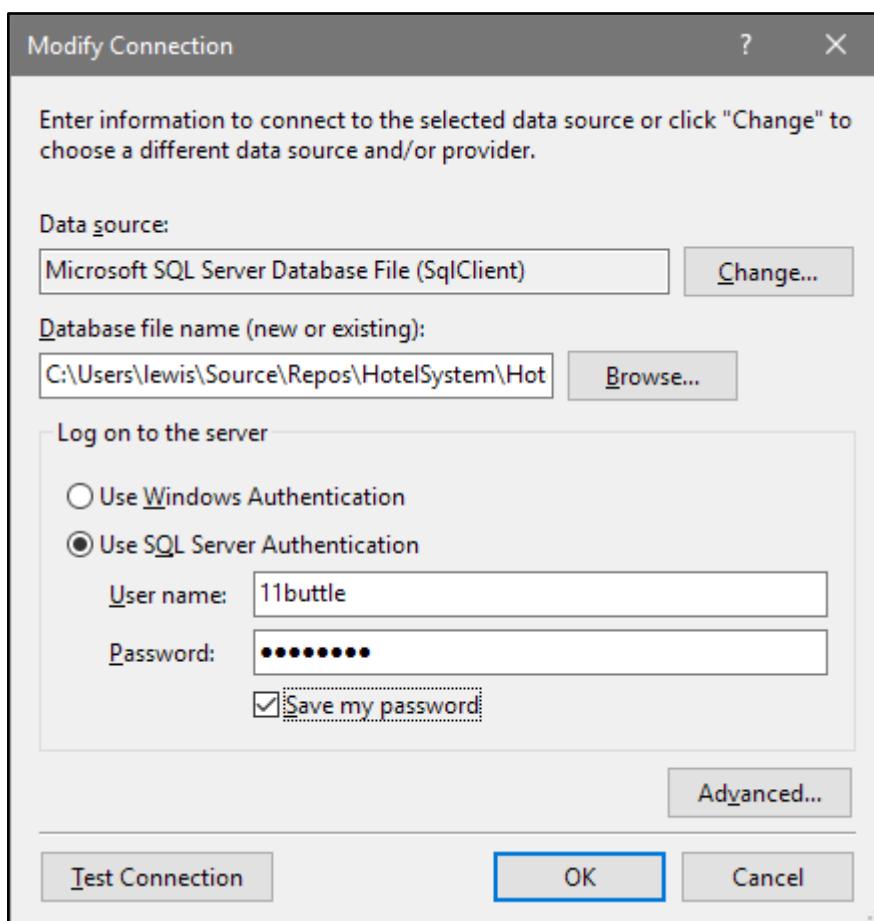
By adding ‘System.Data.SqlClient’ to the using paths at the beginning of the form code, this allows me to simplify the code, shown below.

```
private void button2_Click(object sender, EventArgs e)
{
    SqlConnection cmd = new SqlConnection();
}
```

As the path of .Data.SqlClient is already called, I can now call SqlConnection straight away without navigating to the directory.

```
private void button2_Click(object sender, EventArgs e)
{
    DBVisitorAdd(textBox1.Text, textBox2.Text, textBox3.Text, textBox4.Text, textBox5.Text);
}
private void DBVisitorAdd(string FName, string SName, string Numb, string VEmail, string Notes)
{
}
```

I began the actual function of adding a record to the visitor table by creating a method which takes all the parameters from the form's text boxes (with better simplified names). This method is then called once the "book" control is activated.



Before I could connect to the database from the form, I first had to ensure the database had a connection to the form, especially in terms of authentication.

```
private void DBVisitorAdd(string FName, string SName, string Numb, string VEmail, string Notes)
{
    string ConStr = "Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\Users\lewis\Source\Repos\HotelSystem\HotelSystem\Database1.mdf;Integrated Security=True";
    using (SqlConnection DB = new SqlConnection(ConStr))
    {
    }
}
```

Beginning to use the constructor method to access the database, I required the 'ConnectionString' attributed to the database within the

properties, to allow this connection to occur. Viewing the string itself, I saw that it is highly personalized to the local machine accessing it - hence I will later perhaps be required to retrieve this string instead of just setting it manually - though for the moment I will focus on adding information locally before ensuring it can add records to any computer(this will need to be tested on different local machines).

```
using (SqlConnection DB = new SqlConnection(ConStr))
{
    DB.Open();
    using(SqlCommand Comm = new SqlCommand(
        "INSERT INTO Visitors VALUES(@FName, @SName, @Numb, @VEmail, @Notes)", DB))
    {
        Comm.Parameters.Add(new SqlParameter("FName", FName));
        Comm.Parameters.Add(new SqlParameter("SName", SName));
        Comm.Parameters.Add(new SqlParameter("ContactNo", Numb));
        Comm.Parameters.Add(new SqlParameter("Email", VEmail));
        Comm.Parameters.Add(new SqlParameter("Notes", Notes));
        Comm.ExecuteNonQuery();
    }
}
```

Above is the foundation to the method in which I would access the database, using two ‘using’ statements.

The first ‘using’ statement is given the variable name ‘DB’ for database, as it is the main SQL command to link the database in its entirety to the code - hence using ‘DB.Open()’ after then connects the database using the connection string, passed through “ConStr”. Opening the database then allows me to access it and decide the code’s specific purpose in the database. From my understanding, I then called another constructor called ‘Comm’ (in relation to communication) via ‘using’ which is a form of SqlCommand, which allows me to then interact with the DB.

I had to use SQL language and input it in a string data, as C# would not recognize it and is simply passing it on to be ran in the SQL code. The code in question is “INSERT INTO Visitors VALUES” followed by the specific names of the tables I will be inserting into. By listing the specific columns I would be inserting into, I then had to add the parameters - which is the name for the values being added to the form.

From research, I found that using parameters is only one method of inputting into the DB - though I chose this method as it allows and includes security to avoid SQL injection from outside sources.

As shown, SqlCommand takes two parameters - the code being inputted into SQL and also the connection - which I made sure to use the previously instantiated “DB” to access this.

I then individually went through each variable being added to the form using ‘SqlParameter’, and added these values as parameters. The command is then executed (similar to ‘break’), sending all of the necessary SQL code as it now as both the information being sent, and the location.

With this foundational writing function set in place (and with little expectancy for it to work), I began the upward spiral of testing - followed by immediate remedial action to ensure this function worked correctly and efficiently.

I began by attempting to input some values to the database through the form (input shown below)

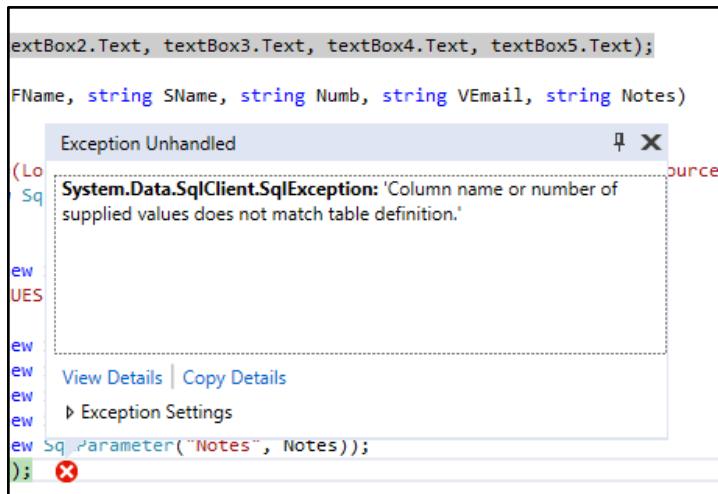
The screenshot shows a Windows application window titled "BookingForm". Inside, there's a form titled "New Booking". The form contains the following fields:

- First Name: Lewis
- Second Name: Buttle
- Contact Number: 012707767
- Email: Lewis.Buttle1999@outlook.com
- Staying from: 04 December 2017
- To The: 04 December 2017
- Room Size: (dropdown menu)
- Search: (button)
- Book: (button)

Below the form is a table with the following structure:

Room	Notes	Price	Book
			<input type="checkbox"/>

By using the form and my initial code, the outcome the function would successfully output would be to have no errors, and to then store the information I inputted as data into an instance of the database. The result of pressing 'Book' is shown below.

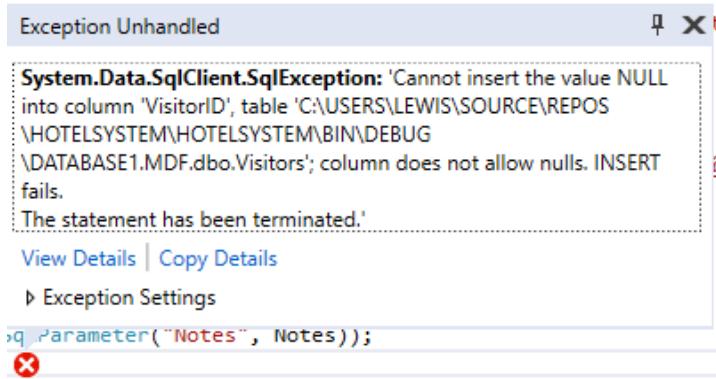


The test unfortunately failed, displaying an error. I learnt from this test that there was an error in how I had communicated the ‘supplied values’, which I then thought may have had to do with the fact that I had given the incorrect names to the SQL string - those which did not match the actual database column names.

Remedial action from this test included that I changed these names to the actual column names, so that it could now be understood by the program to write to the correct columns. I also noticed from this test, that I had forgotten to include the specific columns following ‘INSERT INTO VISITORS’, as just communicating visitors without listing the columns I needed to access would not allow me to input into these columns.

By bracketing the columns I would be accessing (Fname, Sname, and so on), this directly correlates with how I organize the VALUES bracketing thereafter - using parameters. For example, the first value being stored is FName and this would be accessed directly from the first value of the VALUES bracket ‘@FName’, which directs to accessing the parameter to retrieve this actual value, to then feed back into the VISITORS table.

```
DB.Open();
using(SqlCommand Comm = new SqlCommand(
"INSERT INTO Visitors (Fname, SName, ContactNo, Email, Notes) VALUES (@FName, @SName, @ContactNo, @Email, @Notes)", DB))
{
    Comm.Parameters.Add(new SqlParameter("FName", FName));
    Comm.Parameters.Add(new SqlParameter("SName", SName));
    Comm.Parameters.Add(new SqlParameter("ContactNo", Numb));
    Comm.Parameters.Add(new SqlParameter("Email", VEmail));
    Comm.Parameters.Add(new SqlParameter("Notes", Notes));
}
Comm.ExecuteNonQuery();
```



After retesting this new code, the above error was displayed - which showed that I was progressive in terms of development as it was a different error to the previous error.

From seeing the error, I found it strange that the VisitorID was being given a null value - instead of incrementing by 1 such that a primary key should, and this is where I felt the error had primarily been a subject to.

Table Designer	
Collation	
+ Computed Column Specification	
+ Full Text Specification	False
- Identity Specification	True
(Is Identity)	True
Identity Increment	1
Identity Seed	1

From viewing the properties of the row for the VisitorID, I noticed that I had to enable 'Identity Specification' and ensure that it was incrementing by 1, and had a seed of 1 (meaning it increased by 1 each record addition, and counting began at 1).

```
CREATE TABLE [dbo].[Visitors] (
    [VisitorID] INT           NOT NULL IDENTITY,
```

From updating this fact, the SQL for the database changed to register VisitorID as an 'Identity' value, showing it was now a fully clarified primary key.

After updating the database, I felt it was suitable to then run another test to see if the code encountered any more errors. The above input was given, and once 'Book' was activated - the program was successful in not producing an error. I then checked the actual data table for the visitor table, and it clearly showed a successful output to the testing - all input values successfully added.

	VisitorID	FName	SName	ContactNo	Email	Notes
▶	1	LEwis	Buttle	010110	Test	Test
*	NULL	NULL	NULL	NULL	NULL	NULL

Though this testing was primarily just to see if the code could input any values, I will do deeper testing into incrementation and validation later on as I structure this foundation more.

The next -smaller- stage I felt I would quickly address before it became a larger problem, was the matter of directory relativity for retrieving the database connection string - as the program could only currently retrieve the database locally and bespokely from my own computer - but would fail as the connection string would be different for every computer system, hence I would need to find out how to have the code find the

connection string itself - to ensure portability of using the program on different computers.

```
string ConStr = "Data Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename=" + LinkString() + ";Integrated Security=True";
using (SqlConnection DB = new SqlConnection(ConStr))
{
```

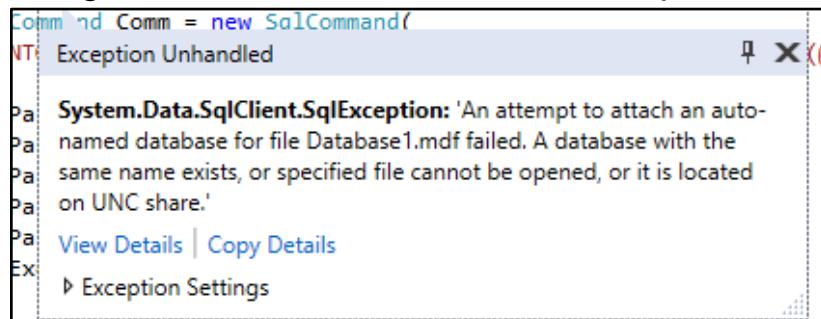
I started solving this by using a method of abstraction, to find the exact segment of the string which would be changing between computer systems - the only part of the string relevant to the problem. I worked out that the only piece of the string needing to be changed is the path to the .mdf file of the database - as this is what I am opening via the SQL command. As this database is certain to be a LocalDB (as I am not creating a solution to harness online capabilities, as I feel it is not appropriate for the effectiveness of the solution), the path to load LocalDB is the same for every computer running the program.

Using string manipulation, I placed the string return from the method ‘LinkString()’ to be placed within the interchanging .mdf path segment - and the code behind this function is shown below (first iteration).

```
private string LinkString()
{
    string path = (System.IO.Path.GetDirectoryName(System.Reflection.Assembly.GetExecutingAssembly().Location));
    return (path + "\\Database1.mdf");
}
```

To retrieve this path, I first felt it would be quite simple as long as it was certain that the mdf was to be kept relative to the execution file of the program, which is a feature of visual studio’s project manager. Hence, I used a command to find the directory name (GetDirectoryName) of the executing assembly (.exe file), and then return the location of this directory via string format (.Location).

I would still need to add the ‘\\Database1.mdf’ part to the string, as this would not be included in the .location return, hence when I return the string I added “\\Database1.mdf” to the path. I then tested this function.



The result upon running the code for the first time, I was met with an error. This error communicated that the code had failed to reach the directory and destination of the database1.mdf file - meaning there was

an error in `LinkString()`. As I had abstracted the necessary segment of directory, this now limits the error to `LinkString()` (Most likely).

```
private string LinkString()
{
    string path = (AppDomain.CurrentDomain.BaseDirectory);
    return ("Data Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename=" + path + "Database1.mdf;Integrated Security=True");
}
```

After researching the problem, I found that there was an easier method of finding the location of the base directory using just one function of code (`AppDomain.CurrentDomain.BaseDirectory`). Whilst this does improve the code's efficiency, it still does not eliminate the error provided by the code.

The solution to the error was quite minute, and it was the fact that the string of the directory location being returned in fact returned with a '\\' at the end of the string line, hence when I added '\\Database1.mdf' to the end, this then totalled to 4 backslashes - making it not understandable to the SQL code. By removing the double backslash when adding the extra string value, this then ran the code without error. I then tested this function once more - hopefully making it to the point of storing data in the database.

Above is the input I tested into the form, upon pressing the 'Book' control, I examined the database to check if the data input was successful - which I had to manually access. Initially I checked the data assigned with the database which I was editing - however this gave me a totally blank table upon each inspection. As I knew the code was successful without errors, I knew the data was being input into some table.

The solution to this was that when I test the program, it runs a copy of the program in the a 'bin' directory within the program files, hence there was a copy or 'instance' of the database.mdf file in this bin directory.

Upon opening this mdf file in the bin directory, it showed the below data:

	VisitorID	FName	SName	ContactNo	Email	Notes
▶	1	Test	Test	Test123	Test	Test
	2	TEsie	Test	Twst	Test	Test
*	NULL	NULL	NULL	NULL	NULL	NULL

As the table had successfully stored the input data from the form, and to the correct directory (the same directory as the .exe file), this testing has finally reached fruition. By fully testing this function, I can reuse this module for inputting information into the database when it is next needed during development, as it is a reusable component.

Reflecting at this point shows that I now have only two other ways I will need to interact with the database - reading the database and searching the database, but by completing the data deposition code this has given me beginning knowledge on how I would tackle both those tasks.

The next stage was to begin validation, where I would be using the same method of validation as I did during the initial setup. I would call a validation function, passing all the data that would need validating into this function. It would then return a bool data - false if the data is not usable, and true if it is usable. Calling upon this validation and using an 'if' statement when the book button is pressed would then prevent data being inputted to the database unless it is correct - hence allowing me to prevent any errors for being produced by the database (especially for the 'allow null' fields, I will need to validate if some fields are blank.). By running this bool through every individual validation, and changing it if it is false - it will then give an indicator to the status of the validation of the code.

Also, displaying a message on the form if the validation fails will also be necessary.

I cannot just return false upon a bad validation, as this would not continue to validate the rest of the inputs, thus only one return message would be displayed instead of communicating all of the validation errors to the user.

```
private bool Validation(string FName, string SName, string Numb, string VEmail, string Notes)
{
    label30.Text = "";
    Regex AlphNum = new Regex("^[a-zA-Z0-9]*$");
    bool end = true;
    if (FName.Length < 20 && AlphNum.IsMatch(FName)) {
    }

    else
    {
        label35.Text = "";
    }
    return false;
}
```

The first validation was to validate the first name input, where I would need it to display a message and maintain the bool as true if the first name is:

- Less than 20 characters
- Consists of only alphanumeric characters.
- Not blank. (Null).

The image above shows the initial preparation of this as I set up the 'if' statements for the first two validations.

Label 35 is the label I would use to house all the outputs of the failed validations, as it would be too tight on space if I attempted to put individual validations for each field and place them next to the field - especially when there can be multiple outputs for one field. Therefore I set up a large label and ensured the default colour was red (the same shade used in the initial setup failed validation outputs.), and at the beginning of the validation method, the label is set to "" (null) to ensure that every time the book button is pressed, it does not simply just keep adding to this label with errors - it constantly resets to show the progress the user is making to having a correct set of inputs for the database to handle.

After resetting the label, I began instantiating the variables which would be useful to validation - specifically the bool value to be returned (named 'end').

I also had to organize a Regex data type variable, which is a type of data which stores a range of characters - mainly for the comparison of strings. I set the new regex as [a-z A-Z 0-9] which means the regex 'allows' all the alphanumeric characters, both capital and lowercase would be allowed when compared with the regex. By using [*\$], this disallows all 'special' characters.

With these variables in place, I then initially created an if statement and tried placing all the comparisons into one if statement - though at this stage I realized that this would disable the ability to have specific invalidation text depending on the error - as all the errors are being grouped into one decision. Hence, I had to improve the code by splitting each decision into different if statements (shown below). This would make the code much more disorganized, so I must add comments splitting the if statements into their specific control field (eg, putting two comment splits '//' between the if statements which deal with the first name.).

```

    private bool Validation(string FName, string SName, string Numb, string VEmail, string Notes)
    {
        label35.Text = "";
        Regex AlphNum = new Regex("^[a-zA-Z0-9]*$");
        bool end = true;
        if (AlphNum.IsMatch(FName)==false)
        {
            end = false;
            label35.Text += "Must contain alphanumeric characters. ";
        }
        if (FName.Length > 20)
        {
            end = false;
            label35.Text += "Must contain less than 20 characters. ";
        }
        return false;
    }
}

```

The version above of the method is the structure I would maintain, as individual if statements.

The first if statement deals with the alphanumeric nature of the first name field. It uses the comparison of:

AlphNum[The Regex Value].IsMatch(FName)[A regex function comparing the unicode of the regex with the chars in FName] == false
[This if statement will be performed if the values do NOT match].

This is a switch from the original code, as this will run the if statement if the validation is *not* met, which helps as I can then add to the label if the function is ran - and changing the bool to false to return this boolean at the end. (I set the function to return false at the end as a placeholder, to ensure the database isn't being filled with unnecessary test data).

As shown, within each if statement it will contain both this bool being changed to false, and then Label35.Text +=[This means the text will not

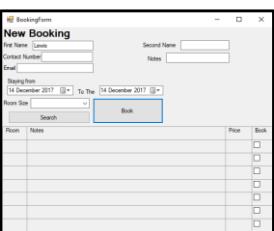
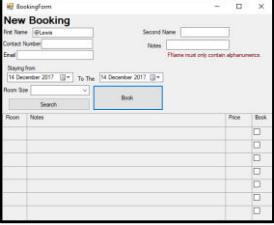
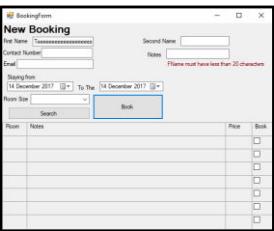
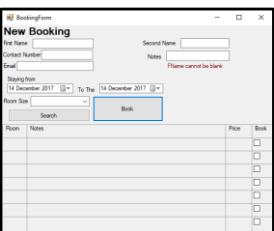
be overwritten, it will simply be added to the text which is already there.], followed by the specific line of string relating to the error.

The next if statement deals with the length of the name, which is a simple comparison of if the ‘FName.Length’ is bigger than 20, which will then add the error of FName length to the label.

```
if (FName == "")  
{  
    end = false;  
  
    label35.Text += "FName cannot be blank";  
}
```

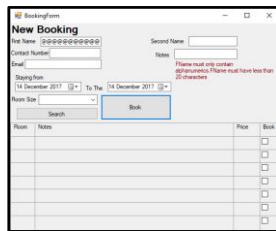
The final statement is to ensure FName is not blank, shown above. It simply compares the string value stored in FName, and compares it to a blank string (“”).

After writing this code, I then tested the validation of each statement.

Input	Expected Output	Output	Evaluation
“Lewis”	Blank Label35		Successfully inputted into form. No further action.
“@Lewis”	‘Alphanumeric s’ error displayed		Error successfully displayed.
“Teeeeeeeeeeeeeeeeest”	‘String length’ error displayed		Error successfully displayed.
“”	‘Null’ error displayed		Error successfully displayed.

"@ @ @ @ @ @ @ @
@ @ @ @ @ @ @ @
@ @ @ @ @ @ @ @
@"

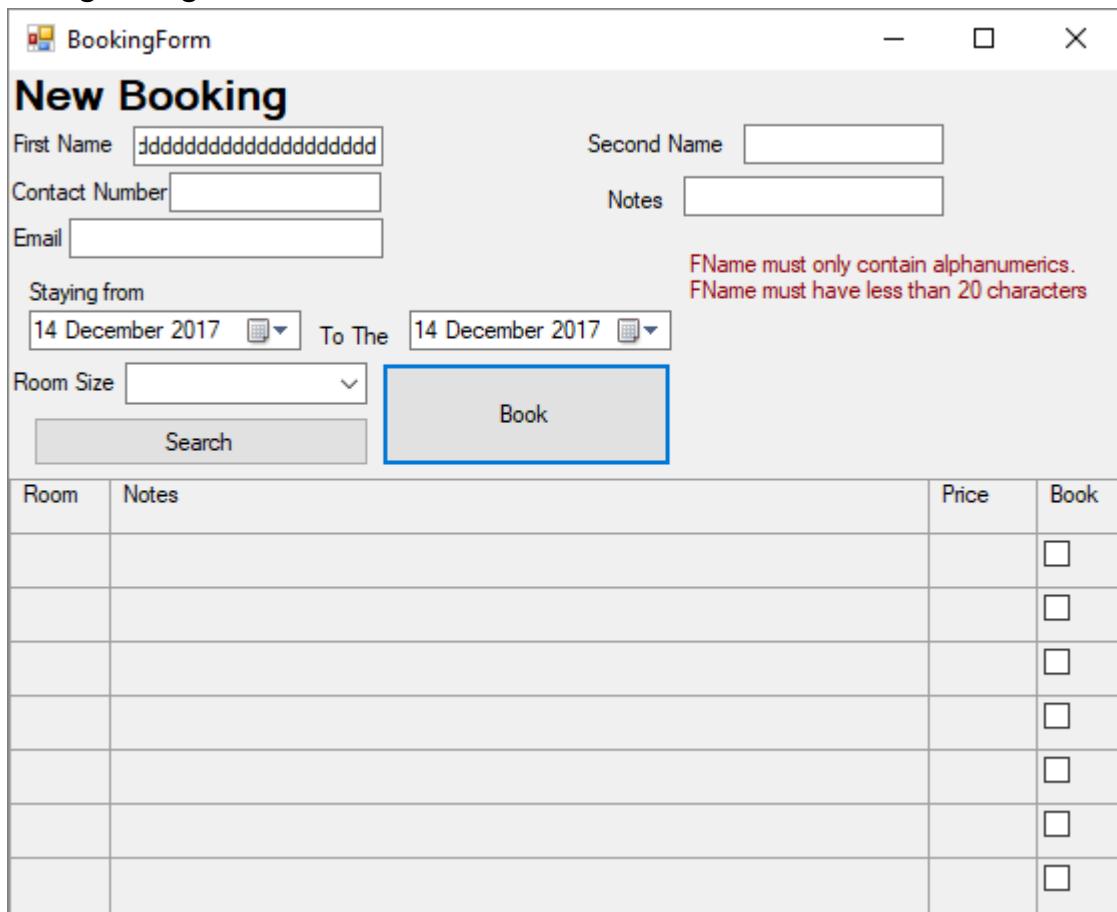
Both
Alphanumerics
and Length
errors
displayed.



Errors
successfully
displayed,
though
inefficiently
displayed in
form.

```
end = false;  
label35.Text += "\r\nPNumber must be between 6 and 14 characters"
```

To then fix the slight output error with the last test, I had to add a slight bit of string manipulation at the beginning of each string input to ensure it began a new line in the label upon being added to the label. By using '\r\n' at the beginning of a string, this requests this new line upon each string being written:



This now solves the problem of formatting - but now means that I must restrict the strings to fit in their appropriate lines to maintain this

formatting - perhaps using reasonable mnemonics to communicate to the user what the error is to ensure this formatting remains correct. The one slight problem from using '\r\n' is that the label storing these errors will always skip the first line of the label - wasting valuable space to display these errors. To maximize space, I slightly oversized the 'roof' of the label over the entire form, to ensure this first line being skipped in unnoticeable.

The screenshot shows a Windows application window titled "BookingForm" with a title bar and standard window controls. The main content is a dialog box titled "New Booking". Inside the dialog, there are several input fields: "First Name" and "Second Name" (both in red), "Contact Number" and "Notes" (both in red), and an "Email" field. Below these are date selection controls labeled "Staying from" with "22 December 2017" and "To The" followed by another date selector. A dropdown menu for "Room Size" is open. At the bottom left is a "Search" button, and at the bottom right is a large "Book" button. Below the dialog is a table with four columns: "Room", "Notes", "Price", and "Book". The "Notes" column contains several rows of text, and the "Book" column contains eight empty checkboxes.

Above is the reformatted form to house the validation errors more securely - the disproportionate label height and with the form being slightly wider.

Implementing this similar validation into both the first and second name field, I moved onto the contact number validation.

The criteria for a passing validation number is if the field is:

- Entirely null (As the number will be optional to input)
- Between 6 and 14 characters long
- Only consistent of numeric characters.

As two of these criteria deal with the length of the string, I grouped them both into a single if statement:

```

if ((Numb.Length>0&&Numb.Length<6) || Numb.Length>15)
{
    end = false;
    label135.Text += "\r\nnPNumber must be between 6 and 14 characters";
}

```

The if statement comparison here makes use of both AND and OR boolean expressions to filter out the specific length which will trigger the *invalid* sequence - using ‘Numb.Length’ (the length of the string in the contact number field).

In a pseudo code format, I have split the ‘if’ sequence to break by **two** criteria, not just one criteria as I have previously done. Using the ‘||’ parameter to represent ‘OR’ in more pseudo terms, I have made the string’s length to be rendered not passable by the code if ‘**Numb.Length>0&&Numb.Length<6**’ (if the length is bigger than 0, AND if the length is smaller than 6). With x representing a positive, *invalid* length of the string, this first part ensures the statement will run if “**1<x<6**” by setting two boundaries.

The second part of the if statement after OR ensures the output is kept discrete between 6 and 15 characters - hence ‘**Numb.Length>15**’ (if the length is bigger than 15).

Hence if either of these criteria are met, it will run the ‘fail validation’ code.

By using these two comparison criterias with an OR parameter, it allowed me to single out two discrete sections of integers that were not necessarily consecutive (0 and 6-15). Hence using two criterias in one if statement may have been the most effective solution in this situation.

The ‘only numerical’ validation required the use of the regex data type once again, though slightly changed to only represent numerical characters, not alphanumeric characters.

Hence, I instantiated a new regex variable called ‘Num’:

```

Regex AlphNum = new Regex("^[a-zA-Z0-9]*$");
Regex Num = new Regex("^[0-9]*$");

```

Comparing it to ‘AlphNum’, it is quite similar just with the ranges of ‘a-z’ and ‘A-Z’ removed, as these would be the alphabetical ranges. By removing these ranges and calling a new regex value, it allowed me to compare the string’s numerical consistency in the if statement:

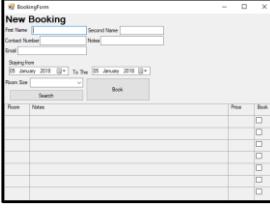
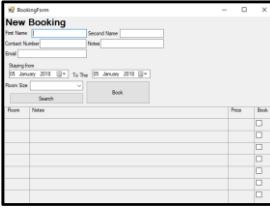
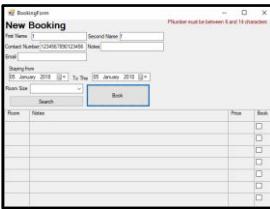
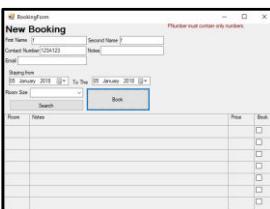
```

if (Num.IsMatch(Numb)==false)
{
    end = false;
    label35.Text += "\r\nnPNumber must contain only numbers.";
}

```

Using “`Num.IsMatch(Numb)==false`”, this means that through the regex class ‘Num’, is the string ‘Numb’ a match to the range of values in this regex? If this comparison is false, the failed validation is then ran, as it shows the string is not comprised of only numbers.

I then tested this validation, with more organized inputs. I feel as though giving a justification to the inputs will make the testing process much more structured, therefore more efficient in providing reflection.

Input	Expected Output	Output	Evaluation
“” - Valid Extreme	No error added to label		Reacted correctly, no further action required.
“555555” Valid Extreme	No error added to label		Reacted properly, no further action required.
“123456789123 456” Invalid Extreme	Error relevant to number length added to label		Correct error displayed, no further action required.
“123A123” Invalid	Error relevant to number consistency displayed.		Correct error displayed, no further action required.

The next stage of validation would be to validate the email address input. I believe it would be best for the code to recognize the criteria for a good email address as:

- Containing the character ‘@’.

- Character limit of 50

The character limit is to ensure no user is left unable to use the software fully due to their email address length, though an infinite length would be abusable and unsuitable for database storage.

By checking if the string has an '@' character, this is checking the structure of the string to ensure it would be valid as an email address - as using an '@' is a standard protocol for email addresses.

```
if (VEmail.Length > 50)
{
    end = false;
    Label135.Text += "\r\nEmail must contain 50 or less characters.";
}
if (AtVal(VEmail))
{
    end = false;
    label135.Text += "\r\nEmail must contain a '@'.";
```

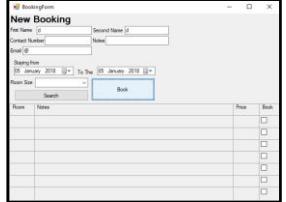
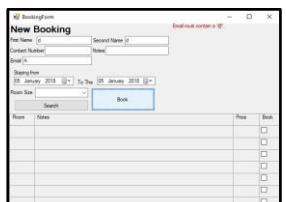
The validation for the length was just a standard .Length comparison, and if the length was over 50 characters I would add to the label a relevant message.

The '@' validation was more complex, as it required another function to be ran to find if the string contained a '@' character. By simply placing the result of this function (of boolean return), if it returned true - the fail statement would run and the error would be shown. The function in question is below:

```
private bool AtVal(string VEmail)
{
    for (int i = 0; i < VEmail.Length; i++)
    {
        if (VEmail[i] == '@')
        {
            return false;
        }
    }
    return true;
}
```

It simply runs a 'for' loop iteration, incrementing i at the end of each loop until it reaches the length of the 'VEmail' string. By then comparing each individual character the character is '@', the program can then return 'false' if there is a single match, but return 'true' (running the fail statement) if all the possibilities for matches end, and every character is exhausted and compared.

I then tested this '@' function:

Input	Expected Output	Output	Evaluation
“@”	No error displayed.		Correctly displayed no error. No further action required.
“A”	Relevant error displayed.		Displayed the relevant error, no further action required.
<pre> if (Notes.Length > 50) { label35.Text += "\r\nNotes must contain 50 or less characters."; } //NOTES END </pre>			

The final visitor validation was the notes sector, which quite simply just needed to be less than 50 characters as it was a much more open ended field for entry. It worked with a simple if statement - and I felt as this code had been previously tested, it was not necessary to test it again in the ‘notes’ context.

I then moved on to the validation of the dates being inputted - where I would need to validate that the dates are chronologically correct. In pseudocode, this could simply be presented as:

Is Date1 ahead of Date2?

The only barrier with this size comparison is that it isn’t simply comparing arithmetic size, as I am dealing with a ‘date’ data type.

I investigated how the program deals with a date input, and how it is formatted - and from there I could make the necessary arithmetic comparisons.

During researching and investigation how I would compare these dates, I found that there was already a sub function which could compare two dates, and return an integer value relating the chronology of the two dates.

My first attempt using this .compare function is as follows:

```

if (DateTime.Compare(DateStart, DateEnd) >= 0)
{
    end = false;
    label35.Text += "\r\nThe second date must follow the first date.";
}
//DATE END

```

As shown using DateTime.Compare (and feeding in the parameters of both the dates), I compared the result of this comparison to 0 to run the fail validation, as the comparison will return “1” if the second date is behind the first date, and will return “0” if the dates are the same. It will return “-1” if the first date is behind the second date, hence a -1 return would indicate a validated pair of dates - and the fail validation message will run if the return is either 0, or higher than 0 (including the ‘1’).

I then tested this validation:

Input	Expected Output	Output	Evaluation
07/01/2018 - 08/01/2018 (Valid)	No error message		Successfully recognized validation - no further action required
07/01/2018- 07/01/2018 (Invalid Borderline)	Error message displayed		Error displayed successfully, no further action required.
07/01/2018- 06/01/2018 (Invalid)	Error message displayed		Error displayed successfully, no further action required.

The last input in the form does not necessarily require validation, as it a drop down box control, meaning only preset values are available to the user.

Adding these options to the control was the problem, as the values had to be inputted manually using ‘comboBox1.Add’.

However, a more efficient method inputting this information was to use a variation of ‘.Add’ called ‘.AddRange’, which allows me to quickly input an array of values straight into the combo box display.

```

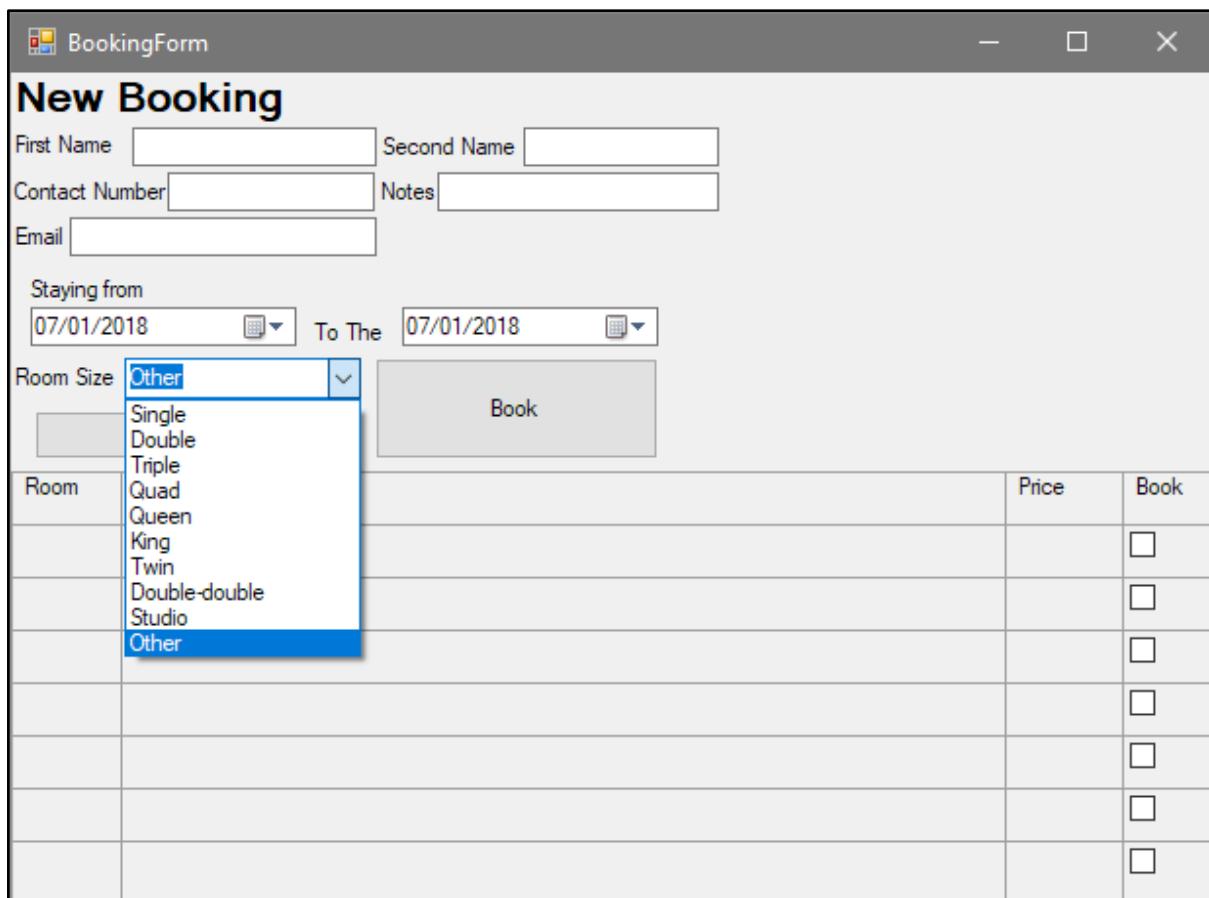
private void BookingForm_Load(object sender, EventArgs e)
{
    string[] combo = new string[] {"Single", "Double", "Triple", "Quad", "Queen", "King", "Twin", "Double-double", "Studio", "Other"};
    comboBox1.Items.AddRange(combo);
}

```

Shown is that upon the form loading, I create a string array filled with all the types of rooms, including an 'Other' option in case the hotel has any bespoke sizes or options which can then be included in the 'notes' option.

The rooms are in size order, starting with 'single' to make it easier for the user to navigate and find a certain size room - adding to the success criteria of 'efficient functionality'.

By then using 'AddRange' with the combo string array, this array data is then added to ComboBox1's selection options - and is viewable for the user upon the form loading. Shown below is how the user will access this combo box:



Later in development, I realized that the combobox must be validated to ensure that it is not blank, as it would therefore render the search function useless - so I added the validation:

```

//COMBO START
if (Combo == "")
{
    end = false;
    label35.Text += "\r\nA size of room must be selected.";
}
//COMBO END

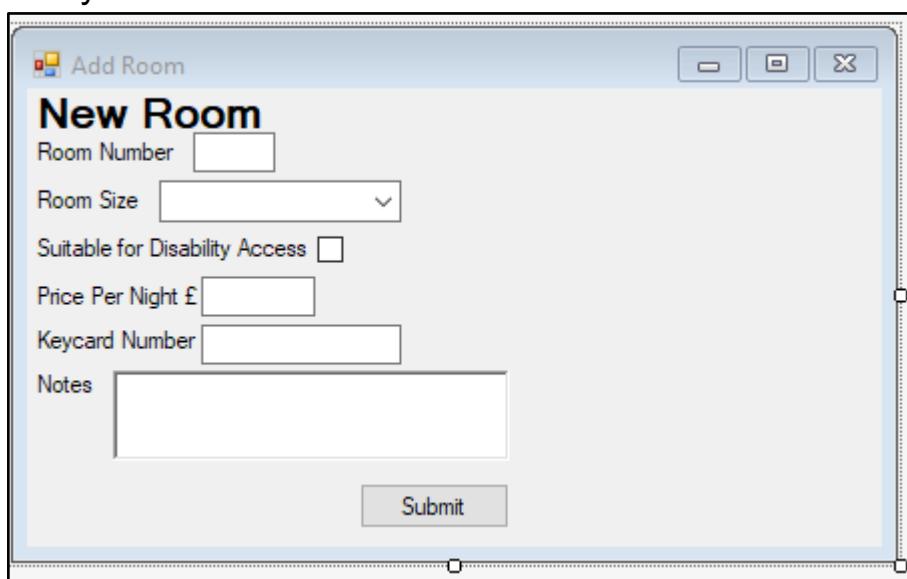
```

This validation would ensure that if the value of the combo box was blank, it would display an error and not allow the search function to continue. This helped development later on, as I had to possibility of needing to accommodate a blank answer to room size, which would be quite difficult and inefficient to deal with rather than just adding the validation and then not needing to create a workaround.

At this point, I feel as though all the methods of input into the visitor table were complete, and the ‘visitor’ side of inputting a record into the booking table was finished, though to move onto the search feature I would first need to focus on inputting data into the ‘room’ table, so that it would be easier to test and develop the search feature - as currently there is no method to input data into the room table.

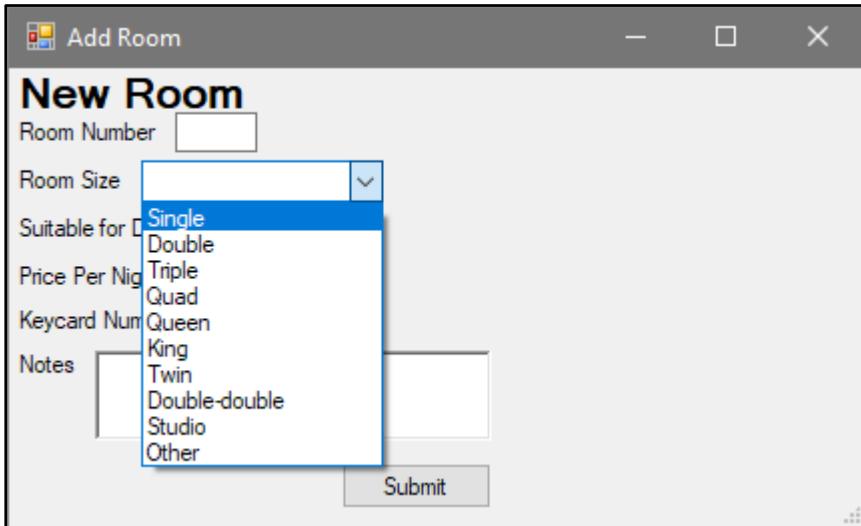
I made a new form, and implemented the previously developed method of making the form return to the ‘main menu’ form once closed. This form would open upon the ‘Add Room’ control being selected on the main menu form.

After reimplementing this navigation solution into the new ‘Add Room’ form, I began working on the controls of the form, to allow an inputs for every column in the ‘Rooms’ table.



Shown above is the structure of the ‘new room’ form. It uses many of the same elements as the previously developed ‘Add Booking’ form, but

rather simplified as its only purpose is to input information into a single table - unlike the booking which required input for both visitors and bookings, as well as the search function.



The combo box control works with the same array of room sizes as I used for the booking room combo box.

I also left space in the form for validation error diagnostics, which I will implement after I successfully have the form inputting values into the 'room' table.

Taking the inputs from the 'room' form and inputting them as a record was a problem I had encountered and solved for the 'visitor' table - meaning I could reuse the tested code for a more efficient solution to this similar problem - though I would just need to rearrange the names to suit the correct table.

```
private void DBVisitorAdd(string FName, string SName, string Numb, string VEmail, string Notes)
{
    string ConStr = LinkString();
    using (SqlConnection DB = new SqlConnection(ConStr))
    {
        DB.Open();
        using (SqlCommand Comm = new SqlCommand(
            "INSERT INTO Visitors (FName, SName, ContactNo, Email, Notes) VALUES(@FName, @SName, @ContactNo, @Email, @Notes)", DB))
        {
            Comm.Parameters.Add(new SqlParameter("FName", FName));
            Comm.Parameters.Add(new SqlParameter("SName", SName));
            Comm.Parameters.Add(new SqlParameter("ContactNo", Numb));
            Comm.Parameters.Add(new SqlParameter("Email", VEmail));
            Comm.Parameters.Add(new SqlParameter("Notes", Notes));
            Comm.ExecuteNonQuery();
        }
    }
}
```

I took the method I had previously written for the booking table, and implemented it into the 'new room' table - with a barrage of errors ensuing the implementation. I then began working on these errors,

mostly errors to do with the parameters I had passed no longer being relevant, and correcting the database's location to the 'room' table.

I also set up the 'new room' validation function, which would be called when the 'submit' control is pressed - to validate the input information before it is submitted to the table.

```
private void DBRoomAdd(string RNumb, string RSize, bool DAccess, string RPrice, string KeyNumb, string RNotes)
```

Above are the new parameters for the DBRoomAdd function - the function dealing with adding the inputs to the room table. In terms of data types, they are no different to the visitor table other than the boolean value of DAccess, as it receives its input from a checkbox control. Other than being cautious for errors when inputting the boolean value, this method should not prove difficult if using the previously tested code from the visitor's table.

I also had to reuse the code for gaining the address path of the database, for use in creating a signal with the database when inputting the data. As the path is identical in finding the database file, regardless of the table I am accessing, I did not have to make any changes to the method of 'LinkString()'.

```
private string LinkString()
{
    string path = (AppDomain.CurrentDomain.BaseDirectory);
    return ("Data Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename=" + path + "Database1.mdf;Integrated Security=True");
}
```

By giving LinkString() an instance in the new form's code, this fixed multiple errors in the reused code - as the code now as a destination of where to access the database.

I then began changing the individual input parameters, and their specific destination in the 'rooms' table.

```
DB.Open();
using (SqlCommand Comm = new SqlCommand(
"INSERT INTO Rooms (RoomNumber, RoomSize, DAccess, Price, Notes, KeycardNo) VALUES(@RoomNumber, @RoomSize, @DAccess, @Price, @Notes, @KeycardNo)", DB))
{
    Comm.Parameters.Add(new SqlParameter("RoomNumber", RNumb));
    Comm.Parameters.Add(new SqlParameter("RoomSize", RSize));
    Comm.Parameters.Add(new SqlParameter("DAccess", DAccess));
    Comm.Parameters.Add(new SqlParameter("Price", RPrice));
    Comm.Parameters.Add(new SqlParameter("Notes", RNotes));
    Comm.Parameters.Add(new SqlParameter("KeycardNo", KeyNumb));
    Comm.ExecuteNonQuery();
}
```

Above is the 'rooms' function for inputting data, with updated parameters. Once again, the SQL statement string uses the '@' character to input parameters inputted within the function, allowing variables to be inputted into the table.

Identity Specification	True
(Is Identity)	True
Identity Increment	1
Identity Seed	1

To reciprocate my actions with designing the visitors table, I had to ensure that

The Rooms_ID (primary key) value was incrementing by 1 for each additional record, to ensure there was no null error when adding a record. Above is the identity specification property in the Rooms.sql properties table - with changed values to ensure the value for the primary key is incrementing.

As expected, the only problem I imagined was the nature of the boolean value in 'DBAccess', as the datatype in SQL for this attribute is a 'bit' which would imply a 1 or a 0 - whilst the C# code is facilitating boolean values as 'True or False'. Only testing would tell if C# understands this communication error and converts the 'true' to a '1' before transferring the string to SQL - or if I will need to change the boolean value myself using selection programming.

Due to this scepticism, I then made a small test on the DBAccess part of the function, before fully testing the entirety of the function with validations:

Input	Expected Output	Output	Evaluation
A set of valid data with DBAccess as false	Record is created in the rooms table, with the correct data, and a 0 or false for DBAccess		Surprisingly stored the record correctly, with DBAccess being correctly denoted as 'false'. Conversion to SQL was successful, and validation can now begin.

I then moved onto the validations for the rooms form. In the objective of saving time and increasing development efficiency, I only properly tested the validations which were making a comparison different to one I had

previously implemented - as I would likely be reusing the code I had previously tested albeit renaming variables and comparison values. For example, I will not need to test a validation for a certain amount of characters as I have previously written the code for this when validating the booking visitor inputs.

The first validation was the room number which:

- Could not be blank
- 4 characters or less long
- Comprised only of numbers

I could piece the ‘blank’ and the ‘4 chars’ validation into one comparison, as they both dealt with the length of the string.

```
bool temp = true;
label8.Text = "";
Regex Num = new Regex("^[0-9]*$");

if (RNumb.Length==0 || RNumb.Length>5)
{
    temp = false;
    label8.Text += "\r\nRoom Number must contain 1-4 characters";
}
if (Num.IsMatch(RNumb) == false)
{
    temp = false;
    label8.Text += "\r\nRoom Number must only contain numbers.";
}

return temp;
```

The above code shows the beginning of the validation, as I label the temporary boolean variable to be returned, as well as the numerical regex I will be using for the ‘number only’ validations. Also setting the error diagnosis label to blank upon each time the validation is called.

The following ‘if’ statement then deals with the length of the room number validation, running the statement if “**the length of the string is equal to 0 OR(||) The length is over 4 characters long**”

The next ‘if’ statement uses the numerical regex to compare the string, to see if it matches with the range of numerical characters set by the regex class. If it does not match the regex (returns false) the validation is ran, and the error displayed in the label.

The next input which required validation was the ‘price per night’ of the room. As I provided the ‘£’ sign already, I would only need to validate

that the number inputted matches the layout of a money value - to allow insertion into the database. I would need to ensure the input only contains numbers at a maximum of two decimal places, hence I would need to create a regex for this.

This one criteria can be met with one regex:

```
Regex Mon = new Regex("^[0 - 9] * (\.\[0-9]{1,2})?\\$");
```

The regex shown allows either the numerical values '0-9' initially, and then uses the '*' character as a means of saying "OR' the regex is valid for containing the numbers 0-9, as well as to 1 or 2 decimal places" (made using the {1,2} segment). This regex now allows just plain integer numbers, or values to 1 or 2 decimal places.

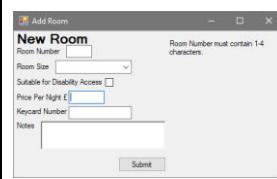
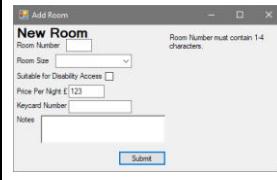
I then implemented this as a validation.

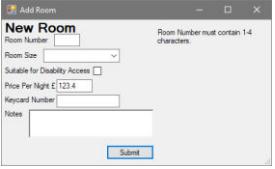
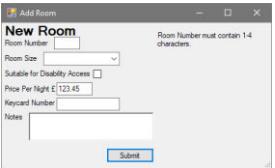
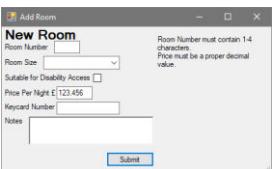
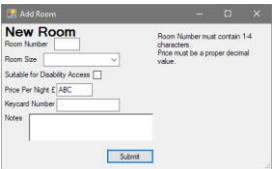
```
if (Mon.IsMatch(RPrice) == false)
{
    temp = false;
    label18.Text += "\r\nPrice must be a proper decimal value.";
}

return temp;
```

This validation should hopefully filter all the incorrect results. It compares the input value to both the regex criteria, and should reject all invalid values, even a null input.

I then tested this validation:

Input	Expected Output	Output	Evaluation
"" (Invalid)	Error displayed in label		A null input was not recognized as a problem - another comparison must be implemented
123(Valid)	No error displayed		No error displayed - no further action required.

123.4(Valid)	No error displayed		No error displayed - no further action required.
123.45 (Valid)	No error displayed		No error displayed - no further action required.
123.456 (Invalid)	Error displayed in label		The error successfully displayed - no further action required.
ABC (Invalid)	Error displayed in label		The error successfully displayed - no further action required.

Reflecting on my testing, I added another comparison to ensure the 'null' value was not able to clear validation.

```
if (Mon.IsMatch(RPrice) == false || RPrice.Length == 0)
{
    temp = false;
    label8.Text += "\r\nPrice must be a proper decimal value.";
}
```

By using '||' as an OR operator, the validation code now runs if the input has a length of '0'.

After retesting the code with the same input; with successful outputs, all the test were now complete and I could move on.

As the keycard input is quite volatile in terms of which information will be appropriate for it, I decided not to yet validate the input as it is subject to change - and even removal later on.

The notes section used the same validation as the booking form - only needing 50 or less characters to allow transferal to the database:

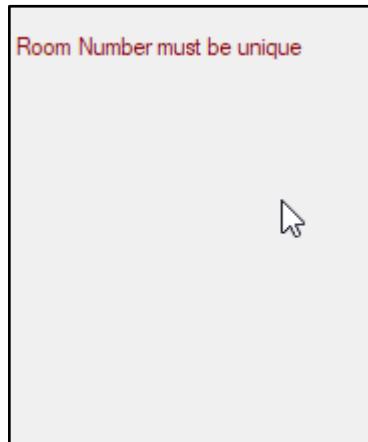
```
if (RNotes.Length > 50)
{
    temp = false;
    label8.Text += "\r\nNotes must contain 50 or less characters.";
}
```

With the database now having an inlet for room information, I could now develop the search function.

To later make sure no room duplicates could occur, I later added an extra validation before the room is added to the database:

```
for (int i = 0; i < RoomList.Count; i++)
{
    if (RNumb == RoomList[i].RoomNumber)
    {
        temp = false;
        label8.Text += "\r\nRoom Number must be unique";
    }
}
```

This is simply another check, which iterates through every room already in the database, and if it detects that the number entered is a match to any of them, it will return temp as false and complete the validation - leaving a message like this:



This will be useful to the calendar additions later on.

Search Function

My initial plan for the search function in terms of technicality, would be that I send an SQL query to the database using the parameters entered by the user. By then performing the query, it would return the matches and create an *object* for each match - to temporarily represent each matching room in the database. For this, I would first need to create a class called 'TempRoom', with all the attributes matching each relevant column in the database so that they may be stored in the C# code. I would then store the object names in an array data structure each time a matching room is made into an object.

I would use an array, as it would suit the need to then empty the array via iteration into the table - to be visualized by the user.

The only data I will need to transfer from the database about the rooms are:

- Their ID (So that the chosen room can be identified, once picked by the user.)
- Notes on the room (to be displayed in the table to be communicated to the user when deciding the room)
- Price of the room (To work out the length of their stay multiplied by the price of the room, to display and store a final cost).

I began by creating the 'TempRoom' class:

```
public class TempRoom
{
    private string _id;
    private string _num;
    private string _notes;
    private string _price;

    public string Price { get => _price; set => _price = value; }
    public string Notes { get => _notes; set => _notes = value; }
    public string Num { get => _num; set => _num = value; }
    public string Id { get => _id; set => _id = value; }
}
```

Above is the initial design of the TempRoom class with attempted encapsulation. Using encapsulation will help in terms of organization, when I later need to make changes to the specific objects - by only being able to interact with the attributes via the 'get, set' functions via the public strings.

At the moment, I do not think the class will require any behaviours, though this may change.

```
public string Price { get => _price; set => _price = value*(BookingForm.dateTimePicker1.Value - dateTimePicker2.Value) }
public string Notes { get => _notes; set => _notes = value; }
```

I attempted to use the full potential of encapsulation, by setting the value of 'price' using the 'get set' functions. By setting the price to the value retrieved from the database, and then multiplying it by the difference of the dates in the form, this would give a quick value for the total price - however this could not work as the instance of the TempRoom class is outside the recognition of reference the form's controls - therefore I would need to make the price changes when instantiating the object.

I then moved onto how the form would react to the 'search' control being activated. When the control is pressed, it will run the validation method once more to ensure the inputs are suitable.

```

private void button1_Click(object sender, EventArgs e)
{
    if (Validation(textBox1.Text, textBox2.Text, textBox3.Text, textBox4.Text, textBox5.Text, dateTimePicker1.Value, dateTimePicker2.Value, comboBox1.Text))
    {
    }
}

```

Update | Script File: dbo.Rooms.sql*

	Name	Data Type	Allow Nulls	Default
1	RoomID	int	<input type="checkbox"/>	
2	RoomNumber	nchar(10)	<input type="checkbox"/>	
3	RoomSize	nchar(10)	<input type="checkbox"/>	
4	DBAccess	bit	<input checked="" type="checkbox"/>	
5	Price	money	<input type="checkbox"/>	
6	Notes	text	<input checked="" type="checkbox"/>	
7	KeycardNo	nchar(10)	<input checked="" type="checkbox"/>	
8	Booked?	bit	<input checked="" type="checkbox"/>	0
			<input type="checkbox"/>	

I then also added another row to the Rooms database, a ‘Booked?’ column, which is a bit data type which shows whether or not the room is booked. This will need to be flipped once a room has been booked, to ensure a room which matches a search criteria but is not available for the guest’s desired visit length will not appear in the search.

At this point through researching how I would search the database, I found that instead of using SQL to search the database - it would be easier to use a hybrid method though mainly using C# to make the comparisons. I made this decision, as I felt it would be easier to take all the information from the table temporarily and then pick out the correct values, rather than going back and forth between the table with the form. Hence, I will iterate through every record in the table and set it to an object, and then search through the objects created for those matching the user’s criteria using C# selection and iteration, not SQL.

```

string Link = LinkString();
using (SqlConnection DB = new SqlConnection(Link))
using (SqlCommand Comm = new SqlCommand(
"SELECT RoomID AS ID, RoomNumber, RoomSize, DBAccess, Price, Notes, Booked? FROM Rooms", DB))
{
    DB.Open();
    using (SqlDataReader reader = Comm.ExecuteReader())
    {
    }
}

```

Using the same method of connecting to the database as inserting data, however it is slightly different as I have to call the ‘using’ SQL command before opening the database, instead of after it is open.

I use the ‘SELECT’ command, and select RoomID as the associated primary key of the table. I then additionally pass over the individual columns which I will want to retrieve from the database as it is being read. At the end, I add ‘FROM Rooms’ to designate the specific table from the database I am accessing.

I then instantiate a reader object, which will be able to read each record in the table and return it to the C# code, allowing me to set the returned data as object attributes.

```
using (SqlDataReader reader = Comm.ExecuteReader())
{
    if (reader.HasRows)
    {
        if (reader.Read())
        {
            ...
        }
    }
}
```

Using the reader, I then check if the table has rows. This is to catch and save the program from encountering errors, as if I set the reader to read without there being any rows it will cause an error. I then place the ‘reader.Read()’ function to only run if the table has rows.

```
if (reader.HasRows)
{
    while (reader.Read())
    {
        ...
    }
}
```

I then run a ‘while’ loop to iterate through these rows. This iteration will work as it will continue to read each record, and perform a piece of code for each record:

```
var OutputList = new List<TempRoom>();
if (reader.HasRows)
{
    while (reader.Read())
    {
        OutputList.Add(new TempRoom { Id = reader.GetInt32(reader.GetOrdinal("RoomID")) });
    }
}
```

The method in which I managed to create an individual object for each room in the table, is that a list is called before the iteration begins on the

reading of the table. This list is comprised of ‘TempRoom’ data - the same class of data I will use to store the individual pieces of info about the rooms for comparison.

I then use ‘OutputList.Add’ to repeat every time a new line is read. This means that each record will be added to a list of the class type - as multiple ‘objects’ contained into one object - the list.

This is useful, as it would seemingly be impossible to give an individual name to each object being created if I were not to use a list. It also bypasses the need for storing the data in an array, as by adding them to a list not only organizes the rooms - but also gives consecutive memory positions so that I may later iterate through the list easily.

Directly within the ‘add’ function, it fleshes out the object being added to the list of objects by reading each piece of column data from the table, and assigning it to the object’s attribute.

However, as it is retrieving from a foreign piece of code - the data must be given a data type as the code cannot automatically port the data type from SQL due to compatibility issues. Therefore, when I label ‘Id = reader.GetInt32’, this means the reader following is going to be reading integer data - hence telling the code to expect integer data. Thus allowing me to assign an int value to an int value, and as these values are all validated there will not be an issue of improper data types being ported from SQL.

```
Id = reader.GetInt32(reader.GetOrdinal("ID")), Booked = reader.GetBoolean(reader.GetOrdinal("Booked?"))
```

Then on a rather long line of code, I could attribute each piece of data being retrieved from the SQL to the attribute in the object being added. All that was different was the reader’s expectancy for data type needing to be changed, as discussed above.

I use the behaviour of ‘GetOrdinal’ to retrieve the specific labelled data type retrieved via the SELECT command sent to SQL - meaning the ordinal is the data attributed to the following string. (For example, the ‘Booked?’ string ordinal will refer to the ‘Booked?’ column in the table.

Once the code had hopefully been retrieved, I moved onto the actual searching function so that I may fully test the it in its entirety.

Now that the list has been filled with ‘TempRoom’ objects, I needed to access these rooms individually for comparison to the search results. I did this using a ‘for’ iteration loop.

```
for (int i = 0; i < OutputList.Count; i++)
{
    if (RoomSearch(OutputList[i]))
    {
        ...
    }
}
```

Using this for loop, with the loop breaking when it reaches the length of the list using ‘OutputList.Count’, I can use the ‘i’ value as a means of iterating through every object in the list.

I set up an if statement which runs on the return of a ‘true’ boolean output from the ‘RoomSearch’ method - which is called with the parameter of the specific object in the list which the ‘i’ is referring to. Roomsearch will return true if ‘OutputList[i]’ is a suitable candidate - and the if statement will contain the code to display the record’s data in the table in the booking form.

The RoomSearch method in question required some thought in terms of logic, and some use of logic gates were required to solve how I would refuse the ‘disability’ aspect of the search.

As if the visitor does **not** require special access for their disability, then it will not matter if the room is suitable for those with a physical disability or not - though a visitor **with** special requirements will only be wanting a room suitable for them.

```
if (DBCheck.Checked == true && room.Db==false)
{
    return false;
}
```

Hence, I checked at the start of the search and returned false straight away if the user **does** require disability requirements, but the room is **not** suitable. This uses logic, as every other combination of user disability status and room status makes the room suitable for them.

For example, A (1,0) represents the visitor’s disability status, and B(1,0) represents if a room is suitable for disabled requirements or not.

There are 4 combinations:

11 A[^]B

10 A[^]¬B

00 $\neg A \wedge \neg B$

01 $\neg A \wedge B$

And as shown, on those coloured green are successful searches: hence I filtered out the single red result straight away before the other, more linear, validations were performed.

```
private bool RoomSearch(TempRoom room)
{
    if (DBCheck.Checked == true && room.Db==false)
    {
        return false;
    }
    //Requires proper time validation later on.
    if (room.Size == comboBox1.Text)
    {
        return true;
    }
    return false;
}
```

The additional if statement added concerns with the room size, where if the room's size matches the request room size, it will return true for the room to be a successful search match.

As shown in the note, later on this will need to be upgraded during the calendar's development, as there is currently no attribute to compare to if the room is even available during the requested time - which will be implemented when the time attribute becomes important during the calendar stage. This stage will be returned to, to ensure a room isn't booked twice during a single day.

```
int y = 1;
Label tabnum;
Label tabnote;
Label tabprice;
TimeSpan diff = Convert.ToDateTime(dateTimePicker2) - Convert.ToDateTime(dateTimePicker1);
for (int i = 0; i < OutputList.Count; i++)
{
    if (RoomSearch(OutputList[i]))
    {
        tabnum = (Label)tableLayoutPanel1.GetControlFromPosition(0, y);
        tabnum.Text = OutputList[i].Num;
        tabnote = (Label)tableLayoutPanel1.GetControlFromPosition(1, y);
        tabnote.Text = OutputList[i].Notes;
        tabprice = (Label)tableLayoutPanel1.GetControlFromPosition(2, y);
        tabprice.Text = Convert.ToString(OutputList[i].Price*diff.TotalDays);
        y = y + 1;
    }
    if (y == 7)
    {
        break;
    }
}
```

Before the loop of search begins, I create a few variables.

The integer 'y' variable is for counting the Y axis position of the table the rooms are being written into. As Y must be separate from i, as y should only increment when a record has been added to the booking form table, to increment to the next line of the table to ensure the correct stepping lengths are made. Also, the loop is broken if 'y' reaches 7, as that would mean the code would try to access the 8th line in the table (which does not exist) on the next iteration, hence this prevents the crash of trying to access a label on the 8th line.

The three label variables all relate to a different piece of information being inputted into the table:

Room	Notes	Price	Book
		<input type="checkbox"/>	

Tabnum is the number of the room, TabNote is the notes column, and Tabprice is the price column shown above. These label variables will be used to interact with the label controls within each empty cell.

Finally, the timespan variable 'Diff' is the difference between the two dates the visitor expects to visit for. By working out this difference by converting both dates into 'Timespan' data types, and then simply using arithmetic to take away the first date from the second date, I now had a variable containing a complex difference in time which I would use to extract the difference in days, to then produce the total price of the visit.

```
tabnum = (Label)tableLayoutPanel1.GetControlFromPosition(0, y);
tabnum.Text = OutputList[i].Num;
```

The iteration of the 'if' statement then begins, and on a successful search the data is then needed to be placed into the table. On a successful search, above is a snippet of how I edit the labels in the table. I first set 'tabnum' to be equal to the position in the 'tableLayoutPanel1' (the table), and this position is retrieved using 'GetControlFromPosition(0,y)' which uses the y value discussed before, which will begin at 1 to specifically miss the first line of the table titles.

The x value position will always be 0, as the 0 column in the table is always the room numbers - and respectively x is 1 for notes and 2 for price - yet y is still the same for the both of them.

After 'tabnum' is set to that position, the actual text is then set equal to the object's .num value - which will be saved in the list. The label's value is then changed, and notes then goes through the same process.

However, price's label setting is slightly different as it involved the arithmetic of producing the total cost:

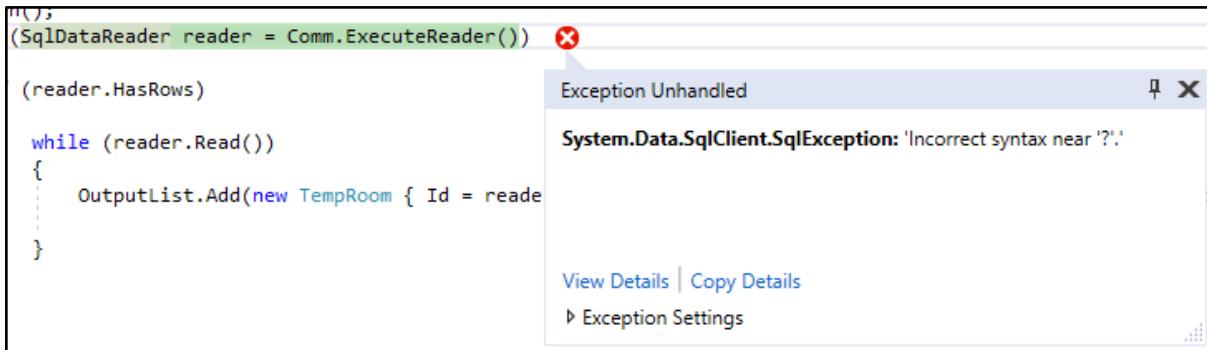
```
tabprice = (Label)tableLayoutPanel1.GetControlFromPosition(2, y);
tabprice.Text = Convert.ToString(OutputList[i].Price*diff.TotalDays);
```

Instead of directly setting the label.text to the object's price, I first multiply the object's price by the diff.TotalDays (the difference timespan, but only the days value). This then has a total cost to the visit in an integer form, where I then convert it to string using 'Convert.ToString' and pass the integer as a parameter in one line, to set the label.

```
y = y + 1;
```

I then increment the y value, to ensure the next line is picked on the next iteration.

I then tested this entire search function, by adding 10 rooms to the database with a mixture of different attributes, and making multiple searches:



The first error I received was when I had attempted a search for a single room - that of which I had added to the database via the 'add room' form in the same instance of the program. Upon searching, the program stopped and displayed the above error, concerning the syntax that the SQL reader was taking in.

Through this testing, I remedialized the program by removing the '?' characters from the SQL's search, and also removing the '?' character from the database in the 'Booking?' column, as the error seemed to communicate that using a '?' character was interfering with reader's understanding of the SQL code, perhaps that '?' has a specific string

manipulation.

```
    Iom(LINK)
    {"SELECT RoomID AS ID, RoomNumber, RoomSize, DBAccess, Price, Notes, Booked FROM Rooms", DB)}
```

I then removed this character from the database and updated it, as well as removing it from the C# code for a hopeful allowance of coherence between the SQL code and the C# code between the reader. After testing the code for the single bed once more, a different error appear - inferring that the '?' error had been fixed.

The screenshot shows a portion of a C# code editor with a tooltip overlay. The code is part of a loop that reads from a database reader and adds items to a list. A tooltip box titled 'Exception Unhandled' displays the error message: 'System.InvalidCastException: 'Unable to cast object of type 'System.Windows.Forms.DateTimePicker' to type 'System.IConvertible''. Below the message are links for 'View Details', 'Copy Details', and 'Exception Settings'. The tooltip has a close button in the top right corner. The code in the editor includes a TimeSpan conversion line: `TimeSpan diff = Convert.ToDateTime(dateTimePicker2) - Convert.ToDateTime(dateTimePicker1);`.

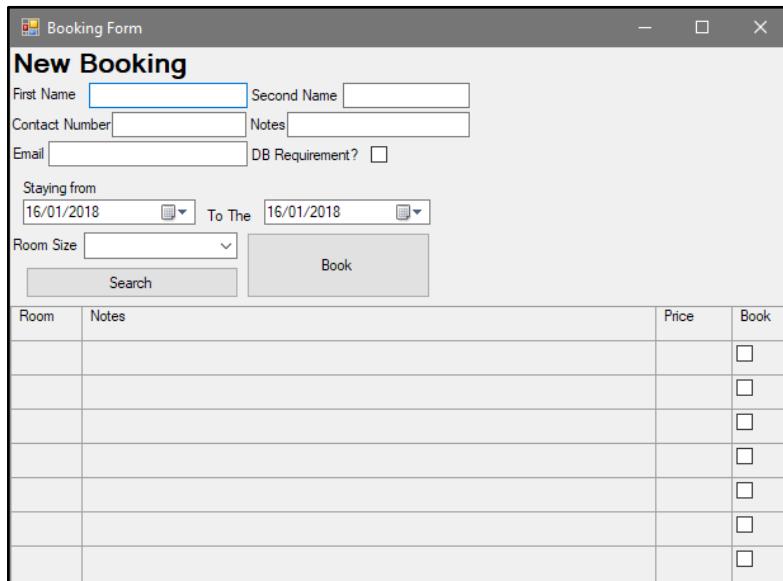
The next error I had to work out was the conversion of the 'dateTimePicker' values to the timespan data type. I realized the problem was to do with converting the data type, as the error stated it was unable to 'cast' the data to the timespan data type when I attempted conversion, inferring that it was not compatible to the conversion - which I noticed was peculiar as the value of the date time field is definitely in a date format.

This was when I realized that I had attempted to convert the literal control that retrieves the date, not the physical date contained within the field. Hence, I removed the conversion, and changed 'dateTimePicker2' to 'dateTimePicker2.Value' which would refer to the actual value contained within the control - and as it was not referring to a raw date, this would be compatible with the TimeSpan data type, hence I could remove the conversions for them to be coherent. After testing the same single bed test once again, the conversion error was removed and once again another error tooks its place - inferring the problem had been resolved.

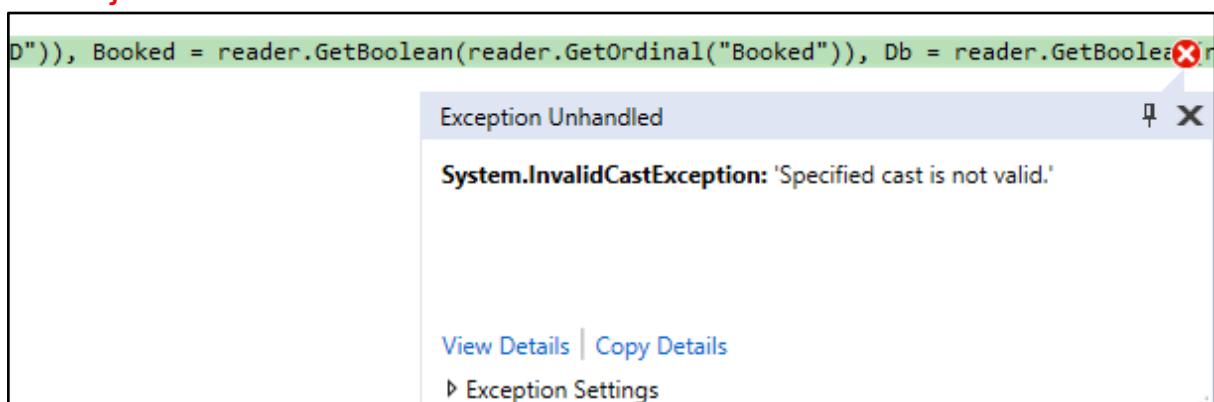
I used a method of testing to deduce the specific area of error, as I knew if I tried to search an empty room table, this 'if' statement which checks if

there are any rows in the table, would skip the complex search function, hence if no result occurred from a search of an empty table, I knew any future errors would be limited to within the search section within the if statement.

Luckily, upon searching the empty table, no result occurred.



As I now know the code works without any rows, I know that the errors are likely contained within the reader function. Hence, upon attempting once more to search for the single room and setting up some stepping, the error displayed in relation to the long line of code which instantiates the object within the list.



The error being displayed within the reader is shown, above, communicating that the error was to do with my data type conversions from the database. My first guess was that I had to use the 'byte' datatype as it was used in the database instead of the 'bool' data type - whilst they both represented a binary event. However, upon checking the details of the error, the IDE showed me the exact data the error was spawning from.

The error was being subjected from the most volatile data type - the 'money' data type in the database. It was the most volatile, as I was the most unsure on how the data would be stored and in what form. Upon researching, I found that the money data type was stored with a '£' sign included - hence it could not directly convert to double data as it contained the unicode character '£'.

I fixed this problem by updating the database to contained float data, instead of money data. This also required I backtrack to the 'add room' form to change how I sent the money data type, as the column in the database was now to a different data type meaning necessary changes had to be changed to how I 'packed' the data being transferred to the SQL.

```
OutputList.Add(new TempRoom { Id = reader.GetInt32(reader.GetOrdinal("ID")),
    Booked = reader.GetBoolean(reader.GetOrdinal("Booked")),
    Db = reader.GetBoolean(reader.GetOrdinal("DBAccess")),
    Num = reader.GetString(reader.GetOrdinal("RoomNumber")),
    Size = reader.GetString(reader.GetOrdinal("RoomSize")),
    Price = reader.GetDouble(reader.GetOrdinal("Price")),
    Notes = reader.GetString(reader.GetOrdinal("Notes")) });
```

Above is the final, working reader class with correct ordinal data type conversions. Hopeful that this error was resolved, I then ran a primitive test on the search function.

Input	Expected Output	Output	Evaluation
Search for a Single room	The single room I added is displayed in the table, with appropriate notes and price.		A logic error (as no crash occurred) is occurring which skips the function to write to the table.

Remedializing from this test, I decided to use stepping to identify the stage in which the code was deciding to not write to table. In terms of chronology, I knew that the data from the SQL table was being added to the list of objects, hence I knew the logic error lied within the method of 'SearchRoom', which sole purpose is to decide whether or not the room is suitable - hence the only decider of the table displaying information or not.

```

222     private bool RoomSearch(TempRoom room)
223     {
224         if (DBCheck.Checked && !room.Db)
225         {
226             return false;
227         }
228         //Requires proper time validation later on.
229         if (room.Size == comboBox1.Text)
230         {
231             return true;
232         }
233         return false;
234     }

```

I set steps on the two current decisions within the method, and ran the code to verify variables at these two points - and to visualize the decision they are making.

Name	Value	Type
DBCheck	{Text = "" CheckState = Unchecked}	System.Windows.Forms.Checkbox
DBCheck.Checked	false	bool
room	{HotelSystem.TempRoom}	HotelSystem.TempRoom
room.Db	true	bool
this	{HotelSystem.BookingForm, Text: Booking Form}	HotelSystem.BookingForm

At the first step, the variables were shown above. By using light white box testing, I followed that '`if (DBCheck.Checked && !room.Db)`' was ran with the variables DBCheck.Checked as false, and room.Db set to true, then the code would return a definite false - therefore skipping the statement which refutes the suitability of the room - therefore I knew the problem lied in the second step.

DBCheck.Checked	false	bool
comboBox1	{System.Windows.Forms.ComboBox, Items.Count: 10}	System.Windows.Forms.ComboBox
comboBox1.Text	"Single"	string
room	{HotelSystem.TempRoom}	HotelSystem.TempRoom
room.Db	true	bool
room.Size	"Single "	string
this	{HotelSystem.BookingForm, Text: Booking Form}	HotelSystem.BookingForm

Straight away, the problem became apparent, as I was simply comparing 'room.Size' to 'comboBox1.Text'. Whilst both of these values contained the same unicode strings, the data from the database had extra spaces tagged onto the end of the string - as a result of using a fixed CHAR data type in the database, meaning the leftover spaces cannot be left null, and are filled with spaces instead.

As a result of this stepping, I went to the database and began editing the data types to suit a VARCHAR set up, with the VAR indicating 'Variable', to suit any length.

RoomSize	varchar(50)
----------	-------------

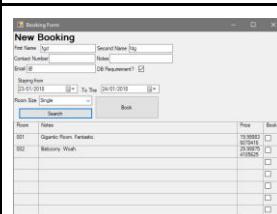
By changing the data type to varchar(50), this would take any string up to 50 characters but save the characters in a dynamic array, instead of a fixed array of 10 as I was previously using. This dynamic array will not need to fill remaining spaces with spaces.

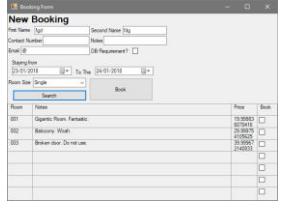
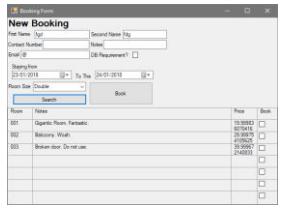
I also changed every string in the other tables to VARCHAR as well, to ensure this problem does not crop up again later in development.

On a later note, I then changed all the VARCHARS to NVARCHARs, as VARCHAR does not include Unicode, only ASCII - which would prove a problem when trying to input slightly unnatural characters, such as '@' for the email input.

With the whole section hopefully working now, I moved onto testing the search function:

Each of these tests were made using a room sample of 3 single rooms, two of those rooms having disability access. All three rooms have different prices.

Input	Expected Output	Output	Evaluation
Searching for single room with a visitor with a physical disability.	Only the two suitable rooms are displayed in the table, with their appropriate notes and prices.		Semi successful, in that the code knew the two rooms which required choosing to be suitable for the user's criteria. However, the price has been incorrectly calculated - possibly due to the timespan data type dealing with seconds, not days.

Searching for a single room with a visitor with no physical impairments	All three single rooms appear, displaying their appropriate notes and prices.		<p>Once again, the correct rooms were identified and returned, but the price was incorrectly worked out. Same as above, it seems the search is correctly working, but the price calculations are not working</p>
Searching for a double room.	No results should be present in the table.		<p>Does not display a blank table, but displays the same result - with no difference. It seems the table requires the function to 'clean' the table upon every search, to ensure there are no leftovers if multiple searches are necessary.</p>

To remediate on these results, I first began working on the pricing problem. It seemed I simply had to change the 'timespan' method of TotalDays to Days, as it seemed TotalDays returned a float value, thus using Days gave me the number of complete days between the two days - though I had to add 1 to this value as the dates did not count as a full day, as the space between the two days is slightly less than 1 for the

first, hence adding one day sorts out this time issue when reading how many days the time span is.

```

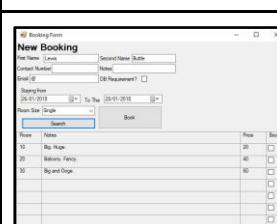
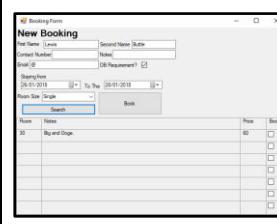
for (int n = 1; n < 8; n++)
{
    tabnum = (Label)tableLayoutPanel1.GetControlFromPosition(0, y);
    tabnum.Text = "";
    tabnote = (Label)tableLayoutPanel1.GetControlFromPosition(1, y);
    tabnote.Text = "";
    tabprice = (Label)tableLayoutPanel1.GetControlFromPosition(2, y);
    tabprice.Text = "";
}
y = 1;
for (int i = 0; i < OutputList.Count; i++)

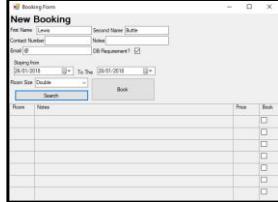
```

The next solution was to ensure the table is clear upon the search writing beginning. This simply required another set of writing iterations, where the code would iterate through every space similarly to when it writes the search results, but this time it simply fills the spaces with blank strings - to clear the table. It uses the same y integer, and resets back to 1 once the iteration is complete, and ready to be used when writing the search results to the table.

As I knew the search function was fine from testing, I only required to retest the aspects which included the price calculations, and the resetting of the table:

These tests were ran with 3 separate single rooms, each priced at £10, £20, and £30 per night. The £30 room is the only one suitable for those with a disability.

Input	Expected Output	Output	Evaluation
Stay in a single room from the 26th to the 28th (2 nights).	The £10, £20 and £30 room displays as £20, £40, £60 respectively.		Successfully displayed the rooms, with the correct prices.
Stay in a single room, requiring disability access (following the previous test)	Only the disability room is displayed.		The table was successfully cleared, and the relevant room was displayed independently in the table. - with

			the correct price.
Stay in a double room, when none are available.	All other table entries are removed- the table is blank.		The table successfully cleared.

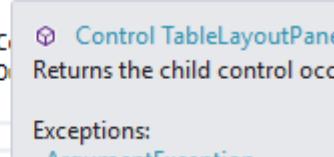
Proceeding this testing; while successful, I noticed a problem with the clarity of displaying the price - as the price column simply displayed the number without the '£' character. I added this by simple string manipulation, but may be advanced later on to accommodate a range of currencies.

```
tabprice = (Label)tableLayoutPanel1.GetControlFromPosition(2, y);
tabprice.Text = "£" + Convert.ToString(OutputList[i].Price * (diff.Days + 1));
y = y + 1;
```

	Price	Book
	£20	<input type="checkbox"/>
		<input type="checkbox"/>

After knowing the search function was complete, I began working on the final segment of stage 3 - the actual booking of the room. For this to work, I will need the program to associate the selection from the user with the object room being selected. So, for whichever checkbox is true, it should choose to associate this checkbox with the room to be inserted into the booking table. When booking is clicked, I will need to validate as to there only being 1 booking selected, hence I will need to iterate and check through the entire column. As to the checkboxes correspondence with the object rooms, I will need a list of 'selected' rooms, to then cycle through and link to the row which has a checked checkbox. Hence, I added a new list which added the object room every time one was added to the table during the search function:

```
tabnum = (Label)tableLayoutPanel1.GetControlFromPosition(0, y);
tabnum.Text = OutputList[i].Num;
tabnote = (Label)tableLayoutPanel1.GetControlFromPosition(1, y);
tabnote.Text = OutputList[i].Notes;
tabprice = (Label)tableLayoutPanel1.GetControlFromPosition(2, y);
tabprice.Text = "£" + Convert.ToString(OutputList[i].Price * (diff.Days + 1));
y = y + 1;
SearchList.Add(OutputList[i]);
```


Control TableLayoutPanel
 Returns the child control occurring at the specified position.
Exceptions:
 ArgumentException

This will now allow me to directly correlate the rooms in the table to an actual order.

Now part of this iteration, I will have a list of all the rooms added to the search table - with a corresponding checkbox available for selection.

When the 'Book' button is pressed, it already validates the visitor information and sends it if the validation clears - hence I only need to add checkbox validation (so that only one is selected when booking is submitted) to the function for Visitor validation.

However, a slight oversight was that I use the same function to validation function for both when 'search' is activated and when 'book' is activated, but I can only validate the checklists when 'book' is activated, as a user won't have selected a checkbox when they want to search for rooms.

Instead of duplicating the entire validation, I simply added another parameter to be inserted into the validation function - a bool value titled 'booking':

```
Validation(bool booking, string[] rooms)
```

The purpose of this new parameter is to tell which type of validation is being entered - a booking (whereas I will pass true as a parameter):

```
private void button2_Click(object sender, EventArgs e)
{
    if (Validation(true, textBox1.Text))
    {
        DBVisitorAdd(textBox1.Text);
    }
}
```

Or if it is a search, where I would pass false:

```
private void button1_Click(object sender, EventArgs e)
{
    if (Validation(false, textBox1.Text))
    {
        var OutputList = new List<Temp>();
        foreach (string room in rooms)
        {
            if (room != null && room != "")
            {
                OutputList.Add(DBRoomSearch(room));
            }
        }
    }
}
```

I can then use this input to choose to run a validation, depending on the value of this input:

```
//CHECKBOX START
if (booking)
{
    if (checkcheckbox() != 1)
    {
        end = false;
        label35.Text += "\r\nOnly one checkbox must be selected.";
    }
}
//CHECKBOX END
```

It uses a nested pair of if statements - the first to check if the validation should even run if booking is true (meaning it *should* validate the check boxes).

The second if statement then checks for the actual validation, which is ran in a separate method called 'checkcheck'. The function of the checkcheck method is to check how many checkboxes are checked, and return the amount. Hence, the failed validation text is only ran when the return is *not* equal to 1 - where it would be an inappropriate amount of checked boxes.

```
private int checkcheck()
{
    CheckBox check;
    int n=0;
    for (int i=1;i<8;)
    {
        check = (CheckBox)tableLayoutPanel1.GetControlFromPosition(3, i);
        if(check.Checked)
        { n=n+1; }

    }
    return n;
}
```

The checkcheck method in question is shown above. I create the variable of type 'Checkbox' to hold temporary values of the checkboxes it will be checking.

I then set the return integer holder as 'n', beginning at 0. As shown, I then use a for loop for int 'i' (starting at 0 to avoid the first row in the table - the header row which will *not* contain a checkbox.) The loop will break upon 'i' reaching 7, which is the total number of rows in the table minus 1 to accomodate for the fact the row 'pointers' begin at 0. Within each loop, I then set 'check' as the value of the checkbox control in the table located at (3,i), using i as a means to iterate through each row systematically, and 3 as a constant column location for the checkboxes. I then use an if statement to check if 'check.checked' is true, meaning the control is checked by the user. If true, 1 is added to n - holding a tally of each time a checkbox is found through enumerating through each row in the table.

After finishing this loop, the value of n is returned to the function - giving an indicator to the amount of checkboxes within the table.

With validation complete, I continued the booking process past the visitor being added to the database. First, I would need to link the box which is

checked to the specific object stored in the list which is made in the search process - when the objects are being placed into the table. A problem however, was the accessing of this list from the other segment of the form - displaying an error explaining that the list was not available in the same context.

```
public partial class BookingForm : Form
{
    public List<TempRoom> SearchList = new List<TempRoom>();
    public BookingForm()
    {
```

Hence, I had to make the list variable public - so now it may be accessed by any method within the form - and it will not need to be passed through as a parameter.

```
private int checkcol()
{
    CheckBox check;
    int n = 0;
    for (int i = 1; i < 8;)
    {
        check = (CheckBox)tableLayoutPanel1.GetControlFromPosition(3, i);
        if (check.Checked)
        { return n + 1; }

    }
    return -1;
}
```

To check the position of the lone checkbox, the above method is ran. It returns an int, and does not require any parameters as it interacts with the form's public elements.

It is extremely similar to the method of finding how many boxes are checked, though instead of returning once the for loop is finished - it returns the value of 'n' once the loop finds a box checked - as it does not need to continue and would be inefficient to compute the rest of the column - as it is only looking for the destination of the checkbox, not the total amount as the code already knows there will only be one checkbox checked, due to the validations running beforehand. However, to satisfy every outcome, I set the code to return -1 as an error integer if the case occurs, however this should be impossible to reach outside the for loop, as this method will only run once the previous validations are completed successfully, however I left this return just to communicate a logic error, if this -1 is reached within testing later on.

I did a quick test on this method, to see if any errors would occur. I used stepping to see the value of 'n' once it returns, when the checkbox was

inputted as the 3rd checkbox down in rows, meaning n should have been returned as '3'.



However, as a result of the quick test - n was shown to have returned as '1' from the stepping. Without an error being displayed, I knew this was a logic error.

I repeated the test with n expecting to be returned as '4', but n was returned as 1 once more. After looking at the code and mentally running each line of code logically (white box testing), I concluded that the method had carried an error from the previous validation method - as n was totally unnecessary unlike it being needed during the validation counter checkcheck method.

```
for (int i = 1; i < 8;)
{
    check = (CheckBox)tableLayoutPanel1.GetControlFromPosition(3, i);
    if (check.Checked)
        { return n + 1; }

}
```

Following this using white box testing, it becomes apparent that each iteration is not incrementing n at all - hence it was returning n(set as 0) +1 (hence returning 1 in total every time).

Remedializing from this testing, I fixed the bugs with the method whilst also making it slightly more efficient:

```
private int checkcol()
{
    CheckBox check;
    for (int i = 1; i < 8;i++)
    {
        check = (CheckBox)tableLayoutPanel1.GetControlFromPosition(3, i);
        if (check.Checked)
            { return i; }
    }
    return -1;
}
```

I totally removed the variable 'n' as it was not necessary - as I already had the counter of 'i' incrementing upon each loop iteration. Hence, this counter would be ready to return as a value rather than incrementing two variables at once, when only one needed to be returned.

BookingID	int	<input type="checkbox"/>
VisitStart	date	<input type="checkbox"/>
VisitEnd	date	<input type="checkbox"/>
Price	float	<input type="checkbox"/>
Notes	text	<input checked="" type="checkbox"/>
RoomID	int	<input type="checkbox"/>
VisitorID	int	<input type="checkbox"/>

The next stage was to actually book the visits. The information I would need to pass via the SQL commands are shown above.

- BookingID is calculated automatically, though I may need to be aware of the auto incrementation which needs activating as I have previously encountered
- Visit start and end are stored in the form controls, though may need to be converted to just a date format.
- Price may be calculated in the same way it was to be displayed in the table - without the '£' character as it must be stored as float
- Notes may begin as a blank field input - though may be used later in the calendar table section.
- RoomID and VisitorID I suspect to be quite simple - simply connecting the relevant IDs from the database to ensure they are present as foreign keys in the booking table - though this may require extra attention other than just inserting the ID of the visitor and room. I may need to retrieve the VisitorID from the visitor table - as I already have the roomID retrieved and stored in an object.

```
DBBookingAdd(SearchList[checkcol()]);
```

I separated the method of writing to the booking table as I did with the visitor database input. The parameter passed through is the room object in the list attributed with the result of 'checkcol()', the previously discussed method which finds the integer location of the checked checkbox. This then passes the relevant object, by using this result of checkcol to send the location in the list to the method. Everything else which needs sending to the database is either public and can be retrieved from within the method, or may need a separate method to retrieve them (Mainly the visitorID).

```

private void DBBookingAdd(TempRoom choiceroom)
{
    TimeSpan diff = dateTimePicker2.Value - dateTimePicker1.Value;
    string ConStr = LinkString();
    using (SqlConnection DB = new SqlConnection(ConStr))
    {
        DB.Open();
        using (SqlCommand Comm = new SqlCommand(
            "INSERT INTO Bookings (VisitStart, VisitEnd, Price, RoomID, VisitorID) VALUES(@VisitStart, @VisitEnd, @Price, @RoomID, @VisitorID)", DB))
        {
            Comm.Parameters.Add(new SqlParameter("VisitStart", dateTimePicker1.Value));
            Comm.Parameters.Add(new SqlParameter("VisitEnd", dateTimePicker2.Value));
            Comm.Parameters.Add(new SqlParameter("Price", choiceroom.Price * (diff.Days + 1)));
            Comm.Parameters.Add(new SqlParameter("RoomID", choiceroom.Id));
            Comm.Parameters.Add(new SqlParameter("VisitorID", VID));
            Comm.ExecuteNonQuery();
        }
    }
}

```

Shown above is the initial start to inserting into the booking table. It uses the same connection method previously used for the visitor insertion. It opens the database once more on this connection, and sends an INSERT statement, concerning the insertion of the listed ‘booking’ rows, which I have access to and can send. I then use string manipulation to insert the variables directly, using ‘@’ within the string, and inserting the values after.

Values like the dates and the price were easily calculated as they were freely accessible by the method - and the RoomID was able to be added as I had loaded the ID when creating the list of rooms as objects.

However, the VisitorID required some work to save - as the code had not got access to this ID directly without retrieving it from the visitor database - and this would also require searching for the record which was last entered.

OUTPUT INSERTED.VisitorID

I got around this problem by having an output from the visitor SQL statement. Adding OUTPUT INSERTED.VisitorID will provide an expected return from the SQL statement, returning the VisitorID from when the record is created - and the ID is incremented.

```

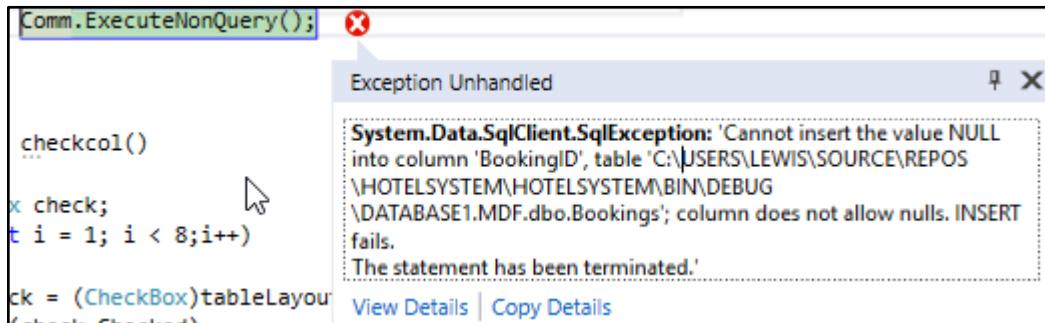
Comm.Parameters.Add(new SqlParameter("EMail", vEMail));
Comm.Parameters.Add(new SqlParameter("Notes", Notes));
Comm.ExecuteNonQuery();
VID = (int)Comm.ExecuteScalar();

```

I then use VID as a variable to store this return, using the ‘ExecuteScalar()’ to execute the command of inserting, and then returning the ID. A current dilemma is the use of ‘ExecuteNonQuery’ before this execution of the scalar. I feel as though this is a command to execute the code without an expected return, hence I am executing with no return, and then executing the code once more with a repeat - perhaps inserting two records into the table. This will need to be tested.

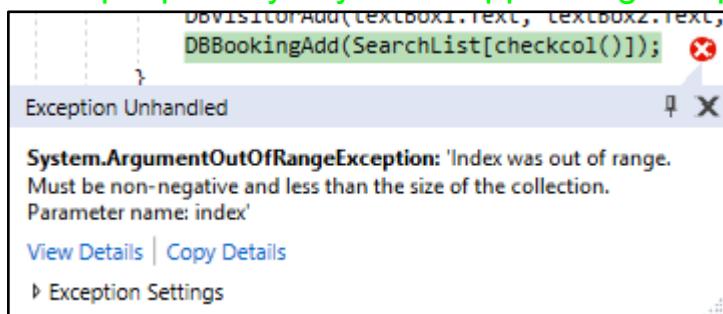
I then use this public variable VID to store this return - to then use in the previously shown VisitorID foreign key insertion in the booking table.

I then attempted to test-book a room with valid inputs, and the below error was displayed.

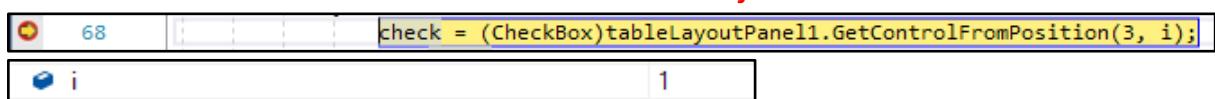


Analysing the error, it seems that one input variable was of null value - which interfered with a column refusing null values. Therefore, the fix was either to allow null values into different columns or to identify where this null value was coming from.

After this test result, I realized that the ID for booking was not incrementing automatically - hence giving a 'null' value by not providing a unique primary key. After appending this problem, I tested once more:



The next problem was with the range return of 'checkcol()'. It seemed that the return from the function exceeded the size of the search result list - and I first figured that this was a problem concerning the beginning of the list starting at '0' or something similar. Hence, I used stepping to find the value of 'i' when there is only one search result - where the return of 'i' should be 0 to indicate the 1st object in the list:

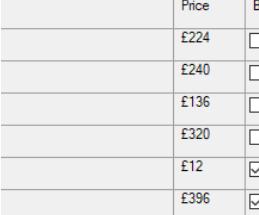
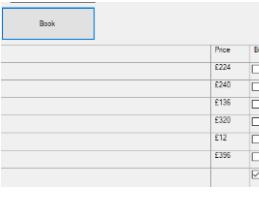


From the testing, 'i' was returned as '1' - breaking the limit of the list. Learning from this testing, I knew the solution was simply a logic error with how I was handling 'i'. Whilst it was incrementing on each loop, I forgot it started at 1 and not 0 to skip the first table row. Hence, when I

return the value of i - I use the value i-1 to recognize the object in the list.
Upon testing once more, I successfully booked a room with no errors.

```
DBBookingAdd(SearchList[(checkcol()-1)]);
```

With it now operational, I tested the booking function using a range of added rooms:

Input	Expected Output	Output	Evaluation														
Booking a double room for £224 (VALID)	The room, with all the appropriate details according to the booking, and saved in the table.	 <p>BookingID VisitorID VisitorName Price Notes RoomID VisitorID 1 31/01/2018 02/02/2018 224 NULL 1 1</p>	All the details were saved successfully, from the date formatting to the price. RoomID and VisitorID also matched the corresponding record. No further action required.														
Booking two rooms at once (INVALID)	The booking does not continue, and a validation error is displayed.	<p>Only one textbox must be checked.</p>  <table border="1"> <thead> <tr> <th>Price</th> <th>Book</th> </tr> </thead> <tbody> <tr> <td>£224</td> <td><input type="checkbox"/></td> </tr> <tr> <td>£240</td> <td><input type="checkbox"/></td> </tr> <tr> <td>£136</td> <td><input type="checkbox"/></td> </tr> <tr> <td>£320</td> <td><input type="checkbox"/></td> </tr> <tr> <td>£12</td> <td><input checked="" type="checkbox"/></td> </tr> <tr> <td>£396</td> <td><input checked="" type="checkbox"/></td> </tr> </tbody> </table>	Price	Book	£224	<input type="checkbox"/>	£240	<input type="checkbox"/>	£136	<input type="checkbox"/>	£320	<input type="checkbox"/>	£12	<input checked="" type="checkbox"/>	£396	<input checked="" type="checkbox"/>	The validation message successfully displayed, and no record for booking was created.
Price	Book																
£224	<input type="checkbox"/>																
£240	<input type="checkbox"/>																
£136	<input type="checkbox"/>																
£320	<input type="checkbox"/>																
£12	<input checked="" type="checkbox"/>																
£396	<input checked="" type="checkbox"/>																
Booking a blank row and pressing 'Book' (INVALID)	Nothing occurs.	 <p>Book</p> <table border="1"> <thead> <tr> <th>Price</th> <th>Book</th> </tr> </thead> <tbody> <tr> <td>£224</td> <td><input type="checkbox"/></td> </tr> <tr> <td>£240</td> <td><input type="checkbox"/></td> </tr> <tr> <td>£136</td> <td><input type="checkbox"/></td> </tr> <tr> <td>£320</td> <td><input type="checkbox"/></td> </tr> <tr> <td>£12</td> <td><input type="checkbox"/></td> </tr> <tr> <td>£396</td> <td><input type="checkbox"/></td> </tr> </tbody> </table>	Price	Book	£224	<input type="checkbox"/>	£240	<input type="checkbox"/>	£136	<input type="checkbox"/>	£320	<input type="checkbox"/>	£12	<input type="checkbox"/>	£396	<input type="checkbox"/>	Surprisingly, no error was created, and no blank record was sent to the table.
Price	Book																
£224	<input type="checkbox"/>																
£240	<input type="checkbox"/>																
£136	<input type="checkbox"/>																
£320	<input type="checkbox"/>																
£12	<input type="checkbox"/>																
£396	<input type="checkbox"/>																

Booking the same room, but set for different times	Both booking are saved, and the same RoomID is stored in the booking table.		<p>Both bookings were successful, and were correct in using the same RoomID for both bookings - which is important in that I can now confidently move onto the next stage, which will require accessing the roomID foreign key quite predominantly.</p>
--	---	---	---

With no action from testing required, I felt the booking and search functions were successfully implemented - so development grew onto stage 4.

I added to the booking form later on, by implementing another validation which would ensure no duplicate bookings could be made accidentally, or if two bookings for the same room could overlap.

```

}
else {
    for (int i = 0; i < BookList.Count; i++)
    {
        if (BookList[i].RoomNumber == RNum)
        {
            if (DateTime.Compare(DateStart.Date, BookList[i].VStart) <= 0 && DateTime.Compare(DateEnd.Date, BookList[i].VEnd) >= 0)
            {
                label35.Text += "\r\nA Booking already exists there.";
                end = false;
                break;
            }
        }
    }
}

```

It works by iterating through the result of BookList(), and comparing each value of the booking room number to the room number being inputted into the booking form. Once it finds a case where they match, it then compares their dates. Using the same long if statement used later on in the calendar development, if the dates interlude at any interval, it will trigger the if statement which will add the appropriate validation text, as well as change the validation boolean to false. It will also break the for

loop - to ensure the text is not written twice accidentally if it keeps looping as it only needs one case to disprove the validation.

Also, post development of this key stage, I made sure the dates reset to the current date, for quicker access, every time the form is loaded.

```
private void BookingForm_Load(object sender, EventArgs e)
{
    dateTimePicker1.Value = DateTime.Now.Date;
    dateTimePicker2.Value = DateTime.Now.Date;
```

This is done by just changing the value of the two datetimepickers and putting them equal to the `DateTime.Now.Date` value. (The current date)

Key Stage 4 - Calendar and Calendar Functions

Stage 4 will entail the use of the previously implemented booking feature, to implement the visualization of the bookings to the user. It will include the UI of the calendar being fully operational, as well as the refresh calendar, and daily visitor table (the table to appear to the left of the calendar, which loads all the bookings expected to be completed on the same date).

The first step of developing these calendar functions would be to begin on a function to fill the table with the booking data.

This function would need to load all the bookings from the table, save them to another list, to then be able to iterate through the list to fill the table. I have already done this segment of code in the booking section, though the only difference is that 'booking' will need to be made an object, and the iteration will need to run through the dates of the tables - which may require another array of the dates stored in the table rows. I began by creating the object for 'bookings' in the home screen form.

BookingID	int	<input type="checkbox"/>
VisitStart	date	<input type="checkbox"/>
VisitEnd	date	<input type="checkbox"/>
Price	float	<input type="checkbox"/>
Notes	text	<input checked="" type="checkbox"/>
RoomID	int	<input type="checkbox"/>
VisitorID	int	<input type="checkbox"/>
		<input type="checkbox"/>

The object would need to have attributes for all the above pieces of information - as well as only the relevant information from the visitor and room tables, as I will have the foreign keys to access these tables.

Initially, I will only store and retrieve these main pieces of information from the booking table, and I will add the later relevant attributes when I encounter their requirement - as I am currently unsure on which the program will require.

```
public class Booking
{
    private int _ID;
    private string _VStart;
    private string _VEnd;
    private double _Price;
    private string _Notes;
    private int _RID;
    private int _VID;

    public int ID { get => _ID; set => _ID = value; }
    public string VStart { get => _VStart; set => _VStart = value; }
    public string VEnd { get => _VEnd; set => _VEnd = value; }
    public double Price { get => _Price; set => _Price = value; }
    public string Notes { get => _Notes; set => _Notes = value; }
    public int RID { get => _RID; set => _RID = value; }
    public int VID { get => _VID; set => _VID = value; }
}
```

Above is the foundation of the booking options - using private variables and 'getting and setting' public variables to safely organize the information within the object attributes. If any changes or validations need to be done later on to the attributes, this can then be added through the 'get set' operators.

With this object set up, I began working on the procedure which would update the entire calendar - and would be called upon once the form is loaded, or if the 'refresh' button is pressed.

The update procedure will need to retrieve the bookings from the database and store them all in a list (a list of the booking objects). This list is then iterated through to populate the table, but only the bookings in the span of the rooms that are displayed, and bookings appropriate to the dates displayed in the table at the current instance.

Once this initial foundation is made, I planned that it will then need to be adaptable to when the user navigates the calendar - for instance when they move the 'x' direction ahead by a week, the bookings will then need to repopulate the table dependant on these new week of dates.

```
using (SqlCommand Comm = new SqlCommand("SELECT BookingID AS ID, Bookings.VisitStart, Bookings.VisitEnd, " +
    " Bookings.Price, Bookings.Notes, Bookings.RoomID, Bookings.VisitorID, " +
    " Rooms.RoomNumber FROM Bookings INNER JOIN Rooms on Rooms.RoomID = Bookings.RoomID", DB))
{
```

I added the room number input from the foreign key, as this would be the first value needing to be stored in the calendar on the left hand side. I retrieved this using the foreign key in the SQL statement as an ‘INNER JOIN’ command. This is used as an extension to the ‘FROM’ statement, where I also connect the ‘Rooms’ table using the ‘Rooms.RoomID = Bookings.RoomID’, where this sets the connection using the ID from the bookings table, to link these tables together. As a result of now having two tables to select from, I then had to ensure each attribute I was selecting was labelled with the table they were from, hence preceding each record with either ‘Booking.’ or ‘Room.’.

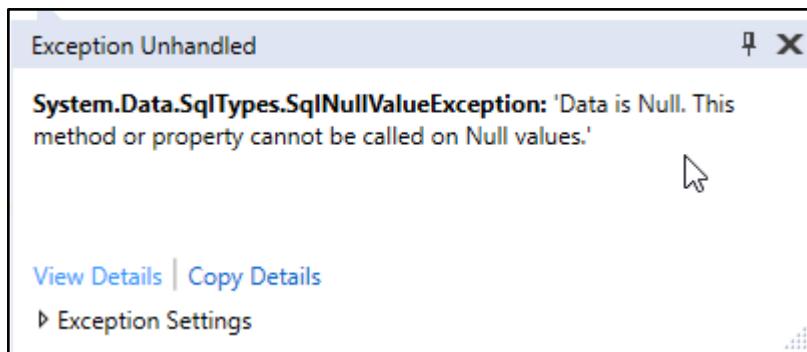
For example, just trying to retrieve ‘VisitStart’ would be confusing, as I have opened two tables using the SQL command, hence using ‘Bookings.VisitStart’ communicates to the command where this attribute is located. I then tested this retrieval using stepping:

Input	Expected Output	Output	Evaluation
Refreshing the calendar with a single booking.	Using variable checking, the booking list stores the correct room number and other booking information.	A screenshot of a debugger window. The code being stepped through is in a loop, reading values from a database reader into a 'Booking' object. The specific line causing the error is 'VStart = reader.GetString(reader.GetOrdinal("VisitStart"))'. A tooltip for 'GetString' shows its signature: 'string GetString(int ordinal)'. A callout box points to the 'GetString' method with the error message: 'System.InvalidCastException: Unable to cast object of type System.DateTime to type System.String.'. The stack trace shows the exception occurred at 'sh_C11'.	There seems to be an error with conversion from the database, prompting that I evaluate the booking object attribute data types.

Following this test, I remedialized the problem by first changing the object’s data types for the start and end date attributes.

```
private DateTime _VStart;
private DateTime _VEnd;
```

By setting them to `DateTime`, I supposed this would help fix the problem by being able to accomodate the return data from the database. Before retesting this aspect, I double checked that the other object properties matched the data types being returned from the database, though deemed there to be no other foreseeable problems. I then retested after remedializing.



From retesting, I received another error which detailed a problem with one property receiving 'null' data, when this should not occur - thus causing the error. From going through each data being received, I noticed only two properties would possibly receive a null value - the 'Notes' and the 'VisitorID' withdrawal.

The error most likely resided with the 'VisitorID', as I remembered I had not yet added the visitor table to the SQL statement, only the rooms table. From this testing and evaluation, I then began remedializing this problem within the SQL string.

```
INNER JOIN Visitors on Visitors.VisitorID = Bookings.VisitorID", DB))
```

Above is what I added to the SQL statement, to hopefully make the retrieval of the VisitorID, and later Visitor records a lot easier.

I then realized at this moment that VisitorID was stored in the booking table, hence it required no special SQL statement to access the Visitor table - though this will be useful later to access the table when loading the first and second name of the booking's visitor.

In terms of fixing the null error, I turned to the notes attribute, and tried to use a 'try' statement to contain any errors when the notes are received, though this was unable to occur as all the retrievals are as a single line of code, meaning that the use of a 'try' would be rendered useless, and any errors would just bypass the entire object creation.

To fix this problem with 'notes', I removed the field entirely from the booking table, as it was unnecessary as the table already stored notes on the rooms and the visitors, and as there was no insertion for booking notes, I decided it will not entirely be purposeful for later development. Upon removing the notes field from the SQL statement, I tested once more.

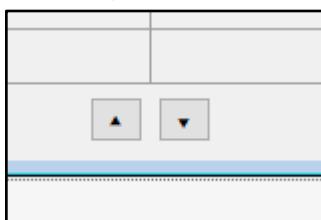
The result of this test was a success, where the list of objects was successfully added to - and no errors entailed. I could now focus on displaying this list within the table.

```
for (int i = 1; i < 16&&i<OutputList.Count; i++)
{
    ((RichTextBox)Calendar.GetControlFromPosition(0, i)).Text = OutputList[(i-1) + r].RoomNumber;
}
```

The above for loop is used to fill the first column with the rooms which have been booked. It uses 'i' set as 1 (not 0, to evade the first row of the table as to not interfere with linking a column with the row titles.), and then finding the position of (0,i) within the home table. The X value will always be 0, as the rooms only populate the first table column. The loop breaks when 'i' reaches 15, as 15 is the number of rows in the table. It also either ends (using && as an AND operator) if 'i' reaches the same size as the list of booking objects, meaning the filling will need to end if there are no more bookings to read from.

Using 'Calendar.GetControl...' this finds the position of (0,i), where I then set the control equal to the value of 'OutputList[(i-1+r).RoomNumber]'. This is using the list of bookings I just received, and setting each row as a different room from the bookings. I find them via 'i-1+r' in the list's indexing, as I take 1 from i to bring 'i' back down to 0, as I had set it to begin at 1 to skip the first table row. Taking 1 away from the 'i' index ensures every booking is read through this iteration.

I then also add on r, which is a public variable concerning the navigation aspect of the table. For example, if the user wants to move down a line of rooms as there are more than 15 rooms (thus overpopulating the table) - then they may navigate 'down' the table by pressing the down button, and alternatively back 'up' the table by pressing the up button:



Shown below, when pressing the down button (button4), this checks if $r!=0$, which will only negate 1 from r if r is NOT 0, thus protecting r from going negative (which would likely break the indexing). After updating r by taking 1 away, CallUpdate is ran once more to update this change - as r will now change the indexes explored by the for loop - as it adds a constant to the index.

Clicking the ‘up’ button will increment r by 1, and run CalUpdate, meaning the new index is for all the i range PLUS the value of r, which displaces the range in its entirety.

```
private void button4_Click(object sender, EventArgs e)
{
    if (r!=0){
        r = r - 1;
        CalUpdate();
    }
}

private void button2_Click(object sender, EventArgs e)
{
    r = r + 1;
    CalUpdate();
}
```

At this point I realized that the calendar would need to represent every room in the hotel - even those without any bookings. Currently, I am retrieving only the rooms which have bookings, and columning those rooms (which may also be duplicates of multiple rooms, as the same room may be booked twice, meaning a single room may be placed in two rows on the home screen table). From this reflection, I knew I would need to make another list of every room in the hotel to store and iterate through - as this would be much more simple and workable than only using the booked rooms.

```
using (SqlDataReader reader = Comm.ExecuteReader())
{
    if (reader.HasRows)
    {
        int b = 0;
        int c = 1;
        while (reader.Read())
        {
            if (b<=15+iv&&b>=iv)
            {
                ((RichTextBox)Calendar.GetControlFromPosition(0, c)).Text = reader.GetString(reader.GetOrdinal("RoomNumber"));
                c++;
            }
            b++;
        }
    }
}
```

Above is the fixed, more complex code for populating the room column. It is fixed, as it now displays every room in the database in the order they are entered into the database. It uses three integer variables to successfully perform this.

As shown, it is adapted from the previous code which only iterated through the booked rooms.

The above function is implemented within another SQLreader, reading only the room numbers from the room table. However, it uses an 'if' statement to only act on a certain number of these retrievals.

Using the if comparison ($b \leq 15 + iv \& b \geq iv$) translates to pseudo as:

If b is less or equal to 'iv+15'

AND

If b is bigger than or equal to iv

The three integer variables used in this counter are:

IV: This is called as a public variable outside of the sql reader, and is directly related to the previous use of 'r' to have a counter for the up and down navigation buttons. When the down arrow is pressed, iv increments by 1, and if the up arrow is pressed, iv decrements by 1 (limiting at 0). I then use this universal counter to see how far the user wants to navigate astray from the 'base' values where all the rooms are placed within the first column.

B: This is the loop counter, which will ensure the loop eventually ends. It begins at 0, and will increment after every iteration of the reader. This counter is used to find the segment of the 15 rooms which should be displayed, by keeping track of which iteration the reader is on. It increments indefinitely after each record has been read. (ending of one reader.HasRows).

C: This is another iteration counter, but ONLY when the if statement's content has been activated. The point of this counter is to know which table y position it should write to, as it cannot use the b counter to know this, as the b counter is independant to how the program would know to write in a consecutive order within the table.

Doing a preliminary white box test to show how this would work:

This white box testing uses 3 as a value, instead of 15 for matters of easier following. Also, this table is considering the database has quite a lot of room populace.

This is a white box test of the (whilst reader.hasrows) loop, for if it is ran once,

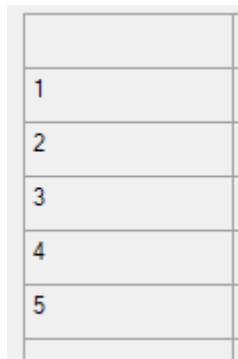
IV	B	C	$b \geq iv$	$b \leq 3 + iv$	Comment
0	0	1	TRUE	TRUE	Writes to Tab[0,1] with list index '0', increments C and B.

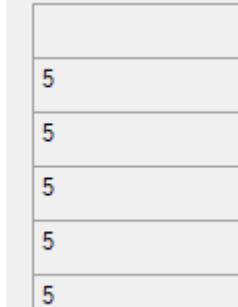
0	1	2	TRUE	TRUE	Writes to Tab[0,2] with list index '1', increments C and B.
0	2	3	TRUE	TRUE	Writes to Tab[0,3] with list index '2', increments C and B.
0	3	4	TRUE	FALSE	Does NOT repeat, waits for next input.
1	0	1	FALSE	TRUE	(IV Increments when the user interacts with the 'Down' Calendar button) One false, so only B increments.
1	1	1	TRUE	TRUE	Writes to Tab[0,1] with list index '1', increments C and B.
1	2	3	TRUE	TRUE	Writes to Tab[0,2] with list index '2', increments C and B.
1	2	3	TRUE	TRUE	Writes to Tab[0,3] with list index '3', increments C and B.
1	3	4	TRUE	FALSE	Does NOT repeat, waits for next input.

Shown above is a better visualization for following this algorithm, to see how these variables work together to numerically to figure out room placement.

I then tested this movement option:

This test was done with 5 rooms in the system.

Input	Expected Output	Output	Evaluation
No input	The five rooms should be displayed in the order that they have been inputted into the system. (In this test case, 1,2,3,4,5)		This worked correctly, meaning all the data was retrieved and stored in the table correctly - and this test shows that this is successful- however it does not prove the movement system works.

5 'up arrow' presses.	No change should occur from the first test, as it would be attempting to change 'r' below 0, five times. Thus, no changes should occur.		This worked correctly, showing that this movement system is validated so that the numbers do not just disappear from the table, and no confusing gaps are able to be left at the top of the table. No further action required.
5 'down arrow' presses (following the above test)	The numbers of the rooms should shift up the table, until they are all completely gone from the table.		Whilst the numbers did move up the table and disappear, they left a path of the last room number in the list - 5. Action will be required to fix this.

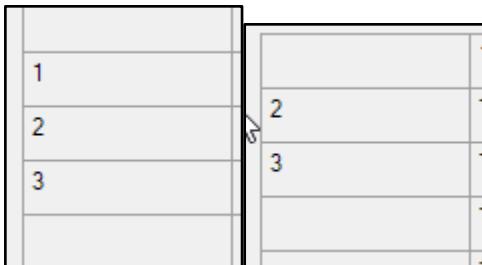
Remedializing from this testing, I figured that the problem was that the program was not clearing the table after each update, simply overwriting the necessary cells. I thought that I could add a new iteration which would clear the entire column before writing to it, but figured it would be much simpler to simply clear the the y value of 'c' *after* the iteration is complete, as this value of c would then be the maximum value in the column, and also the only cell which would have any possibility of not updating correctly.

```

        c++;
    }
    b++;
}
((RichTextBox)Calendar.GetControlFromPosition(0, c)).Text = "";
}
}

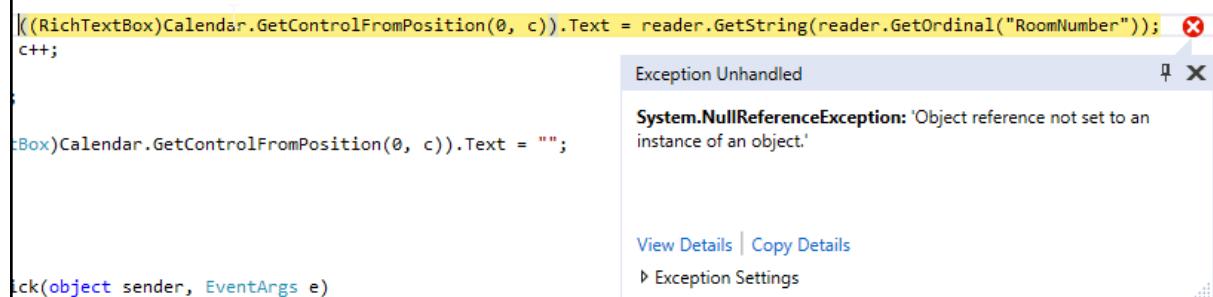
```

Shown above is this line of code placed after the iteration is fully complete. It uses c for this final Y coordinate, and replaces the cell with a blank string “`""`”. I then tested this using three rooms:



As shown, the rooms now correctly move across the column, correctly simulating a ‘blank’ space instead of leaving a trail.

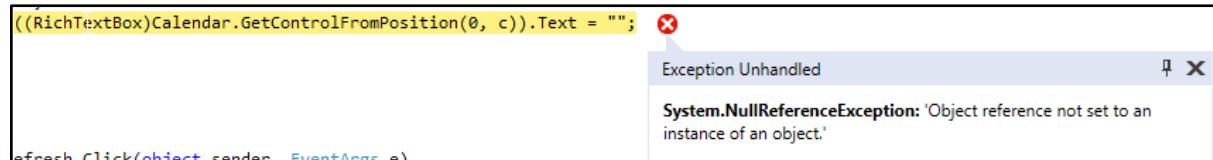
I then ran a simple test to check how it handles a room capacity larger than the column size. Refreshing the calendar with about 20 rooms in the system bought out this error:



As the errored statement did not include any of my own created objects, I learnt that the problem lay with the request of finding (0,c) in the table being unacceptable, due to c likely being a number larger than 15.

```
if (b<=14+iv&&b>=iv)
```

From this test, I identified the problem as the if statement loop’s conditions being too large, as I had used `b<=15` not `b<15`, and as c can go one larger than b, this bought c to 16 in the last loop - thus breaking the table. Hence, I tried lowering b and testing once more.



Upon testing once more with the same test variables, I had the same error though in a different section of code - indicating that the previous error had been fixed. This error was with the section I added as a result of previous testing, repairing the blank column cells. Though thinking about this solution, it is only viable for when the table is *not* filled with data fully, as c is always looking to be one more than b - which in this case maxes out at a value beyond the table's range.

```
if (c < 16)
{
    ((RichTextBox)Calendar.GetControlFromPosition(0, c)).Text = "";
}
```

From this testing reflection, it sought the solution to have a quick validation to ensure that the overwrite fix is only run when the table is *not* full, when c is less than 16. Hence, I inserted the same piece of code into an if statement with this comparison 'c<16'.

Upon testing this new fix with the same test variables:

	12		1
1	Te	12	T
1	Te	231	T
1	Te	231	T
1	Te	231	T
1	Te	231	T
1	Te	231	T
12	Te	234	T
12	Te	234	T
12	Te	5432	T
12	Te	5432	T
231	Te		T
231	Te		T
231	Te		T

No errors displayed, and the values successfully moved up and down the column with no problems.

With the user now fully able to interact with the calendar, I could now work on filling the individual cells with the booking data.

```
public DateTime[] Dates = new DateTime[7];
```

I started by creating a public array of datetime data types of length 7. This array 'Dates' will hold all the temporary dates which are currently stored in the top row of the table. As these values change with every 'left and right' navigation, this array will be updated every time the table is navigated in the x direction. By having this array, it will allow the code to easier iterate through the entire 2D space, as I now have the x axis stored in a list, and the y axis variables; constantly updated, in an array. Using the previously designed 'test' function, which filled every vacant placement in the table with 'Test', I could now reinterpret the table using this function, but changing what the actual test function is reading - as it now needs to take into account the different rooms which are currently displayed in the y axis, and the dates in which they are displayed across the x axis.

```
RichTextBox Rich;
for (int x = 1; x < 8; x++)
{
    for (int y = 1; y < 16; y++)
    {
        if (OutputList[(y - 1) + iv].VStart.Ticks < Dates[y].Ticks && Dates[x - 1].Ticks < OutputList[(y - 1) + iv].VEnd.Ticks)
        {
            Rich = (RichTextBox)Calendar.GetControlFromPosition(x, y);
            Rich.Text = "Test";
        }
    }
}
```

It runs in two concurrent for loops, the first going through the x values (by going from an int x, beginning at 1 to avoid overwriting the first column, and breaking when it reaches 7, as this is the size of the table in the x direction.) The same applies to the next loop which deals with the y variable, which breaks once it reaches 16 - also the size of the table in the y direction. This means that the following code will be executed within every empty cell in the table, and that code is as follows:

```
if (OutputList[(y - 1) + iv].VStart.Ticks < Dates[x-1].Ticks && Dates[x - 1].Ticks < OutputList[(y - 1) + iv].VEnd.Ticks)
```

Breaking this down individually:

```
if (OutputList[(y - 1) + iv].VStart.Ticks < Dates[x-1].Ticks
```

This start an if statement, and the comparison is if the object located in (OutputList(y-1), whereas I take away 1 as y begins at 1, meaning I must account for this constant change in range to apply for every value in the object range. I use y to navigate the list, as all the rooms are stored in the y axis - and I will use the x value to navigate the array of dates, as they are all stored in the first row in the x axis.

From this object in OutputList, I add the value ‘iv’ to the index, as this accounts for the other change in scope that the system may encounter, where the user may have navigated down a few rooms so that they are no longer in order. However, this constant movement is measured with ‘iv’, and has not changed since we last used it, hence adding it on will realign this index search to the correct value it is referring to.

With the correct object now selected, I use ‘.Vstart’ to select the start date of the room being compared. It then goes one step further, converting this date time into a date called ‘ticks’ using ‘.ticks’ on the end of the statement. This retrieves the date time and gives it a numerical value, as to how ‘large’ the date is. For example, a date which is later chronologically than another date, will have a *higher* tick value than the previous date. Hence, this is suitable to use for date comparisons and to see which dates come first.

I then compare to see if this object’s tick start date is smaller than the value in the array ‘Dates[x-1].ticks’, once again using x-1 to find the corresponding date in the array and convert it to ticks. It sees if the start date is smaller than the table date, as I want to code to check if the date in the table being checked is *larger* than the start date, yet *smaller* than the end date, which is checked in the other part of the if statement using the ‘&&’ operator as an AND:

&& Dates[x - 1].Ticks < OutputList[(y - 1) + iv].VEnd.Ticks)

Compared to the last statement, this is very similar in that it uses the same ‘dates’ value previously compared. However, the only difference is that it’s comparing if the dates ticks value is *lower* than the same object’s *end* date ticks. Hence, this statement only runs if the date being compared is between the booking start and end date - hence it can then set the cell to ‘test’ by assigning the cell to ‘rich’ and changing its value.

I then tested this with a single booking with a single room, to see if it would succeed in placing ‘test’ in a booked room.



Booking a room and then refreshing the calendar gave me the above error. It showed that the program was attempting to retrieve data from the object list which did not exist, as the index was out of range.

↳ SException	{"Index was o
↳ this	{HotelSystem
↳ OutputList	Count = 0
↳ Rich	null
↳ x	1
↳ y	1

Using stepping shown above, I could see that y had incremented to 1 whilst the maximum index of OutputList was only 0 - as it only had one value.

This testing prompted me to fix the program, by adding another condition to the y loop. As unlike the x loop, that had an infinite and fixed amount of dates which could not be exceeded - whilst the y range was limited to the discrete nature of the amount of rooms the hotel has. Hence, iterating through all 16 values of y would inevitably cause an error if the list was less than 16 objects large.

```
for (int y = 1; y < 16&&y< (OutputList.Count() - iv); y++)
```

This new if statement now has two comparisons, interconnected using the ‘&&’ operator as AND. The loop will continue as long as y is < (less than) 16 - to still ensure that the table’s limits are not broken, AND if y is less than ‘OutputList.Count()(length of the list) - iv (to account for table navigation movements). The second comparison is the addition, to ensure that the indexing of both the table *and* the outputlist are not broken. I then tested once more to see if ‘test’ would appear:

Input	Expected Output	Output	Evaluation																		
Five rooms with no bookings	The five rooms are loaded into the table, and nothing else appears.	<table border="1"> <thead> <tr> <th></th> <th>19.02.Monday</th> <th>20.02.Tuesday</th> </tr> </thead> <tbody> <tr> <td>1</td> <td></td> <td></td> </tr> <tr> <td>2</td> <td></td> <td></td> </tr> <tr> <td>3</td> <td></td> <td></td> </tr> <tr> <td>4</td> <td></td> <td></td> </tr> <tr> <td>5</td> <td></td> <td></td> </tr> </tbody> </table>		19.02.Monday	20.02.Tuesday	1			2			3			4			5			All five rooms successfully displayed. No further action required.
	19.02.Monday	20.02.Tuesday																			
1																					
2																					
3																					
4																					
5																					

Five rooms with a booking each - all at the same time	The word 'test' is written in all 5 of the correct cells, representing all 5 rooms.	<table border="1"> <thead> <tr> <th></th><th>19.02.Monday</th><th>20.02.Tuesday</th></tr> </thead> <tbody> <tr><td>1</td><td></td><td>Test</td></tr> <tr><td>2</td><td></td><td>Test</td></tr> <tr><td>3</td><td></td><td>Test</td></tr> <tr><td>4</td><td></td><td>Test</td></tr> <tr><td>5</td><td></td><td></td></tr> </tbody> </table>		19.02.Monday	20.02.Tuesday	1		Test	2		Test	3		Test	4		Test	5			Test was written in correct column, yet it missed just the last cell. This is a slight issue, and I imagine it to be an issue with one part of the scope requiring an extra incrementation.																																										
	19.02.Monday	20.02.Tuesday																																																													
1		Test																																																													
2		Test																																																													
3		Test																																																													
4		Test																																																													
5																																																															
Five rooms with two bookings each, following one another	Test is written in all the cells where the bookings are relevant. MUST Recognize the two different bookings.	<table border="1"> <thead> <tr> <th></th><th>19.02.Monday</th><th>20.02.Tuesday</th><th>21.02.Wednesday</th><th>22.02.Thursday</th><th>23.02.Friday</th></tr> </thead> <tbody> <tr><td>1</td><td>Test</td><td></td><td></td><td></td><td></td></tr> <tr><td>2</td><td></td><td>Test</td><td></td><td></td><td></td></tr> <tr><td>3</td><td></td><td></td><td>Test</td><td></td><td></td></tr> <tr><td>4</td><td></td><td></td><td></td><td>Test</td><td></td></tr> <tr><td>5</td><td></td><td></td><td></td><td></td><td>Test</td></tr> <tr><td></td><td></td><td>Test</td><td>Test</td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td>Test</td><td>Test</td><td></td></tr> <tr><td></td><td></td><td></td><td></td><td>Test</td><td>Test</td></tr> <tr><td></td><td></td><td></td><td></td><td></td><td>Test</td></tr> </tbody> </table>		19.02.Monday	20.02.Tuesday	21.02.Wednesday	22.02.Thursday	23.02.Friday	1	Test					2		Test				3			Test			4				Test		5					Test			Test	Test						Test	Test						Test	Test						Test	Whilst it was unexpected it would continue to write the extra bookings to the table, despite being dedicated to the same object as other bookings, it seems there is a problem with reading multiple bookings to write on one line - as though it finished the first rows and then moves onto the next, duplicate bookings and stores them on another set of rows. This will need to be fixed, to ensure they are written to the same row
	19.02.Monday	20.02.Tuesday	21.02.Wednesday	22.02.Thursday	23.02.Friday																																																										
1	Test																																																														
2		Test																																																													
3			Test																																																												
4				Test																																																											
5					Test																																																										
		Test	Test																																																												
			Test	Test																																																											
				Test	Test																																																										
					Test																																																										

			as their allocated object room. Initially, I think this could be sorted using a .find option when navigating the list.
Booking a single room, between 20/02 - 25/02	Every date between and including 20/02 - 25/02 should have 'test' written inside.		The 'test' strings are written in all the correct boxes, except for missing the last date - the 25.02. They were also one row too high for the room I selected. This may be an issue with the ticks.

To repair these testing problems, I first began with adding a new procedure called 'cleartable'.

```
private void ClearTable()
{
    RichTextBox Rich;
    for (int x = 0; x < 8; x++)
    {
        for (int y = 0; y < 16; y++)
        {
            Rich = (RichTextBox)Calendar.GetControlFromPosition(x, y);
            Rich.Text = "Test";
        }
    }
}
```

This procedure is called at the start of the 'UpdateCal' procedure, and is important to making sure the table is entirely clear before being added to. This will clear up any issues later on regarding the word 'test' not being overwritten when refreshed a second time.

Also, for this new procedure to be successfully called at the beginning of CalUpdate, I now had to incorporate the date writing function into CalUpdate, otherwise the dates would be removed when ClearTable is

called at the start of CalUpdate.

```
ClearTable();
//Dates refresh
for (int i = 1; i < 8; i++)
{
    Dates[i - 1] = DAnchor.AddDays(i - 1);
    ((RichTextBox)Calendar.GetControlFromPosition(i, 0)).Text = DAnchor.AddDays(i - 1).ToString("dd.MM.yyyy");
}

//DatesRefresh
```

I did this by changing DAnchor into a public DateTime variable, which is then directly changed by multiplying by 7 etc when then left and right buttons are pressed, in their own separate functions. With DAnchor now changing, the row of dates can now be written using this DAnchor at the start of CalUpdate.

```
if (DateTime.Compare(OutputList[(y - 1) + iv].VStart, Dates[x - 1])<=0 && DateTime.Compare(OutputList[(y - 1) + iv].VEnd, Dates[x - 1]) >= 0)
{
    Rich = (RichTextBox)Calendar.GetControlFromPosition(x, y);
    Rich.Text = "Test";
}
```

Also, to remedialize the last test which concerned the dates not correctly being filled in (albeit one row too high), I changed the comparison of the dates condition from using ticks. Instead, I used the datetime function of 'Compare' (which returns either 1,0, or -1 depending on the chronology of the two input date parameters.). Shown above, I simply just translated the previous statement using ticks into these two compare functions - which I felt was much more concise and efficient than using the ticks of the dates myself.

As shown, the statement takes the dates:

1. The start date of the booking
2. The end date of the booking
3. The date being compared in the table

Hence, the test is only written if [1] compared to [3] is smaller than or equal to 0 ([1]<=[3])

AND

If [2] compared to [3] is bigger than or equal to 0 ([2]>=[3])

It will place test in the correct cell.

At this point in development, I felt as though it would be much simpler to rewrite this section of code and begin anew. The section I had wrote was too particular in how it made its comparisons, in that it was selecting only a range of values in the list using the 'iv' constant, and was not broad enough to be malleable for later developments, as the code was very unclear and particular in how it handled variables. It was also how it was

written which caused many of the problems, including the constant misalign with placing the ‘test’ strings.

My new idea for writing this section would be to check **every** booking in the system, and compare it to every number room *visible* in the table. Hence, it can then check if a booking is needing to be represented in the table.

With this clear, it can then iterate through only the relevant row of the table, comparing the dates to this object it has found to be in the table. It can then make the decision on whether or not the dates are chronologically inclined.

```
RichTextBox Rich;
bool checkie;
int XVal=0;
for (int m = 0; m < OutputList.Count; m++)
{
    checkie = false;
    for (int numplace=1; numplace < 16; numplace++)
    {
        if(Calendar.GetControlFromPosition(0, numplace).Text == OutputList[m].RoomNumber) {
            checkie = true;
            XVal = numplace;
            break; }
    }
    if (checkie)
    {
        for (int D = 0; D < 8; D++)
        {
            if (DateTime.Compare(OutputList[(m)].VStart, Dates[D-1]) <= 0 && DateTime.Compare(OutputList[(m)].VEnd, Dates[D-1]) >= 0)
            {
                Rich = (RichTextBox)Calendar.GetControlFromPosition(XVal, D);
                Rich.Text = "Test";
            }
        }
    }
}
```

Above is the new section of code. It begins by instantiating the RichtextBox ‘Rich’, which will be used as it has previously been used - to contain a certain cell to write to.

The bool ‘checkie’ is made, and this will be used to check if the booking room number being checked is in the table (where it will be set to true) or if it is not in the table (where it will be set to false).

The int ‘XVal’ is made and set to 0, and will be used to set the value of the actual X position - if found - of the booking room number if found in the first column of the table.

It then begins a for loop, with integer ‘m=0’. The condition is that m will cycle through every number between 0 and the number of objects in the ‘OutputList’.

Checkie is reset to false every time a loop begins, as it can then be meaningful if it is reset to true later on - unique to that loop.

An embedded for loop is then ran, where an integer called ‘numplace’ begins at 1 (to skip the first blank cell in the first column). This loop

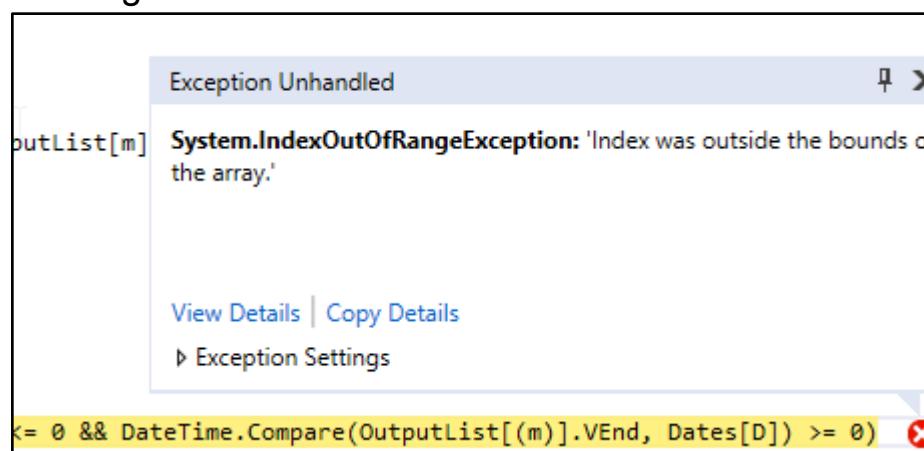
continues through all 15 cells in the first column, and compares each value in the rich text boxes to the original booking room name 'OutputList[m].Roomnumber'. Once this placement has been found, checkie is set to 'true' to show that the object's room number *is* in the table. It also breaks the for loop, preventing the 'numplace' integer from incrementing anymore, thus preserving it as the integer in which the object's number is held - for later locating.

Following this, the next piece of code is ran only if 'checkie' is true - to prevent any index errors if the room number was *not* in the table.

It contains a for loop, looping using 'D' from 0 to 8, where it will cycle through all the dates stored at the top of the table. It then compares to see if OutputList[m]'s start and dates fit between the date given. If it is successful, it will add 'test' to the cell in position (XVal,D) which is the identified value where the room number is stored, followed by the y position of 'D', whereby it will be relevant to that specific loop.

This is an overall improvement to the last function, as the order of travel is following down the x axis, and then stretching out across the y axis when it detects a booking match - unlike the previous function which just travelled through every cell and made a comparison(much more complex to follow).

The only disadvantage to this new function, is that it required every room to have a unique room number - or else it will not properly retrieve multiple cells - only break when it reaches the first duplicate. Hence, I will need to add a method of checking multiple room numbers in the booking form.



Upon testing this for the first time with a few rooms and bookings, this error was given. It seemed like a small error with array sizes or something similar.

I used stepping on the if statement to check the values, and lightly white box test the statement.

```
ates[D]) <= 0 &&  
D|7=
```

Using stepping, I saw that D had reached 7, which means it exceeds the array indexing which begins at 0 and ends at 6. By lowering the for loop conditions on D, this could be prevented.

```
for (int D = 0; D < 7; D++)
```

	20.02.Tuesday	Test	22.02.Thursday	Test	24.0
121		Test		Test	
122		Test		Test	
123		Test		Test	
124		Test		Test	

Upon testing once more with 4 rooms, there was no error with indexing. The above result is quite unexpected, however it did correctly write to the correct start and end dates - however, it wrote it to *all* the rooms and not just the ones I selected.

```
folFromPosition(D, XVal);
```

I noticed that the 'Test' strings were travelling downwards, and from this test realized that I had placed the 'D' and the 'XVal' the wrong way around when placing 'test'. I also noticed that test was overwriting the first row cells, so I deduced from the test that maybe this was a problem with the starting conditions of 'D', and I saw that D started at 0 and not 1, hence it was not skipping the first column, shown below.

	20.02.Tuesday	21.02.Wednesday	22.02.Thursday	23.02.Friday	24.02.Saturday	25.02.Sunday
12						
13						
Test	Test	Test	Test	Test	Test	Test
15						

```
for (int D = 1; D < 7; D++)
```

Hence upon changing this starting limit, I then tested properly:

Input	Expected Output	Output	Evaluation																														
5 rooms, a booking for each for the same dates	'Test' stretches out across the relevant dates - including both start and end dates	<table border="1"> <thead> <tr> <th></th> <th>20.02 Tuesday</th> <th>21.02 Wednesday</th> <th>22.02 Thursday</th> <th>23.02 Friday</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>Test</td> <td>Test</td> <td>Test</td> <td></td> </tr> <tr> <td>2</td> <td>Test</td> <td>Test</td> <td>Test</td> <td></td> </tr> <tr> <td>3</td> <td>Test</td> <td>Test</td> <td>Test</td> <td></td> </tr> <tr> <td>4</td> <td>Test</td> <td>Test</td> <td>Test</td> <td></td> </tr> <tr> <td>5</td> <td>Test</td> <td>Test</td> <td>Test</td> <td></td> </tr> </tbody> </table>		20.02 Tuesday	21.02 Wednesday	22.02 Thursday	23.02 Friday	1	Test	Test	Test		2	Test	Test	Test		3	Test	Test	Test		4	Test	Test	Test		5	Test	Test	Test		All the tests appeared in correct areas, except once again missing the final date of the booking.
	20.02 Tuesday	21.02 Wednesday	22.02 Thursday	23.02 Friday																													
1	Test	Test	Test																														
2	Test	Test	Test																														
3	Test	Test	Test																														
4	Test	Test	Test																														
5	Test	Test	Test																														
A single room, but with two different bookings.	There is a gap between the two bookings, and they are both on the correct dates	<table border="1"> <thead> <tr> <th></th> <th>20.02 Tuesday</th> <th>21.02 Wednesday</th> <th>22.02 Thursday</th> <th>23.02 Friday</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>Test</td> <td>Test</td> <td>Test</td> <td></td> </tr> <tr> <td>2</td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>3</td> <td>Test</td> <td>Test</td> <td></td> <td></td> </tr> </tbody> </table>		20.02 Tuesday	21.02 Wednesday	22.02 Thursday	23.02 Friday	1	Test	Test	Test		2					3	Test	Test			It correctly split the bookings across the two dates, albeit missing the end date off of both of them.										
	20.02 Tuesday	21.02 Wednesday	22.02 Thursday	23.02 Friday																													
1	Test	Test	Test																														
2																																	
3	Test	Test																															
A single room with a single booking, but it stretches across two weeks.	By navigating across the two weeks, every date will be viewable.	<table border="1"> <thead> <tr> <th></th> <th>20.02 Tuesday</th> <th>21.02 Wednesday</th> <th>22.02 Thursday</th> <th>23.02 Friday</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>Test</td> <td>Test</td> <td>Test</td> <td></td> </tr> <tr> <td>2</td> <td>Test</td> <td>Test</td> <td>Test</td> <td></td> </tr> <tr> <td>3</td> <td>Test</td> <td>Test</td> <td></td> <td></td> </tr> </tbody> </table>		20.02 Tuesday	21.02 Wednesday	22.02 Thursday	23.02 Friday	1	Test	Test	Test		2	Test	Test	Test		3	Test	Test			The dates did not rightly align, and they supposedly moved places when I went left and right across the dates.										
	20.02 Tuesday	21.02 Wednesday	22.02 Thursday	23.02 Friday																													
1	Test	Test	Test																														
2	Test	Test	Test																														
3	Test	Test																															

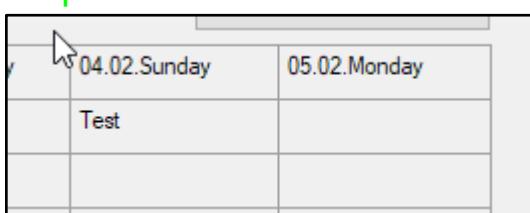
	20.02.Tuesday	21.02.Wednesday	22.02.Thursday	23.02.Friday	24.02.Saturday	25.02.Sunday
1	Test	Test	Test			
15	Test		Test	Test		
2	Test	Test	Test			
3	Test	Test	Test			
3332						
334						
345						
4	Test	Test	Test			
45						
46						
5	Test	Test	Test			
56						
76						
87						

The main complaint from the testing is the problem with the dates not aligning correctly with the bookings. I first remediated this from the tests, by changing the left and right button functions.

```
private void button1_Click(object sender, EventArgs e)
    //RIGHT DATE
{
    DAnchor = DAnchor.AddDays(7);
    CalUpdate();
}

private void button3_Click(object sender, EventArgs e)
    //LEFT DATE
{
    DAnchor = DAnchor.AddDays(-7);
    CalUpdate();
}
```

I removed the 'displace' variable, and simply added or took away 7 days from DAnchor depending on which button was pressed. This simplified the code, and perhaps fixed a few logic errors with the incrementation of 'displace'.



I also noticed a problem was that the last column of the table was not being given ‘test’ whatsoever, and I learnt this from the last test. I thought perhaps this was an index error, and that the loop ended prematurely. I looked at the ‘test’ write function, and rechecked the for loop conditions

```
(int D = 1; D < 8; D++)
{
    if (DateTime.Compare(OutputList[m].VStart, Dates[D-1]) <= 0 && DateTime.Compare(OutputList[m].VEnd, Dates[D-1])
```

I noticed that D was stopping when it reached 6, hence it was covering 1-6 - thus missing one date as there are 7 dates in the top row. I noticed I had not taken 1 from the Dates[D] index array, meaning I had only been checking the second dates in the array- hence putting everything in disarray. By adding 1 to the D condition, and negating one from the Dates array, it was now able to reach every column in the table:

Upon testing once more:

	27.02.Tuesday	28.02.Wednesday	01.03.Thursday	02.03.Friday	03.03.Saturday	04.03.Sunday	05.03.Monday
1	Test	Test	Test	Test	Test	Test	Test
2							
3	Test	Test	Test	Test	Test	Test	Test
4							

Now test could reach every column, as well as now being able to represent the end date in a booking - hence every tested problem was now solved.

I now needed to work on what would be written into the boxes, other than just ‘test’.

```
Rich = (RichTextBox)Calendar.GetControlFromPosition(D, XVal);
Rich.Text = OutputList[m].FName;
Rich.BackColor = Color.Green;
```

Using Rich and its functions, I was first able to change the .Text value to the first name of the visitor - associated with the object. Also, I was able to change the .BackColor of the textbox to green. Also, by using a boolean variable called ‘check’, and making it so that it starts as true at the start of an iteration, but changes to false after a ‘single use’, I made it so that only the starting date contains the name of the visitor - making the calendar slightly more professional visually, and similar to the solutions I researched.

```

if (check)
{
    Rich.Text = OutputList[m].FName;
    check = false;
}
ch.BackColor = Color.Green;

```

This shows that the first time 'check' is used to access the if statement, it does the naming of the cell and then changes check to false - making it unable to rename a cell until it is reset to true at the start of an iteration.

	20.02.Tuesday	21.02.Wednesday	22.02.Thursday	23.02.Friday	24.02.Saturday	25.02.Sunday	26.02.Monday
1							
2	Johnny	Malcolm					
3		John					
4							
5			Pegg				

However, later on I realized that the green colouring was not being cleared through 'tableclear()', only the text values were being cleared. Hence, I added another lines to the clear iteration loop to ensure every cell was being reset fully.

```

Rich = (RichTextBox)Calendar.GetControlFromPosition(x, y);
Rich.Text = "";
Rich.BackColor = SystemColors.Control;

```

Rich.BackColor = SystemColors.Control; simply resets the chosen 'Rich' cell back to the system colour 'control', which is the default colour.

This end result looks quite similar to the original plan, and indicates that the calendars visual functionality is quite nearly complete.

The next stage was to be able to interact with the cells of the calendar.

```

Point GetRowColIndex(TableLayoutPanel tlp, Point point)
{
    int w = tlp.Width;
    int h = tlp.Height;
    int[] widths = tlp.GetColumnWidths();

    int i;
    for (i = widths.Length - 1; i >= 0 && point.X < w; i--)
        w -= widths[i];
    int col = i + 1;

    int[] heights = tlp.GetRowHeights();
    for (i = heights.Length - 1; i >= 0 && point.Y < h; i--)
        h -= heights[i];

    int row = i + 1;

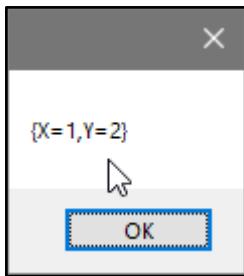
    return new Point(col, row);
}
private void Calendar_Click(object sender, EventArgs e)
{
    Point cellPos = GetRowColIndex(Calendar,Calendar.PointToClient(Cursor.Position));
    cellPos.X
}

```

This following code allows the program to have a direct output to the user clicking a cell in a table. Not only will it detect the click, but it will retrieve the exact cell which was retrieved.

It uses a function which returns a ‘Point’ data type - which stores two coordinates for a 2D plane. It takes a table (the calendar in this case) and a point (where the mouse’s position is) as parameters. The point of this function, is that it then breaks down the mouse’s coordinates as it clicks the table. It then decides which exact cell the mouse pointer was on as it clicked the table. It does this by retrieving the width and height of the table passed to it, as well as breaking it down by how many rows and columns it has. From working this out, and iterating through each width and row slice, and comparing it to the position of the mouse click, it can then pinpoint the specific cell in which the click occurred.

Calling on this function with the supposed parameters, and returning the result point to ‘cellPos’ allows the program to now handle this click. For example, using messagebox.Show(Convert.ToString(cellPos)) (because it is a point type and must be convert) returns something like this:



With the program now able to retrieve this information, the specific bookings and rooms can now be displayed and interacted with - also allowing the user to delete certain rooms and bookings. I feel that using a switch statement would best suit how I would let the user respond using a message box. Creating a whole other form to display the selected information would be slightly too much I think, as there is a simpler solution.

```
public class Rooms:Booking
{
    private string _RoomSize;
    private string _KeyCard;
    private string _Notes;
    private bool _DBAccess;

    public string Notes { get => _Notes; set => _Notes = value; }
    public bool DBAccess { get => _DBAccess; set => _DBAccess = value; }
    public string KeyCard { get => _KeyCard; set => _KeyCard = value; }
    public string RoomSize { get => _RoomSize; set => _RoomSize = value; }
}
```

First, it required that I had an object which could hold rooms, as I would need to store all the rooms from the database. As many of the room attributes were already part of the booking class, I inherited the booking class as part of the 'rooms' class - though I added a few attributes missed out by the booking class.

```

private List<Room> RoomGrab()
{
    var OutputList = new List<Room>();
    using (SqlConnection DB = new SqlConnection(LinkString()))
    using (SqlCommand Comm = new SqlCommand("SELECT RoomID AS ID, RoomNumber, RoomSize, Price, Notes, KeycardNo FROM Rooms", DB))
    {
        DB.Open();
        using (SqlDataReader reader = Comm.ExecuteReader())
        {
            if (reader.HasRows)
            {
                while (reader.Read())
                {
                    OutputList.Add(new Room
                    {
                        ID = reader.GetInt32(reader.GetOrdinal("ID")),
                        RoomNumber = reader.GetString(reader.GetOrdinal("RoomNumber")),
                        RoomSize = reader.GetString(reader.GetOrdinal("RoomSize")),
                        Notes = reader.GetString(reader.GetOrdinal("Notes")),
                        KeyCard = reader.GetString(reader.GetOrdinal("KeycardNo")),
                        Price = reader.GetDouble(reader.GetOrdinal("Price"))
                    });
                }
            }
        }
    }
    return OutputList;
}

```

I then implement a function which, when called upon, will return a list of every room in the database. It retrieves this list using the same method I used to retrieve all the bookings.

```

var RoomList = RoomGrab();

if (RoomList.Count > 0)
{
    int b = 0;
    int c = 1;

    foreach (Room room in RoomList)
    {
        if (b <= 14 + iv && b >= iv)
        {
            ((RichTextBox)Calendar.GetControlFromPosition(0, c)).Text = room.RoomNumber;
            c++;
        }
        b++;
    }
}

```

As I now had this function for listing every room, I decided to change the method in how the table populates the first column with rooms.

Previously, it used the reader to loop whilst it filled the table directly from the database. However, this new method takes the list from the RoomGrab() function, and iterates through each room object's room number attribute, and populates the column that way instead.

Using an if statement embedded with a switch statement, I could handle a message box to simply display the room's specifications, and offer an option to delete the room:

```

if (cellPos.X == 0 && Calendar.GetControlFromPosition(cellPos.X, cellPos.Y).Text != "") {
    Room Obj = OutputList[cellPos.Y + (iv - 1)];
    switch (MessageBox.Show("\r\nRoom: " + Obj.RoomNumber + "\r\nPrice: " + Obj.Price + "\r\nSize: " + Obj.RoomSize + "\r\nNotes: " + Obj.Notes))
    {
        case DialogResult.Yes:
            break;
        case DialogResult.No:
            break;
    }
}

```

I had to ensure that I had two types of message boxes appear, and that the system knew how to recognize either one. The first type is a ‘room’ type, where the user clicks a cell which contains a room.

To limit a user’s click to this, the if statement *only* runs if cellPos.X (the x value of the returned point) is 0. This would mean the selection is in the first column of the table - meaning it is in the only position the rooms are stored. Also, selecting a blank one of these first column cells should be recognized as not a true room cell - hence the if statement checks if the control from position ‘cellPos’ is empty (“”) or not. If it is not empty, and within the first column, it will know this cell represents a room. By then assigning the Room object represented by the cell to variable ‘Obj’, setting it equal to `OutputList[cellPos.Y + (iv-1)]`; I begin at cellPos.Y as this is a good starting place to find the index of the room being represented. I then add iv, to account for any change due to user navigation, and as well I take 1 away - as the cellPos.Y will be 1+ the represented value, as the table skips the first (0,0) cell as it is the corner. This should then provide the index of the room in the list of rooms retrieved from the database.

I then write the content of the message box using the attributes of ‘Obj’. Using string manipulation, I can set out each attribute to an individual line using “\r\n”, as this is recognized to start a new line within the string body.

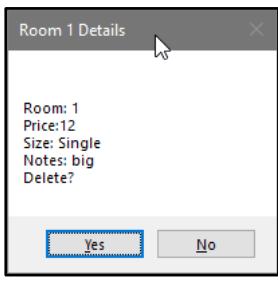
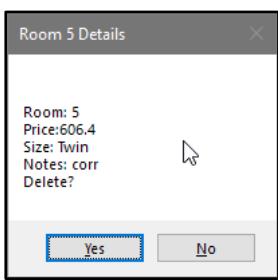
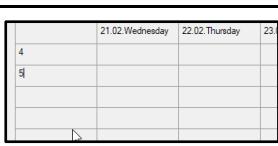
Then the switch statement kicks in, whereas the user has a choice, where the result is either they click ‘Yes’ to delete the room, or ‘No’ to return to the home screen with no change and no deletion, and the switch statement suddenly breaks back to the home screen.

```

case DialogResult.Yes:
    using (SqlConnection DB = new SqlConnection(LinkString()))
    using (SqlCommand Comm = new SqlCommand("DELETE FROM Rooms WHERE RoomID=''' + Obj.ID + '''", DB))
    {
        DB.Open();
        Comm.ExecuteNonQuery();
        DB.Close();
    }
    break;
}

```

In the case the user says yes to deleting the room, the program will create a connection with the database using the `LinkString()` function. It then has the command “`DELETE FROM Rooms WHERE RoomID=’Obj.ID’`”. This means that it will open the rooms database, and delete the record where the RoomID is equal to Obj.ID, which is the object being seen in the message box. It is important that RoomID is the comparison, as this is the only attribute in the record definite to be unique - therefore there will be no accidental deletions of records. I then tested this specific room deletion:

Input	Expected Output	Output	Evaluation
5 rooms with differing sizes, prices, and notes. Then clicking on their individual cells.	Each click displays a message box showing the room's properties correctly, and in the correct ‘line by line’ format.		All the rooms information successfully displayed, however, the price did not show a price tag - failing the formatting test. This could easily be added via string manipulation.
With the same 5 rooms, navigate the calendar 3 spaces upwards, and click on the two remaining rooms.	The 2 clicks bring up a message box displaying the rooms information - with no change due to the navigation.		The information successfully displayed despite being moved around - meaning the indexing worked correctly and requires no further action.
Pressing ‘yes’ on deleting one of the remaining displayed	The room is deleted from the database, and no longer shown		As shown, the room is deleted from the database,

rooms.	on the calendar after the message box shuts down.		however, it is not strictly removed from the calendar unless refreshed. Therefore, this could be improved by calling 'CalUpdate' after the deletion is complete.
Clicking on the a blank cell	Nothing occurs.		Nothing successfully occurred. No further action required.

To remediate on these tests, I fixed the formatting error by adding a '£' symbol to the string before the price is expected:

```
"\r\nPrice: £" + Obj.Price + "\r\n"
```

As well, I added that CalUpdate() is called at the end of either switch statements, meaning the calendar will always be updated after a deletion.

The next switch statement concerns bookings, and can be limited to simply if the cell is green or not, as all bookings so far turn the colour green.

```
else if (Calendar.GetControlFromPosition(cellPos.X, cellPos.Y).BackColor == Color.Green)
```

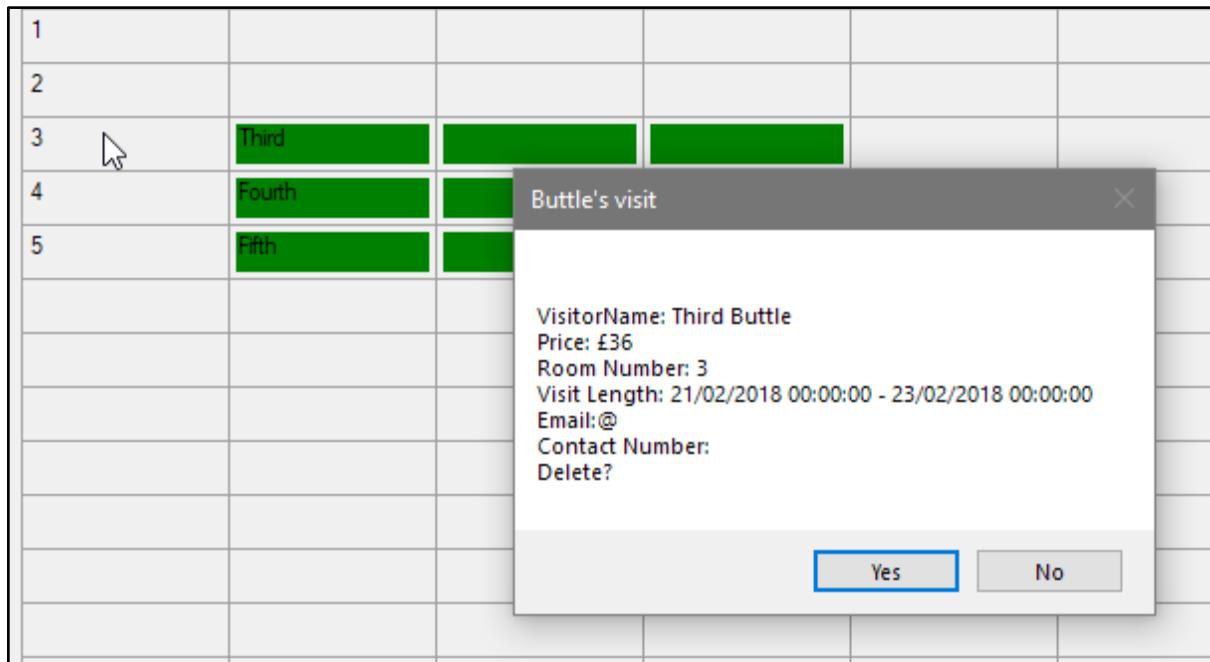
Hence, the other switch statement knows it is dealing with a booking if the cell being clicked is the colour green. The next issue was then retrieving the booking from the booking list, as it would not be as simple as retrieving a room as they were displayed in a 1D plane, not 2D. Also, multiple cells may be referring to a single booking.

```

else if (Calendar.GetControlFromPosition(cellPos.X, cellPos.Y).BackColor == Color.Green)
{
    Booking Book=BookList[0];
    for (int i = 0; i < BookList.Count; i++)
    {
        if(DateTime.Compare(BookList[i].VStart, Dates[(cellPos.Y)-1]) <= 0 && DateTime.Compare(BookList[i].VEnd, Dates[(cellPos.Y) - 1]) >= 0)
        {
            Book = BookList[i];
            break;
        }
    }
    switch (MessageBox.Show("\r\nVisitorName: " + Book.FName+ " " + Book.SName + "\r\nPrice: £" + Book.Price + "\r\nRoom Number: " + Book.RoomNumber + "\r\nVisit Length: " + Book.VStar
    {
        case DialogResult.Yes:
            using (SqlConnection DB = new SqlConnection(LinkString()))
            using (SqlCommand Comm = new SqlCommand("DELETE FROM Bookings WHERE BookingID='"+ Book.ID + "'", DB))
            {
                DB.Open();
                Comm.ExecuteNonQuery();
                DB.Close();
            }
            break;
    }
}

```

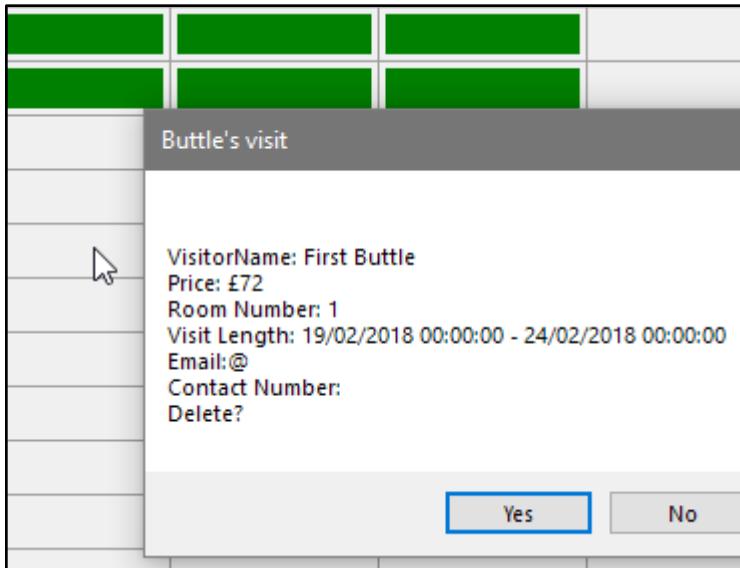
The booking switch uses the same if statement as before, to find if the specific cell is in a booking between two certain dates. Using a for statement, it iterates from i=0 to i=the length of the list of books - 1. By iterating through the entire list of bookings, it can compare the specific dates of the bookings to the date in which the user clicked on (the Y axis). It then breaks when it finds the object with start and end dates that match the date selected. Once done, it sets the variable 'Book' to the booklist[i] which met this condition.



Upon testing this and selecting a booking, I found that no matter which booking is selected - the uppermost booking was always the first to be selected. Either that, or the first booking to be placed into the system. I thought that maybe this was an issue with setting 'Book' to the first value in BookList as a default whilst it searched the database, meaning the 'if' statement was not always selecting a correct value.

Dates[cellPos.Y]) >= 0)

From this testing, I realized that I had been taking 1 from the dates array when comparing when I used the Y Coordinate from the click point. However, as these coordinates begin at 0 - there is no scope error when communicating, therefore I retracted this '-1' function - which may have been totally missing the booking conditions and using the default '0' index. I tested once more with multiple bookings:



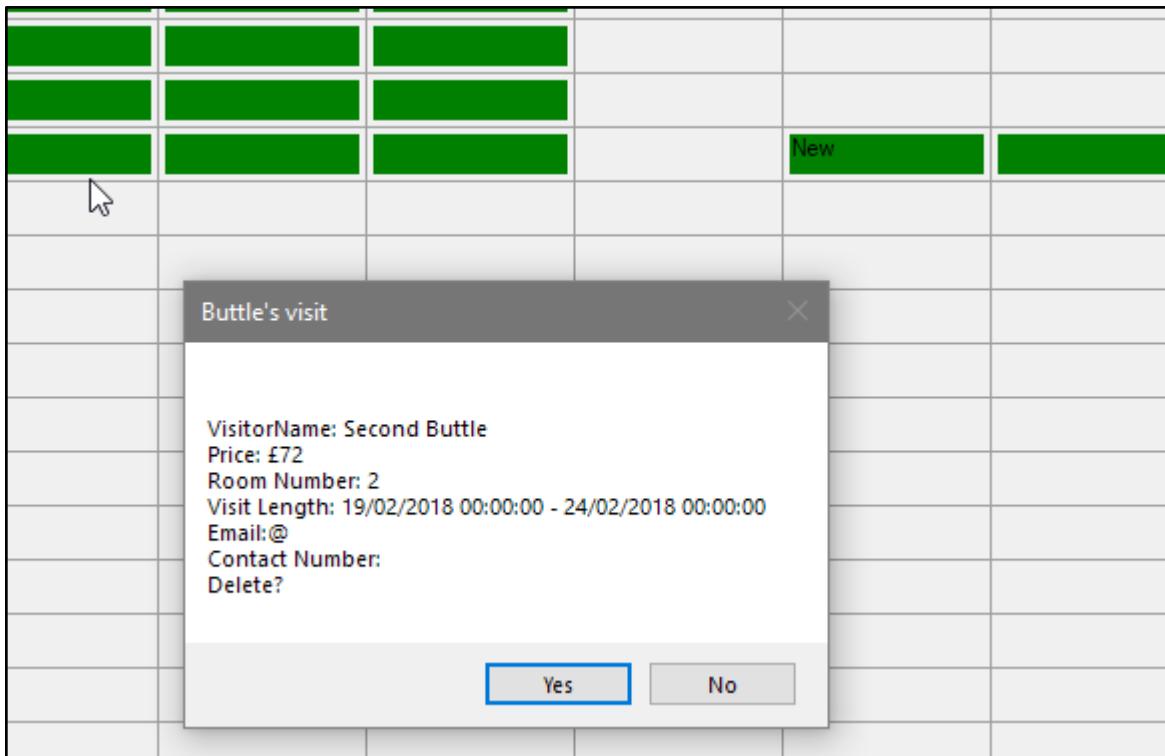
Once again, it only displayed the first booking of the room.

However, after using this testing to white box test the algorithm, I found that I had missed a logical condition in the if statement:

```
&&BookList[i].RoomNumber == Calendar.GetControlFromPosition(0, cellPos.Y).Text|
```

Whilst the condition checked if the cell selected had a relevant time in *any* booking, it did not check if the room number of the x Value on the table matched the object's room number. Therefore, it just went down the list of bookings and the first booking with a matching date simply worked and broke the for loop - hence the first one always worked. Hence the above condition fixed this problem, by comparing the object's .roomnumber and only working if it is directly equal to the text value in the rich textbox within the table position (0,cellPos.Y) (the first column of the table which holds the room numbers). This will not produce an error, as only green cells can activate this function, and green cells only lie on rows with room numbers.

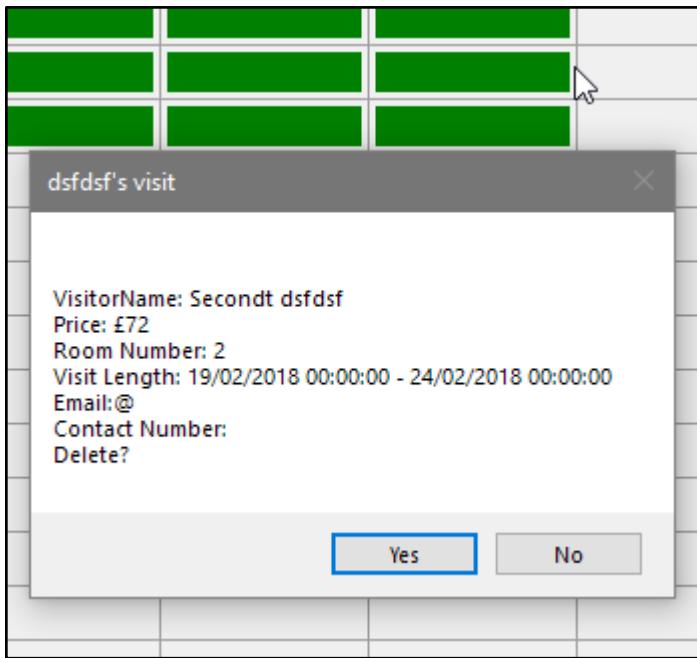
I tested once more:



This time, it did not display the first index of the list for all selections - however for very specific bookings it still displayed the first value in the list.

```
Dates[cellPos.X]) <= 0 && DateTim
```

Upon reflecting on the stepping on the tests, I noticed I had accidentally been using the Y value of the click to measure the date length, *not* the x value where the dates reside upon. By changing the index in 'dates' to the cellPos.X, this was likely to have fixed the problem - and would explain the quite confusing results from the testing.

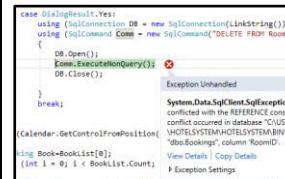


Upon retesting, it appeared that most cells worked correctly, except for the last cells in the booking - they always gave the default 'first' value in the booking list. It appears that this problem occurred as I had previously removed the '-1' from the index in 'dates'. This is bad and caused this error, as all the dates are out of sync as the dates begin at point 1,0, not 0,0. Hence, this 1 has to be accounted for.

```
s[cellPos.X-1]) <
```

I then tested properly:

Input	Expected Output	Output	Evaluation
5 Rooms with 5 booking, each on different days of the week. Then clicking on each booking.	Each click should bring up a message box, displaying the correct and relevant set of information. Every cell should bring up the same message box.	<p>The screenshot shows a Windows application window with a title bar 'Buttle's visit'. Below the title bar is a grid of 15 cells (3 rows by 5 columns). The last cell in the bottom row is highlighted with a light gray background. A mouse cursor is positioned over this highlighted cell. A modal dialog box is open, titled 'Buttle's visit'. It contains the following information:</p> <ul style="list-style-type: none"> VisitorName: Lewis Buttle Price: £24 Room Number: 1 Visit Length: 21/02/2018 00:00:00 - 22/02/2018 00:00:00 Email:@ Contact Number: Delete? <p>At the bottom of the dialog box are two buttons: 'Yes' (highlighted with a blue border) and 'No'.</p>	All the correct dates are shown, as well as bringing up the correct message box for every cell. Navigation seems complete, and no further action required.

Having two bookings on the same room, at different times in the week. Then clicking the two bookings.	They will both bring up message boxes relevant to the same room number, but different information on the visitor, etc.		Correct splits the two bookings, and clicking on either one brings up their relevant information. No further action required.
Deleting a room whilst it has bookings.	It should not be allowed to be deleted.		An error is displayed - meaning it needs to be validated to ensure this cannot happen to a room.

This testing was useful, as I could now see the progress of the calendar system and to see that it is now functioning to a foundation extent - where all the base features are there aside from a few bugs.

To remedialize from the last test, it appears the sql code cannot process a command which wants to delete valuable information, which is linked to another entity in another table. I went ahead with validating the deletion of the rooms:

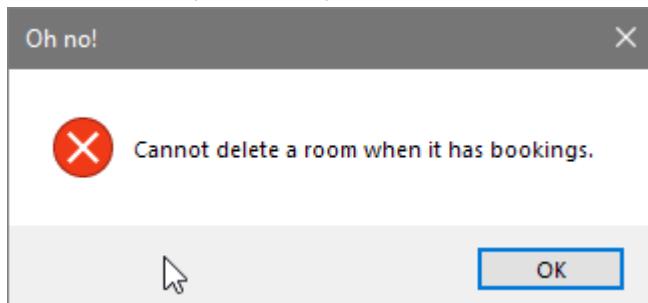
```
for (int n = 0; n < BookList.Count; n++)
{
    if (Obj.ID == BookList[n].RID)
    {
        RGotsBooks = true;
    }
}
if (!RGotsBooks)
{
    using (SqlConnection DB = new SqlConnection())
    {
        DB.Open();
        SqlCommand Comm = new SqlCommand("DELETE FROM Rooms WHERE RID = @RID", DB);
        Comm.Parameters.AddWithValue("@RID", Obj.ID);
        Comm.ExecuteNonQuery();
        DB.Close();
    }
}
```

The above code is ran during the switch statement in case 'yes'. When yes is selected to delete the room, the program then iterates through every index of the Booklist length using a for loop with variable 'n'. Every loop checks if the room being deleted has an ID the same as any booking with the same .RID (Room ID). If at any time there is a match, it will set RGotsBooks to true, to represent the room has at least one booking. It then uses this bool variable to make an 'if statement'

decision, where if the bool value is false (!RGotsBooks), it will run the deletion code in SQL, as it will know the room has no bookings if RGotsBooks is false.

```
else
{
    MessageBox.Show("Cannot delete a room when it has bookings.", "Oh no!", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

However, if this fails, there is an 'else' statement which is ran in any other case, which will simply display another message box alerting the user that they cannot delete the room, as it has bookings. It displays the 'MessageBoxIcon.Error' to indicate something has gone wrong. After the user clicks yes, they are returned to the home menu.



With the main section of the calendar complete, I moved onto the 'mini calendar' which would appear next to the main calendar. This mini one would display the days events of the hotel - such as anyone who would be checking in or out on that day. For initial planning, I expect to just read the booklist to see which bookings have start or end dates the *exact* same as the 'date.now' datetime. It will then store this information in a table using the same richtextbox method as the main calendar, and then allow the user to check off when the visitor has been dealt with or not. The initial plan is as follows:

Visitor Name	Booking In/out?	Status	Note
Kieran Turo	In	<input checked="" type="radio"/>	
Peter Johns	Out	<input type="radio"/>	
John Lemon	Out	<input checked="" type="radio"/>	
Wilson Hundura	In	<input type="radio"/>	
		<input type="radio"/>	

Save  Settings 

Developing from this initial design, I feel adding another column for price would be suitable, so the user may know how much to charge the visitor.

Visitor Name	In/Out	Cost	Notes	Status
				<input type="checkbox"/>

The above is how the table will look on the home screen. It contains a space for the visitor's full name, as well as the status of if they are going into the hotel, or going out. (this can be calculated by the matching of the start or end dates). Cost is also displayed (stored in the booking object). As well, any notes about the booking are displayed in the small textbox. The 'Status' checkbox is for ticking off which visitors have been exited or entered into the hotel. By ticking this box, the booking will then associate that it has been 'entry completed' or 'exit completed', which

will be stored as separate variables in the database, to indicate they have been dealt with.

```
private void MiniUpdate()
{
    List<Booking> BookList = ViewBooks();
    List<Booking> MiniList = new List<Booking>();
    for (int i = 0; i < BookList.Count; i++)
    {
        if (BookList[i].VStart == DateTime.Now.Date)
        {
            MiniList.Add(BookList[i]);
        }
        else if (BookList[i].VEnd == DateTime.Now.Date)
        {
            MiniList.Add(BookList[i]);
        }
    }
}
```

This function will be called at the start of CalUpdate(), and this function's purpose is to update the mini calendar table. It should be quite simpler to code, and may use many reusable components from the calendar. It begins by creating two lists of bookings - one which is simply storing all the bookings in a list using the ViewBooks() method. The other 'MiniList' is for narrowing down this full list, into just the booking which start or end on the same day the visitor is in.

I do this by using a for loop, starting at i=0, to iterate all the way to the size of the full BookList. Within this loop, it checks if BookList[i].Vstart is the same as Now.Date (comparing the start date of the booking to the current date), and an else if statement which does the same but for the VEnd date - the end of the booking. It now filters out a booking to see if it either starts or ends on the current date - and then differentiates the two. Within either if statement, the booking is added to the MiniList to create this list of current event bookings.

```

private void MiniUpdate()
{
    List<Booking> BookList = ViewBooks();
    List<Booking> MiniList = new List<Booking>();
    int c=1;
    for (int i =0; i < BookList.Count; i++)
    {
        if(BookList[i].VStart == DateTime.Now.Date)
        {
            MiniList.Add(BookList[i]);
            ((RichTextBox)tableLayoutPanel1.GetControlFromPosition(0, c)).Text = BookList[i].FName + " " + BookList[i].SName;
            ((RichTextBox)tableLayoutPanel1.GetControlFromPosition(1, c)).Text = "In";
            c = c + 1;
        }
        else if(BookList[i].VEnd == DateTime.Now.Date)
        {
            MiniList.Add(BookList[i]);
            ((RichTextBox)tableLayoutPanel1.GetControlFromPosition(0, c)).Text = BookList[i].FName + " " + BookList[i].SName;
            ((RichTextBox)tableLayoutPanel1.GetControlFromPosition(1, c)).Text = "Out";
            c = c + 1;
        }
    }
}

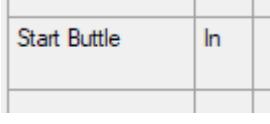
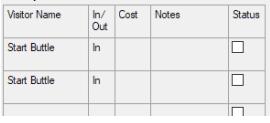
```

Past adding it to this list, it can then further handle the depending result of the booking being on the same date. If the start date is the same, It will write to the corresponding text box of (0,c), where c increments every time a booking is encountered which activates either if statement. It then uses c to move its way down the mini table, to fill it.

The function then fills (0,c) with the full name of the booking visitor, as it knows for certain that the column '0' is the name column. It will continue to do this for each matching booking. Also, depending on which if statement the booking triggered, it will insert either 'in' or 'out' into (1,c) as x=1 is the column for in and out visualization. As this in or out decision is made depending on which booking date matches today's date, that is what makes it requiring of two seperate if statements.

As shown, at the end of either statement, c=c+1 is ran to increment c, to ensure the next row is selected. I then tested this function:

Input	Expected Output	Output	Evaluation
Booking with no start or end date on the current day	The booking is shown in the calendar, but not shown in the mini list		Minilist successfully ignored the booking. No further action required to recognize the bookings.

Booking with a start date on the current date.	The booking's full name is displayed in the mini list as well as "In"		An 'In' booking is successfully recognized and placed within the table. No further action required.
Booking with an end date on the current date.	The booking's full name is displayed in the mini list as well as "Out"		Successfully recognized the 'out' booking, and placed it in the table.
Deleting a booking which is present in the mini list	The mini list updates, removing the relevant booking.		Successfully recognized the booking had been removed, though the table was not cleared as it navigated back up the table - showing a duplicate. Action required.

From this testing, I remediated by running a quick iteration loop to clear the table before it is filled:

```
for(int x = 0; x < 4; x++)
{
    for(int y = 1; y < 8; y++)
    {
        ((RichTextBox)tableLayoutPanel1.GetControlFromPosition(x, y)).Text = "";
    }
}
```

It is an embedded for loop, using x and y variables to navigate through the entire free space of the table - avoiding the first row and last column, as they are important constants. It then retrieves the control, and sets it to blank. I then tested the last test once more:

	Out		
dffdgdfg fdgdgdg	In		

And it successfully moves back up the table, without a duplicate.

I then moved onto the colouring system of the calendar. This means, that the colour of the booking in the calendar will change depending on their status:

- **Green** to show a booking yet to be signed in
- **Amber** to show a booking in progress (Signed in, but not signed out)
- **Red** to show a complete booking.

Status	nchar(10)	<input type="checkbox"/>	'G'

Firstly, it required a new data set in the booking table to maintain recognition of this state. This state can then be changed and stuck to the booking, when required. I also set it to default at the letter 'G' to represent green, as this will be the colour the booking will be before being booked in.

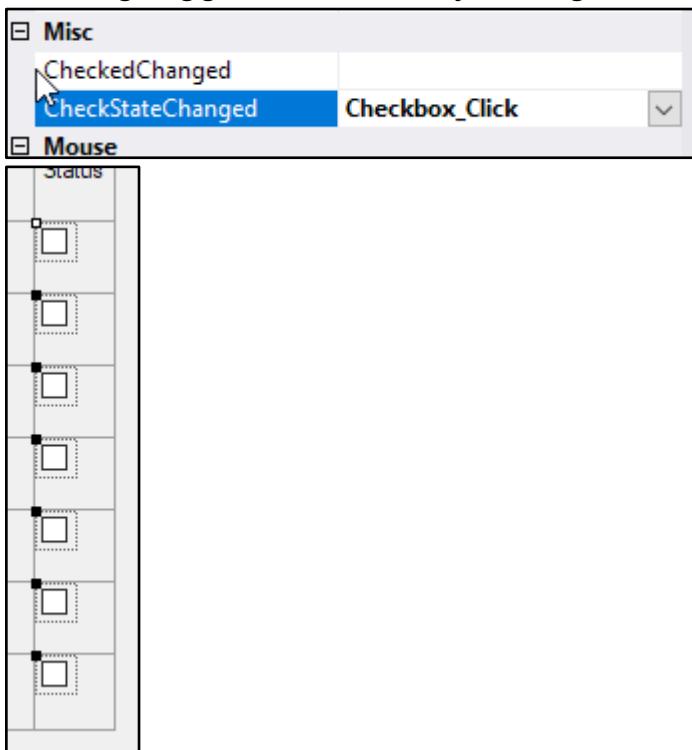
```
for(int y = 1; y < 8; y++)
{
    ((RichTextBox)tableLayoutPanel1.GetControlFromPosition(x, y)).Text = "";
    ((CheckBox)tableLayoutPanel1.GetControlFromPosition(4, y)).Visible = false;
}
```

```
if(BookList[i].VStart == DateTime.Now.Date)
{
    MiniList.Add(BookList[i]);
    ((RichTextBox)tableLayoutPanel1.GetControlFromPosition(0, c)).Text = BookList[i].FName + " " + BookList[i].SName;
    ((RichTextBox)tableLayoutPanel1.GetControlFromPosition(1, c)).Text = "In";
    ((RichTextBox)tableLayoutPanel1.GetControlFromPosition(2, c)).Text = "£" + BookList[i].Price;
    ((RichTextBox)tableLayoutPanel1.GetControlFromPosition(3, c)).Text = BookList[i].Notes;
    ((CheckBox)tableLayoutPanel1.GetControlFromPosition(4, c)).Visible = true;
    c = c + 1;
}
```

Also, to tidy up the validations, I added a method of ensuring only the correct checkboxes would be selected. On the 'clean up' embedded for loop, I added an extra line which iterates through the last column of the table, and sets every checkbox control's visibility to false. Therefore, by default the status column will have no checkboxes. However, once one of the if statements are activated - it will highlight the specific checkbox via (4,c) to then change the checkbox visibility to true. By doing this, it will help make validation easier on the checkboxes - as I now no longer need to handle a checkbox tick which is out of the range of the 'MiniList'.

Visitor Name	In/ Out	Cost	Notes	Status
dsff fdsf	Out	£48		<input type="checkbox"/>

Shown above is how only the checkboxes become visible once a booking triggers the visibility through the if statements.



I then batched all the checkboxes together, and gave them all an event response to if the checkstate changes (CheckStateChanged). It then calls upon the events argument 'Checkbox_Click'. This will be ran if any of the checkbox states are changed whatsoever.

```
for(int y = 1; y < 8; y++)
{
    ((RichTextBox)tableLayoutPanel1.GetControlFromPosition(x, y)).Text = "";
    ((CheckBox)tableLayoutPanel1.GetControlFromPosition(4, y)).Visible = false;
    ((CheckBox)tableLayoutPanel1.GetControlFromPosition(4, y)).Checked = false;
}
```

I then also added a line to the clear table for loops, where it will change the state of all the checkboxes to ‘false’ (not checked) when it iterates through them, to ensure only one checkbox is checked at a time, and that they are all reset every time the MiniList is updated.

```
private void Checkbox_Click(object sender, EventArgs e)
{
    List<Booking> BookList = ViewBooks();
    int val=0;
    for(int i = 1; i < 8; i++)
    {
        if(((CheckBox)tableLayoutPanel1.GetControlFromPosition(4, i)).Checked == true)
        {
            val = i-1;
            break;
        }
    }
    if (((RichTextBox)tableLayoutPanel1.GetControlFromPosition(1, val+1)).Text == "In")
    {
        using (SqlConnection DB = new SqlConnection(LinkString()))
        using (SqlCommand Comm = new SqlCommand("UPDATE Bookings(Status) VALUES ('A') WHERE BookingID='"+MiniListG[val].ID+"'", DB))
        {
            DB.Open();
            Comm.ExecuteNonQuery();
            DB.Close();
        }
    }
    else if (((RichTextBox)tableLayoutPanel1.GetControlFromPosition(1, val + 1)).Text == "Out")
    {
    }
}
```

Upon any of the checkboxes being checked, the above function is ran, due to using the event argument organiser on the controls. As shown, it creates a BookList from Viewbooks() once more, to have the bookings in the context of the code, and sets a default ‘Val’ integer to 0 - which will be used to store a value within an if statement to be ‘carried out’ later on. It then begins a for loop from i=1 to i=7, in the purpose of iterating through every row in the last column (except the first row, as that holds the richtextbox control).

Within every iteration, an if statement condition is checked, where it compares to see if the control from position (4,i).checked == true (asking the question, is this checkbox checked?)

Any loop which enters the if statement, will set the value of val to i-1 (to account for the nature of beginning at 1 for the iteration, as val will be mainly used as an index from now on).

Once the for loop is broken, I then once more make a split of if statements depending on if the booking is an ‘in’ or an ‘out’. If the text in the checked booking is “in”, it will know to want to change the status of the booking from the default ‘G’ to ‘A’. it does this, by using the sql connection from ‘SQLLink()’, and executing an UPDATE SQL command in the table:

UPDATE Bookings(Status) VALUES ('R') WHERE BookingID='MiniListG[val].ID'.

This update command assigns itself to the ‘Bookings’ table and especially the (Status) attribute in the records. It then uses VALUES to show the physical replacement values, in this case would be the single character ‘R’, represented as a string using single speech marks (due to the syntax of SQL). To then locate the specific booking to change, it uses WHERE BookingID=, followed by what the ID should be equal to to then find this record. In this case, it uses the value of ‘MiniListG[val].ID, which is specifically the ID from the booking object being conditioned in the if statements.

```
else if (((RichTextBox)tableLayoutPanel1.GetControlFromPosition(1, val + 1)).Text == "Out")
{
    using (SqlConnection DB = new SqlConnection(LinkString()))
    using (SqlCommand Comm = new SqlCommand("UPDATE Bookings(Status) VALUES ('R') WHERE BookingID='" + MiniListG[val].ID + "'", DB))
    {
        DB.Open();
        Comm.ExecuteNonQuery();
        DB.Close();
    }
}
```

And as shown above, the ‘Out’ if statement is not much different, in that it simply replaces status with ‘R’ to represent RED instead.

```
public DateTime? DateFrom { get; set; }
List<Booking> MiniListG = new List<Booking>();
```

As I could not use the full BookList to find the ID from the mini table, as Minilist stores the order of the minilist bookings, I had to make this minilist public. I did this by creating a public list of bookings called ‘MiniListG’ to stand for global. Then, once minilist has been normally filled - the value of MiniListG is set equal to this local variable to then be available anywhere in the code - until it is changed.

```
MiniListG = MiniListP;
```

Shown above is how the list is made public.

```
s.Y).BackColor != SystemColors.Control)
```

At this point, if I am going to change the colours of the booking cells - I will need to change the identifier I used in the ‘delete booking’ function, where it would only bring up a message box if the cell selected was Color.Green - though the colours can now be amber or red now, so this identifier must change. Hence, I changed the identifier to if the backcolor != (NOT equal) to the default SystemColors.Control - which is the default cell colour. Using this is much less painstaking than setting up 3 AND

statements in the if statement.

```
        }
        if(OutputList[m].Status == "R") { Rich.BackColor = Color.Red; }
    else if (OutputList[m].Status == "G") { Rich.BackColor = Color.Green; }
    else if (OutputList[m].Status == "A") { Rich.BackColor = Color.OrangeRed; }

}
```

Here is the decision, using the object attribute ‘.Status’ to decide which colour to change the richtextbox to. It is simply 3 if statements, two preceding are else if to avoid multiple colour overwrites - and that only the minimum amount of comparisons need to be made. Then, depending on which condition is triggered, it will change Rich.BackColor to the correct colour.

```
Notes = reader.GetString(reader.GetOrdinal("Notes")),
Status = reader.GetString(reader.GetOrdinal("Status"))
```

Also, this required another retrieval and addition to the booking object from the database, shown above being added to the sql command ‘ListBooks()’

I then tested the function so far:

The first problem which occurred was a problem with the sql command, whereas there was a direct error with running the command, displaying that there was an error with a placement of a bracket. From this, I learnt that I was writing the sql command for a batch of values - when I was only changing a single value. Hence, I updated the statement to be more simplified:

```
"UPDATE Bookings SET Status = 'R' WHERE BookingID = "+MinilistG[val].ID, DB))|
```

It simply knows to UPDATE the Booking table, and to then SET the status attribute to ‘R’, WHERE the BookingID is equal to that same ID value from minilist. This simplified statement then allowed the program to run without errors.

The next slight logic error was that the table was not directly updating as soon as a user clicked a checkbox. It required the user to click “Refresh calendar” for colour change to take effect.

```

        Comm.ExecuteNonQuery();
        DB.Close();
    }
    CalUpdate();
}

```

The fix for this was simply that CalUpdate() had to be called at the end of the CheckBox_Click event - which then allowed it to update.

Using stepping, I was able to identify another problem with the code. It seemed that none of the ‘if statements’ for changing the booking colours were being activated. I guessed that the comparison variables were just not matching, so I used stepping to check the status of the variables at the comparison:

```

us == "R") { Rich.BackColor = Color.Red;
at OutputList[m].Status "G" != br.G
status == "A") { Rich.BackColor = Color.Orange;

```

It seemed as though I had accidentally made the data type in the database for status to a fixed character limit, hence the additional spaces after “G”. I fixed this by changing the data type to a variable character length in the database, and updating it:

Through testing, I identified that the minilist rows were not being deleted from the minilist when they were checked, as the rows remained in the table after the calendar updated.

```

== DateTime.Now.Date && BookList[i].Status!="A")
ow.Date && BookList[i].Status != "R")

```

From this testing, I fixed the problem by identifying another condition when the code makes decisions when filling the mini calendar list. As well as checking if the start date is the same date as ‘date.now’, it now checks the status of the booking before writing it to the table.

For example, if a booking is described as ‘in’, and has a status of amber, the program knows it should *not* be displayed in the table as it has been

dealt with. The next time that booking will display in the table is when it is over, and on the last day of the booking.

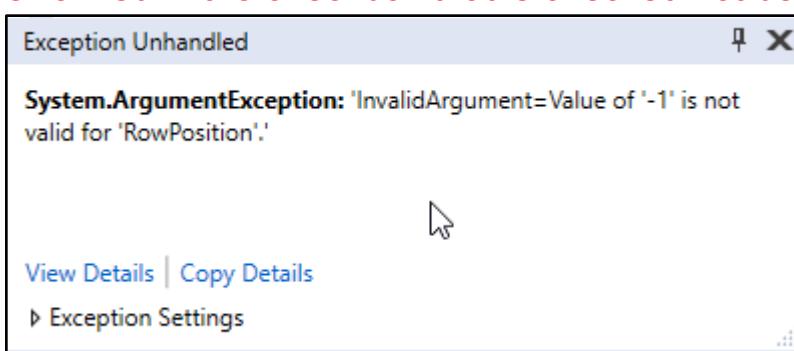
The screenshot shows a software window with two main sections. On the left, there is a guest list table with columns: Visitor Name, In/Out, Cost, Notes, and Status. The guest list contains three entries: Room 1 (Lewis), Room 2 (Lewis), and Room 3 (empty). Above the guest list, the text "Boracay Summer Palace" and the phone number "7647 294010" are displayed. On the right, there is a calendar grid for February 2002. The grid has columns for "22.02.Thursday" and "23.02.Friday". The entry for Room 1 on Thursday is highlighted in red, and the entry for Room 2 on Friday is highlighted in orange. The row for Room 3 is empty.

	22.02.Thursday	23.02.Friday
1	Lewis	
2	Lewis	
3		

Upon testing once more, I tried with two bookings for rooms 1 and 2 - where 1 ended on today and 2 started on today. Upon 'checking' room 2 from the mini list, it then provided the above result. It seemed to colour every preceding booking in the calendar, as well as remove every single booking from the mini list. I imagined this had something to do with the if statements previously tested, and perhaps the checked value was not being identified correctly - hence I tried testing val with a default value of -2 (which should not be passed through in any case, but would cause an error).

```
List<Booking> BookList = ViewBooks();
int val=-2;
for(int i = 1; i < 8; i++)
{
```

Upon running the code, the following error occurred, showing that the error lied in the checkbox that is checked not being identified correctly:



As -1 can only be reached from the default -2.

```
public int val;
```

However, the fix to this problem was simply that val was not being passed and changed successfully with the if statement, hence I made val a public variable - where it would then be able to be changed and recognized anywhere in the code.

I then tested the function of changing the calendar booking colours:

Input	Expected Output	Output	Evaluation																	
Adding a booking which ends on today's date	The booking's name, price and notes are added to the mini list	<table border="1"> <thead> <tr> <th>Visitor Name</th> <th>In/Out</th> <th>Cost</th> <th>Notes</th> <th>Status</th> </tr> </thead> <tbody> <tr> <td>dfdf dfdf</td> <td>In</td> <td>£36</td> <td>Chubs</td> <td><input type="checkbox"/></td> </tr> </tbody> </table>	Visitor Name	In/Out	Cost	Notes	Status	dfdf dfdf	In	£36	Chubs	<input type="checkbox"/>	Successfully recognized the booking, and displayed it in its entirety.							
Visitor Name	In/Out	Cost	Notes	Status																
dfdf dfdf	In	£36	Chubs	<input type="checkbox"/>																
Adding a booking which starts on today's date	The booking's name, price and notes are added to the mini list	<table border="1"> <thead> <tr> <th>Visitor Name</th> <th>In/Out</th> <th>Cost</th> <th>Notes</th> <th>Status</th> </tr> </thead> <tbody> <tr> <td>dfdf dfdf</td> <td>Out</td> <td>£36</td> <td>Chubs</td> <td><input type="checkbox"/></td> </tr> </tbody> </table>	Visitor Name	In/Out	Cost	Notes	Status	dfdf dfdf	Out	£36	Chubs	<input type="checkbox"/>	Successfully displayed all the attributes of the second booking, as well as displaying the check box.							
Visitor Name	In/Out	Cost	Notes	Status																
dfdf dfdf	Out	£36	Chubs	<input type="checkbox"/>																
Checking off the first booking	The entirety of the booking in the calendar turns orange, and the first booking is removed from the mini list.	<table border="1"> <thead> <tr> <th>Visitor Name</th> <th>In/Out</th> <th>Cost</th> <th>Notes</th> <th>Status</th> </tr> </thead> <tbody> <tr> <td>dfdf dfdf</td> <td>Out</td> <td>£36</td> <td>Chubs</td> <td><input type="checkbox"/></td> </tr> </tbody> </table> <table border="1"> <tr> <td>dfdf</td> <td style="background-color: orange;"></td> <td style="background-color: orange;"></td> <td style="background-color: orange;"></td> </tr> </table>	Visitor Name	In/Out	Cost	Notes	Status	dfdf dfdf	Out	£36	Chubs	<input type="checkbox"/>	dfdf				Successfully removed the line from mini list, replacing it with the second booking instantly. Booking then changed to amber.			
Visitor Name	In/Out	Cost	Notes	Status																
dfdf dfdf	Out	£36	Chubs	<input type="checkbox"/>																
dfdf																				
Subsequently checking off the second booking	The entirety of the booking in the calendar turns red, and the booking is removed from the mini list.	<table border="1"> <thead> <tr> <th colspan="5">boracay Summer Palace 7047 Z34010</th> </tr> <tr> <th>Visitor Name</th> <th>In/Out</th> <th>Cost</th> <th>Notes</th> <th>Status</th> </tr> </thead> <tbody> <tr> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> </tbody> </table> <table border="1"> <tr> <td>12</td> <td style="background-color: red;">dfdf</td> </tr> </table>	boracay Summer Palace 7047 Z34010					Visitor Name	In/Out	Cost	Notes	Status						12	dfdf	Successfully removed the line from mini list, leaving a blank table instantly. The booking in the table turns red.
boracay Summer Palace 7047 Z34010																				
Visitor Name	In/Out	Cost	Notes	Status																
12	dfdf																			

At this last stage of successful testing, I felt as though the entire calendar function was complete. I then moved onto implementing the settings button - to bring development to a close.

Key Stage 5 - Settings

For settings, I felt as though the majority of settings features were already implemented within the ‘initial setup’ form, though lacking the option to choose a colour for the background of the program.

I planned to implement this colour feature into the initial setup, and then subsequently copy the initial setup form and design it for use as a settings screen.

I began by using the ‘color dialog’ tool and adding it to the initial setup. Like the file dialog, it has no physical appearance until called upon - where it will display a UI for selecting a colour.



In the initial setup, this is how the bar will look. It is a blank, read only textbox which can subsequently have its color changed. Pressing the right button will begin the dialog selection of colour.

```
private void button3_Click(object sender, EventArgs e)
{
    DialogResult result = colorDialog1.ShowDialog();
}
```

As shown using the even argument from clicking the button, I create a variable ‘result’ from data type DialogResult, which stores the return from the ShowDialog return of colorDialog1 - which is the instance class of the colour dialog.

```
private void button3_Click(object sender, EventArgs e)
{
    DialogResult result = colorDialog1.ShowDialog();
    if (result == DialogResult.OK)
    {
        Color SysColour = colorDialog1.Color;
        textBox1.BackColor = SysColour;
    }
}
```

Using result not as the actual result of the dialog, but as a running instance of the dialog, I can then extract the user’s choice using ‘colorDialog1.Color’. I use an if statement to see the result of the dialog, where the result may be ‘OK’, being the OK button displayed in the dialog. If this button is pressed, the colour changes are then approved, and SysColour (of class Color) then becomes equal to the return of this dialog. I then change the textbox.backcolor colour to that same SysColour.

```
public Color SysColour;
```

I then make a public variable of SysColour, allowing the program to access it anywhere in the Initial Form solution.

```
    writer.WriteLine(contactno.Text);
    writer.WriteLine(pictureBox1.ImageLocation);
    writer.WriteLine(SysColour.ToString());
}
HomeCall();
```

Having it public now allows the program to write this colour to the .txt file which stores all the other settings information. Now I know that the 4th line of the file will contain the information for the system colour. With this complete, I moved onto other forms recognizing this colour, and reading it from the file.

The first problem I realized, is that form.Backcolour rejects transparent colours, hence I will need to change the form's setting when the form is first loaded to allow transparent colours.

```
private void Form2_Load(object sender, EventArgs e)
{
    BookingForm BookingForm = new BookingForm();
    this.SetStyle(ControlStyles.SupportsTransparentBackColor, true);
    ...
```

As shown, I used `.SetStyle(ControlStyles.SupportsTransparentBackColor, true)`, in which I used the `SetStyle` function with the two parameters to change the boolean value of the transparency property.

```
CoverPic.ImageLocation = reader.ReadLine();
this.BackColor = Color.FromArgb(Convert.ToInt32(reader.ReadLine()));
```

Then, within the loading of the form, it can then read from the text file to retrieve the string of the colour. As the colour is represented by integer data (ARGB), it must first be convert from string, to int32, to then as an ARGB value to be recognize and converted into a colour.

```
private void Form2_Load(object sender,
{
    Form1.send = 1;
    BookingForm BookingForm = new Book
```

As you can either enter Form1 (settings setup) from two ways, from the main menu or as a first startup, the program needs to differentiate the two kinds of entrances. Hence, I had a public variable called 'Send' which would be called upon the program loading, and begin at 0. Then, if The home screen is loader, it will be turned into 1. I can then know, if send is 0, it is a first time loading the forms, or if it is 1, it will simply just

need to reload the homescreen form, as it has already been instantiated. This saves multiple home screens being made at once.

```
private void Form1_Load(object sender, EventArgs e)
{
    if (System.IO.File.Exists("C:\\Program Files\\HotelSystem\\config.txt") && send == 0)
    {
        HomeCall();
    }
    this.FormClosing += new FormClosingEventHandler(BookingForm_Closing);
}
```

It is important now that HomeCall() is only called when Send==0, or else it will create duplicates of the home screen, which would cause issues if changes are being made to the home screen via the settings menu. As there would be issues between forms needing to load a different home form, when settings is making changes and creating a new home screen to update it, I felt it would be easier and safer to simply exit the program restart the program, where it would create one single home screen from the new read variables.

```
if (send == 1)
{
    MessageBox.Show("Application must restart to have effective changes.");
    Application.Exit();
}
else { HomeCall(); }
```

Hence on pressing submit on the settings menu, if 'send == 1' (if the user accesses settings from the home screen) they will be greeted with a message box alerting them to the restart of the program. It will then use Application.Exit() to close every instance of the program.

Otherwise, if send != 1 it will know to simply call HomeCall() for the first instance of the home screen.

```
using (System.IO.StreamReader reader = new System.IO.StreamReader("C:\\Program Files\\HotelSystem\\config.txt"))
{
    reader.ReadLine();
    reader.ReadLine();
    reader.ReadLine();
    this.BackColor = Color.FromArgb(Convert.ToInt32(reader.ReadLine()));
}
```

For every form to then load the chosen user colour, I had to implement the same read function into every "Form_Load" event, so that the colour is changed permanently once the form is loaded for the first time.

With the settings now completely working, it seemed as though every planned part of the development had been complete, and some advanced features had been implemented. Due to time and economical restraints, the RFID scanning advanced feature was unfortunately

unable to be implemented, however, every basic feature had seemingly been achieved, and was ready to be evaluated.

Evaluation

Post Development Testing

Feature Testing & Testing for Robustness

Once development concluded, I moved straight on to post development testing, in which I am using the first plan of the end test plan to finally conclude which parts of the program met the initial success criteria. As this test plan was made pre-development, it will be unbiased testing as to prove critical analysis of the final state of the program.

Once again, I used a colour scheme of green, amber and red to indicate how successful the test was, as well as a small analysis of the output.

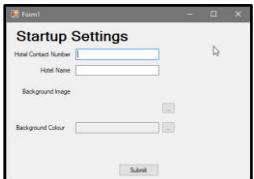
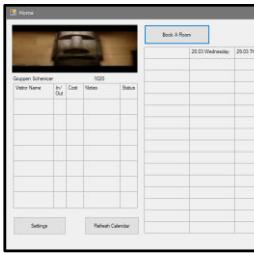
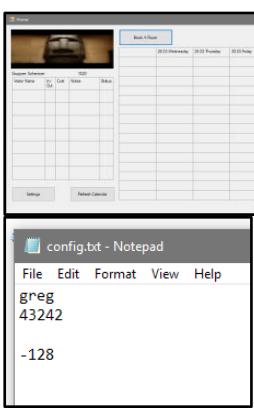
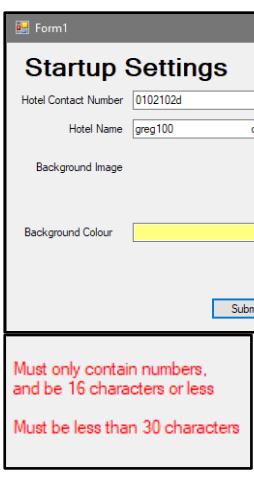
FUNDAMENTAL

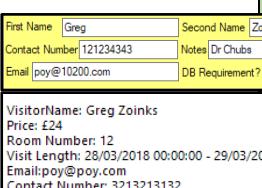
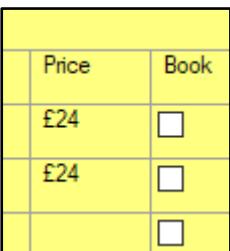
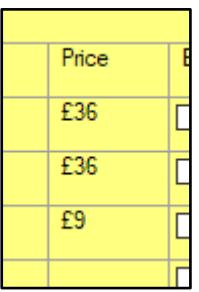
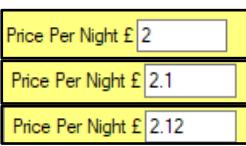
- Initial Setup
- Ability to input room and customer information and indicate a price
- Ability to input room and customer information from a range of different input devices
- A Graphical User Interface
- Calendar Display
- Saving and Restoring Data

ADVANCED

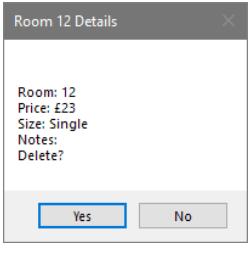
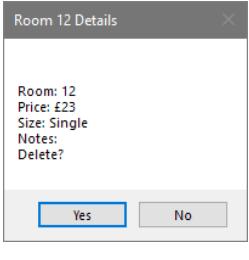
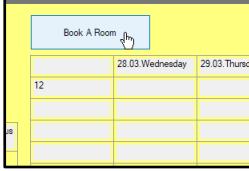
- Keycard Scanning and unlocking upon validation
- Safeguarding (Storing Unlock Attempts)
- Responsive ‘Beeping’
- Colour scheme organization
- Search Function

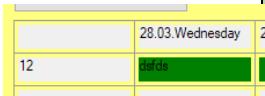
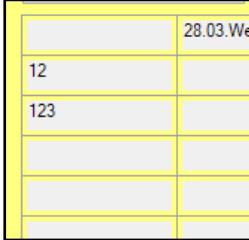
<u>Input</u>	<u>Expected Output</u>	<u>Output</u>	<u>Analysis</u>

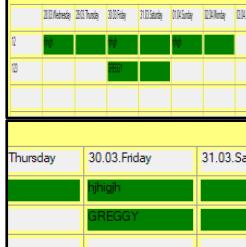
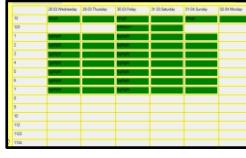
<p>Running the program for the first time and reaching the home screen.</p>	<p>The program notices it is the first time running, and displays the initial setup form.</p>		<p>Successfully recognized that the program was running for the first time, and brought up the Startup Settings menu instead of the home screen form.</p>
<p>Running the program for a second time, following the previous time.</p>	<p>The program displays the home screen instantly.</p>		<p>Upon running the program once more, it successfully recognized that the program had been ran before and with previously input settings ready to use for the home screen - and it was able to run the home screen straight away with these same settings.</p>
<p>Inputting VALID data into the initial setup form. (greg, 43242, -128 respectively)</p>	<p>The information is submitted to the config file in a specific order, and the home screen opens.</p>		<p>The valid data was submitted to the config file, and stored line by line successfully - as well as bringing the user to the home screen as expected.</p>
<p>Inputting INVALID data into the initial setup form. (001201026, "greg100 errorerrorproblem" respectively)</p>	<p>The initial setup screen remains open, and red validation messages appear relevant to the invalid inputs. No changes are made to the config file.</p>		<p>The validation was successful, in that the appropriate validation messages appeared and halted the progress of the user - to ensure the wrong data was not submitted. The form remained, as expected.</p>

Take 3 peers from my class and input them as visitors to bookings, storing a length of visit, size of required room, their name, phone number, and any special requirements.	The three peers' details are stored as an abstract representation within the database - able to be retrieved and viewed.	 <p>First Name: Greg Second Name: Zoin Contact Number: 121234343 Email: poy@10200.com Notes: Dr Chubs DB Requirement?</p> <p>VisitorName: Greg Zoinks Price: £24 Room Number: 12 Visit Length: 28/03/2018 00:00:00 - 29/03/2018 Email:poy@poy.com Contact Number: 3213213132</p>	Upon entering a visitor's details, all three times led to the visitor being successfully added to the database with all their inputted details as shown, with a designated area for all of the required information.								
Attempt to manually change price of a booking, overriding the calculated price.	The inputted override price is submitted to the database, ignoring the calculated price.	 <table border="1"> <thead> <tr> <th>Price</th> <th>Book</th> </tr> </thead> <tbody> <tr> <td>£24</td> <td><input type="checkbox"/></td> </tr> <tr> <td>£24</td> <td><input type="checkbox"/></td> </tr> <tr> <td>£24</td> <td><input type="checkbox"/></td> </tr> </tbody> </table>	Price	Book	£24	<input type="checkbox"/>	£24	<input type="checkbox"/>	£24	<input type="checkbox"/>	Whilst there is a price system, there is no way of manually changing the price without creating a new room of £0 price.
Price	Book										
£24	<input type="checkbox"/>										
£24	<input type="checkbox"/>										
£24	<input type="checkbox"/>										
Book three rooms for different lengths, and check the booking prices are correct.	Each of the bookings are correctly calculated (Price of room * number of nights stayed)	 <table border="1"> <thead> <tr> <th>Price</th> <th>Book</th> </tr> </thead> <tbody> <tr> <td>£36</td> <td><input type="checkbox"/></td> </tr> <tr> <td>£36</td> <td><input type="checkbox"/></td> </tr> <tr> <td>£9</td> <td><input type="checkbox"/></td> </tr> </tbody> </table>	Price	Book	£36	<input type="checkbox"/>	£36	<input type="checkbox"/>	£9	<input type="checkbox"/>	I tested 3 rooms, two priced at £12 and one at £3 a night. I then made the booking for 3 nights - which gave the correct numbers for all 3 bookings.
Price	Book										
£36	<input type="checkbox"/>										
£36	<input type="checkbox"/>										
£9	<input type="checkbox"/>										
Attempt to input rooms prices for 4	The first 3 rooms are accepted, but	 <table border="1"> <tr> <td>Price Per Night £ 2</td> </tr> <tr> <td>Price Per Night £ 2.1</td> </tr> <tr> <td>Price Per Night £ 2.12</td> </tr> </table>	Price Per Night £ 2	Price Per Night £ 2.1	Price Per Night £ 2.12	For the first 3 valid inputs, all 3 were accepted as shown, and					
Price Per Night £ 2											
Price Per Night £ 2.1											
Price Per Night £ 2.12											

different rooms, each to 4 different decimal places (0,1,2,3)	the 3 decimal place price is denied via validation.		made rooms in the hotel without being halted by the validation. However, the 4th test of 3 decimal places halted the progress and denied the room - displaying the appropriate validation text.
Attempt to book and delete a room using only a mouse	The program is navigable, and a room is able to be booked and deleted successfully.		I was successfully able to delete a room using only a mouse, by using virtual keyboard tools alongside the mouse to ensure the mouse was the only system input.
Attempt to book and delete a room using only a keyboard	The program is navigable, and a room is able to be booked and deleted successfully.		I was able to use the tab controls built into the program to navigate the controls in the forms. As each control has a tab index, meaning each tab press selects the next control index. I was then able to select whichever control I liked - however it may require some optimisation in logical ordering of the tab controls, as the tab indexes seem to jump across the screen in a non-logical way. For example, the tab index for the 'refresh' button at the bottom of the page follow the index for the 'book a room' index, as shown in the image.

Attempt to book and delete a room using only a touchscreen device	The program is navigable, and a room is able to be booked and deleted successfully.		Using a windows surface laptop for the touchscreen, I was able to navigate the program using this input method, with the touchscreen using both a virtual mouse and keyboard.
Attempt to book and delete a room using a puff suck switch and keyboard.	The program is navigable, and a room is able to be booked and deleted successfully.		The program successfully recognized the input device, and worked successfully with a keyboard to help navigate the program - thus showing the program is available to a range of different peripheral devices.
Search to see if more than 3 prominent colours are used for the GUI	3 or more colours are successfully used to differentiate different visual elements in the program.		Upon searching, there were exactly 3 predominant colours in the system- <ul style="list-style-type: none"> - Background colour - Button select(blue) - Text space (Grey) This means that the program uses a significant variance in colour for GUI visualization.
Try and navigate the program through buttons - Is it a GUI?	The program qualifies as a GUI based program		The layout of the program successfully identifies as a GUI, as it uses buttons and virtual space to represent information in a colourful and interactive fashion. This meaning that the feature of being a GUI is a success.

Are images used to help communicate button purpose?	Images are identified in the program for the purpose of aiding navigation		No images were used as originally planned in the final design. This shows that more time probably needed to be put into the final cosmetic touches of the program.
Can bookings be presented in a tabular form?	The bookings are displayed in some sort of table form.		As shown, the bookings are able to be successfully organized in a table form - meaning the feature of having a calendar for visualization was successfully implemented.
Can an individual booking be selected, and interacted with?	A booking can be selected, and viewed.		The user can interact directly with a booking via clicking it, as well as able to delete the booking if necessary. This means the ability to interact with bookings beyond adding them to a database is successfully implemented.
Add a room to the system. Check the calendar's status.	The calendar adds the rooms automatically to the calendar view.		The new room '123' was added straight to the database, and hence the calendar. This shows how responsive the program is to change and inputs, and responds well as planned to change. This makes it a simple and stress-free experience to interact with the program, as listed as an initial feature.

Add 3 bookings and close the program. Reopen the program and access the home screen.	The three bookings should be prevalent in being displayed on the home screen - despite the program resetting		I added 3 bookings and reopened the program, where the same database file was read - meaning the calendar was successfully populated. This completes the integrity of the 'save/load' feature, meaning the program could be properly utilized in a professional environment.
Add 10 rooms and 20 bookings for those rooms, and access the calendar.	The loading speed of the program is not noticeable.		As shown, all the bookings and rooms were placed in the calendar - and no apparent lag occurred. This shows that the calendar is portable for a large array of bookings, and can withhold many bookings at once.
ADVANCED	TESTING		
Can a keycard scanning be used to communicate with the program?	A keycard being scanned can be recognized by the program		Unable to test, as the keycard system was not implemented
Does the scanner have different outputs for a successful	The program recognizes a 'successful' scan and an 'unsuccessful' scan.		Unable to test, as the keycard system was not implemented

and failed scanning?			
Scanning a valid card at a valid time	The program outputs a 'success' reaction and records the time and location of the scanning.		Unable to test, as the keycard system was not implemented
Scanning an unrecognised card	The program outputs a 'unsuccessful' reaction and records the time and location of the scanning.		Unable to test, as the keycard system was not implemented
Scanning a valid card at a valid time	A single 'beep' is emitted		Unable to test, as the keycard system was not implemented
Scanning an invalid card	Two consecutive 'beeps' are made.		Unable to test, as the keycard system was not implemented
Changing the system's colour	The change is recorded, and every form now has that same colour.		As shown, I was able to change the colour of all the forms from red to blue via the settings menu. This shows the forms cosmetic interchangeability.
Setting the colour to something visually obtrusive.	The program counteracts this obtrusive change, to ensure every element is		The obtrusive nature is not counteracted, the program suffers extreme quality loss by not being able to display all the text, as the background

	still viewable.		is the same colour. This would require extra work to ensure this is avoided - such as changing the text colour to white when necessary.
Searching for a room which does not exist.	The program returns no results		The program displays no results, successfully showing the desired consistency of the search function.
Searching for a room where multiple rooms meet the conditions	All the possible rooms are shown.		The program displays both the results, successfully showing the desired consistency of the search function.

I will use this test data to cross reference with the original success criteria and come to my end reflection conclusions - as well as to identify which areas would require improvement during a second development iteration.

End User Testing

I designed an end user survey previously during the design section, which would test how a user interacts with the program, whereas I would measure their competency with navigating through different given tasks - with no prior experience with the program. This would then give data to relate to the success criteria, in which I would test how simple the program is to read, understand, and navigate.

As well as the original tests, I decided to add more questions to the survey to relate to the actual thoughts of the stakeholders about the program, and about how the system would integrate into a real business scenario. I decided to add these tests to gain a better idea of the program's ability to 'embed' - as it lacks the keycard system which would be the best evidence of the program being able to embed, but asking questions to the stakeholder would be the next best chance of gaining an idea of how well this program would be embeddable in a hotel

environment. For the questions which require time recordings, I will go through the questions face to face with the stakeholder.

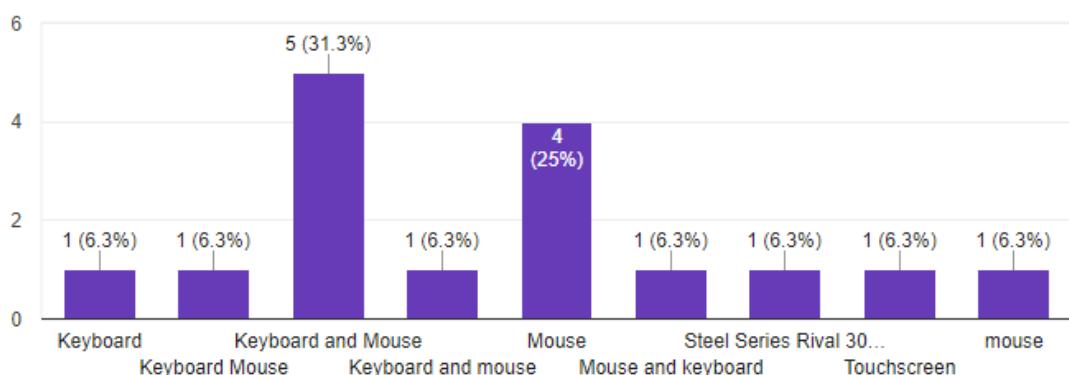
The questions I asked are as follows:

- **What peripheral device did you use to navigate the program?**
- **On a scale of 1-10, how simple was it to navigate the program using your peripheral device?**
- **On a scale of 1-10, how simple was it to find the different features of the program?**
- **How helpful were the colours, in helping you navigate the program?**
- **(Requires set up of a booking, and will record the time in which the user responds) Is room 1 available today?**
- **Were you able to successfully book yourself into a room?**
- **How much more efficient do you feel this solution would be, rather than using a paper based solution?**
- **Could you see yourself using a solution like this in a hotel workplace?**
- **How long do you feel it would take an average member of staff to fully learn the features of this program?**
- **Do you feel any features are irrelevant or unnecessary?**
- **Do you feel there are any features missing, or required?**
- **How well do you think this program would integrate into a workplace environment?**

Once distributing the survey, and surveying the same number of users as originally surveyed (16), I could then reflect on the results:

What peripheral device did you use to navigate the program?

16 responses



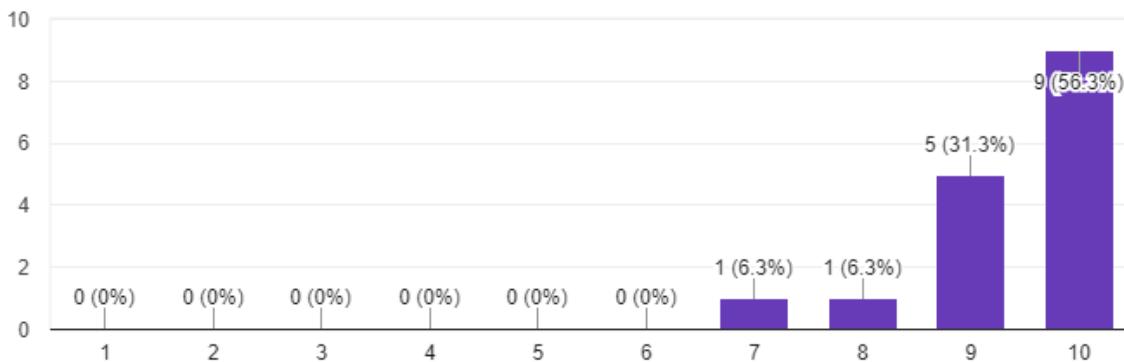
This first question was to get an idea of the range of peripherals being used, to then tie the results of this to the following questions about how simple the program was to navigate using this peripheral device. As the question was an open ended text question, the above results translate to more simpler terms. The mouse and keyboard were used by 15/16 stakeholders, whilst the 16th used a touch screen. Whilst this may not be a good test for other devices, it is a good representation of the devices that the majority of users will be using - the mouse and keyboard.

Hence, if the following tests show that the program handled their peripheral well, it will mean the majority of the demographic will be certain to have good access to the program features - as the results show that the majority of users are using this mouse and keyboard combo. Also, evidence of other devices were already obtained through the black box testing performed above - meaning there is evidence of devices other than the keyboard, mouse and touchscreen.

In all, this test shows that majority of test subjects were using the mouse and keyboard combination, and this is a valid representation of the larger demographic which will use the program.

On a scale of 1 - 10, how simple was it to navigate the program using your peripheral device?

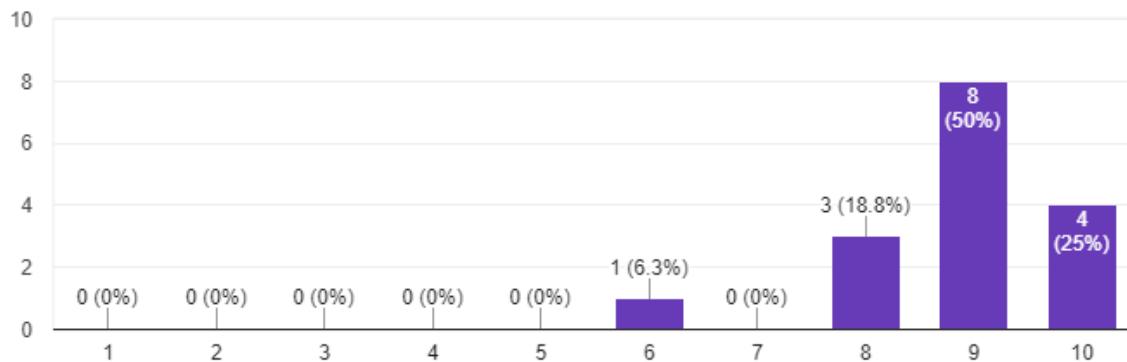
16 responses



The scale of this question ranges from 0 (Extremely difficult) to 10 (Extremely simple). As shown, the results range from 7-10, with majority leaning towards a certain 10. As the results for 7 and 8 are only individual people, whilst the other 14 people chose 9 and 10, it could be considered for the majority of the population to be residing within this bracket, and those individuals who voted for 7 and 8 to be considered anomalous. These two lower results for 7 and 8 may have been produced from an odd hardware issue, nevertheless 7 and 8 are still high values for this rating. Overall, this shows that the majority of users (which happen to be mainly keyboard and mouse users) found the program easy to navigate using their chosen peripheral, arguing that the program was a success in being simple to navigate and learn upon being used for the first time.

On a scale of 1-10, how simple was it to find the different features of the program?

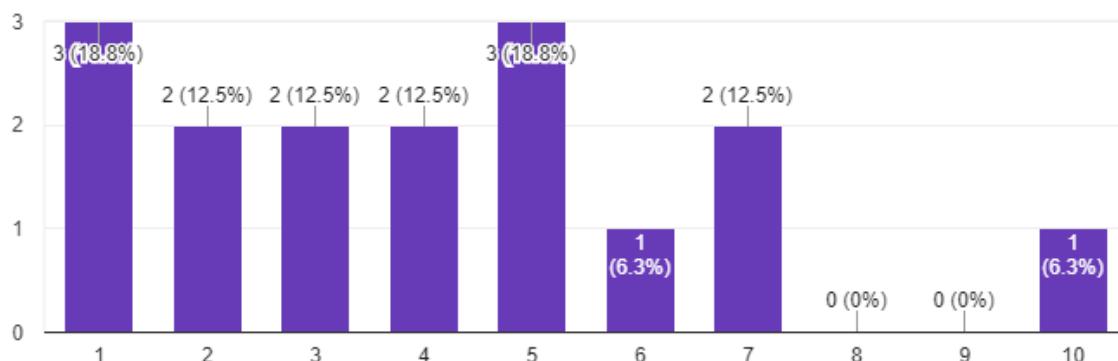
16 responses



This test once again had a clear pattern of users feeling that it was on a high level of simplicity to find different aspects of the program. This aspect scored highly between 8-10, which is quite a consistent range of scores throughout the demographic. This tests the other side of the simplicity of not only the physical navigation of the program (as tested in the previous question) but how anyone can understand the reasoning and desire to navigate to certain areas of the program - where both of these aspects go hand in hand. It is pleasing to see both tests determining that the users felt as though the program was not only simple to navigate, but they could also feel they could access the areas of the program they desired without much confusion. This result shows the program successfully creates an environment using a GUI to present information and pathways to a user - for their ease of communication and navigation.

How helpful were the colours, in helping you navigate the program?

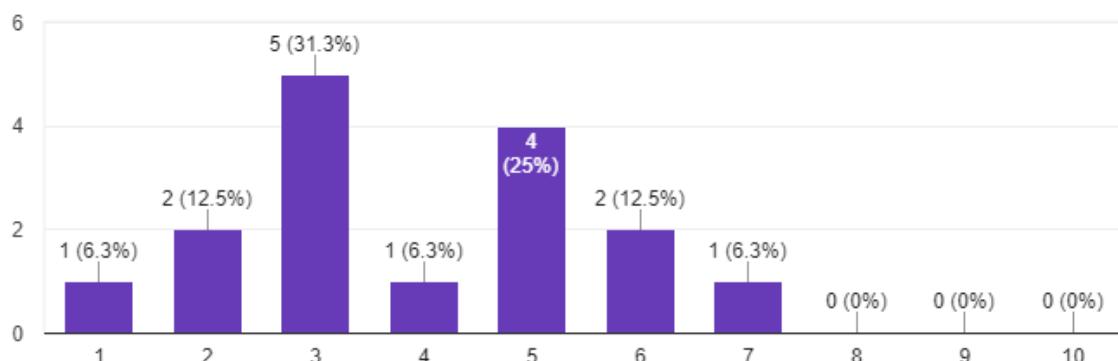
16 responses



The results of this test are quite inconsistent, though edging more towards the users feeling as though the use of colours were not as helpful in their navigation of the program. With 9 results being below 5, and only 4 results being more than 5, it shows that the colours described and used in the program were not helpful for navigating the program, perhaps insinuating that there are larger factors that matter when visually interacting with a GUI - perhaps this research would need to be done in the next iteration of development. With the inconsistency of results, this could mean that the question was too broad or quite unclear, or that the users felt indifferent to the colour being a high factor in their visual cognition when interacting with the software - which was an initial feature and goal of the program. On the topic of how to improve this will be discussed in a later section.

Is Room 1 available today? (Time Result)

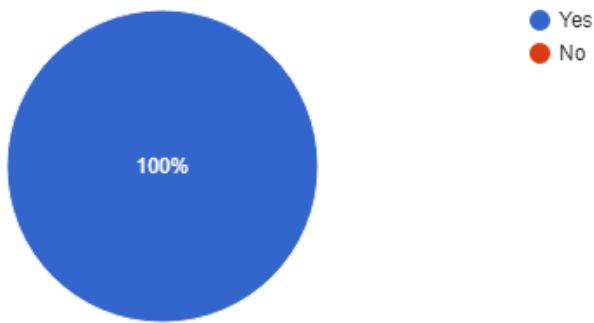
16 responses



With the mean of the results being 3.9, and the supposed ‘upper limit’ of the results being 7, these results show that the calendar is quite readily able to be read, and would not require much to learn for a newcomer to the program. The results are quite normally distributed, with 3-4 being the ‘peak’ of the distribution, meaning this is likely the pattern which would follow if the population was larger. This end result shows that the user takes an average of 3-4 seconds to understand the calendar for the first time, showing that the GUI successfully communicates which information represent what, and that it is simple to come to terms with - which was a basic success criteria. Generally, I would imagine 3-4 seconds to be a reasonable time, especially with results so concise.

Are you able to book yourself into a room?

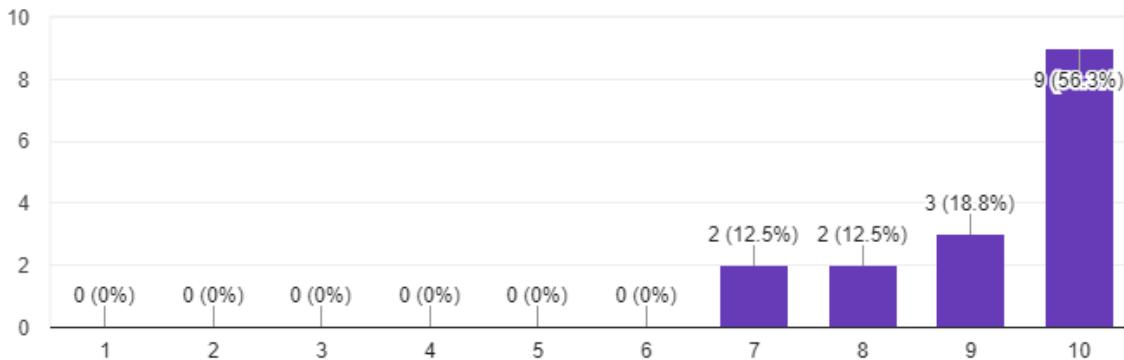
16 responses



Quite definitely, these results show that the program was intuitive enough to allow a user to book a room from their own logic and reasoning. This shows that the program is quite simple to follow for newcomers, and provides its most basic service of booking a room indefinitely to the entire demographic.

How much more efficient do you feel this solution would be, rather than using a paper based solution?

16 responses



I asked a similar question in the stakeholders survey to begin with, but by answering this question again, it allowed me to coincide with the original stakeholder question to find out if my solution actually met the match of being a software which would outmatch a paper based solution - which it originally planned to become. As shown from the results being 7 or above, and with the mode of results falling on the highest 10, it shows that the solution definitely got the approval from the stakeholders that the software could outmatch and be a genuinely better alternative to the paper based solution. The results showed that the users felt that the program I provided would be a better, more robust solution to a paper based solution in a workplace environment, which is exactly the sort of approval which the program set out to achieve.

Could you see yourself using a solution like this in a hotel workplace?

16 responses

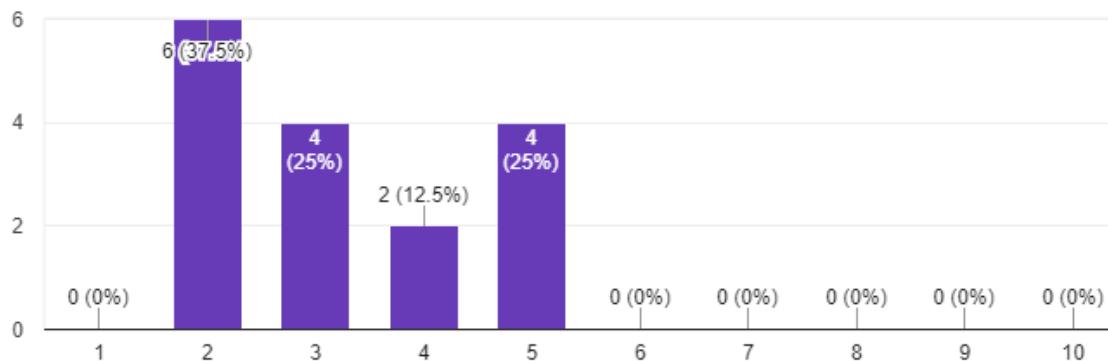


This question makes the connection between being just a basic software, to a solution which could genuinely manifest into a professional

environment. As shown, it reached the maximum approval in showing that the solution would be thought to have a real use in a hotel environment. This result is important, as it shows the stakeholder's trust in the program, especially in that they believe it has the functions and features to be of use in a hotel environment.

How long do you feel it would take an average member of staff to fully learn the features of this program?

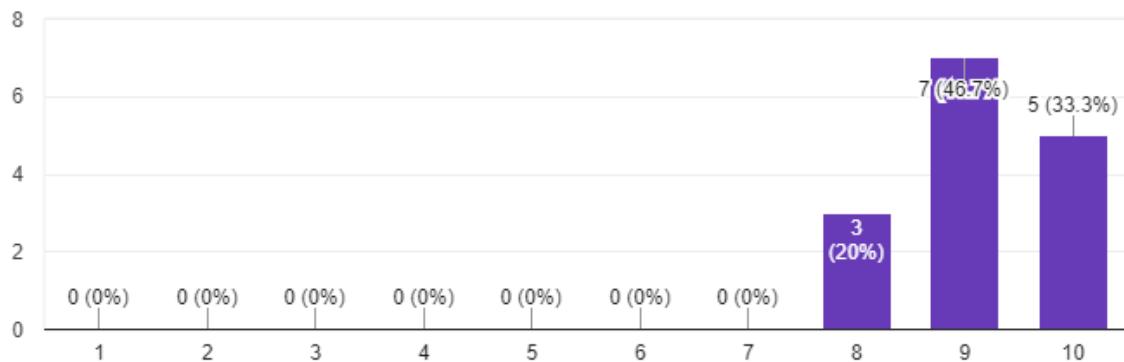
16 responses



With 1 representing 'Instantly' and 10 representing 'A very long time', it appeared that the end users were confident that the software would be relatively easy to learn, which is important in a rapidly developing environment where implementation of this software would need to be quite sudden - as an entire set of staff may need to learn how to use the software at once, so a more simple software the more attractive it is to professionally implement. As some results are tending to 5, this would mean that some are not confident that the program could be learnt quite so easily - hence ideas could be discussed later on for the next iteration of development for what could make the program easier to learn for newcomers, such as a manual or an on screen tutorial.

How well do you think this program would integrate into a workplace environment?

15 responses



For this question, the results were concise between 8,9 and 10. This shows a firm belief from the stakeholders that this program would be able to integrate well into the already existing professional scene of hotel room booking.

Do you feel any features are irrelevant or unnecessary? If so, what are they?

12 responses

None (4)

No (2)

no

The background image

none

Transparent background colours

Some check boxes were transparent.

I liked all the features and thought they were all handy

Whilst a majority of the stakeholders felt that there were no features irrelevant or unnecessary, there were a few points made about some of the program's features, or certain parts of the program which the users had a problem with - or felt were unnecessary. As many felt that there were no unnecessary features, it shows that the users felt every step taken during development was relevant, and that the final product is well bespoken to its final goal.

The features mentioned:

- The background image

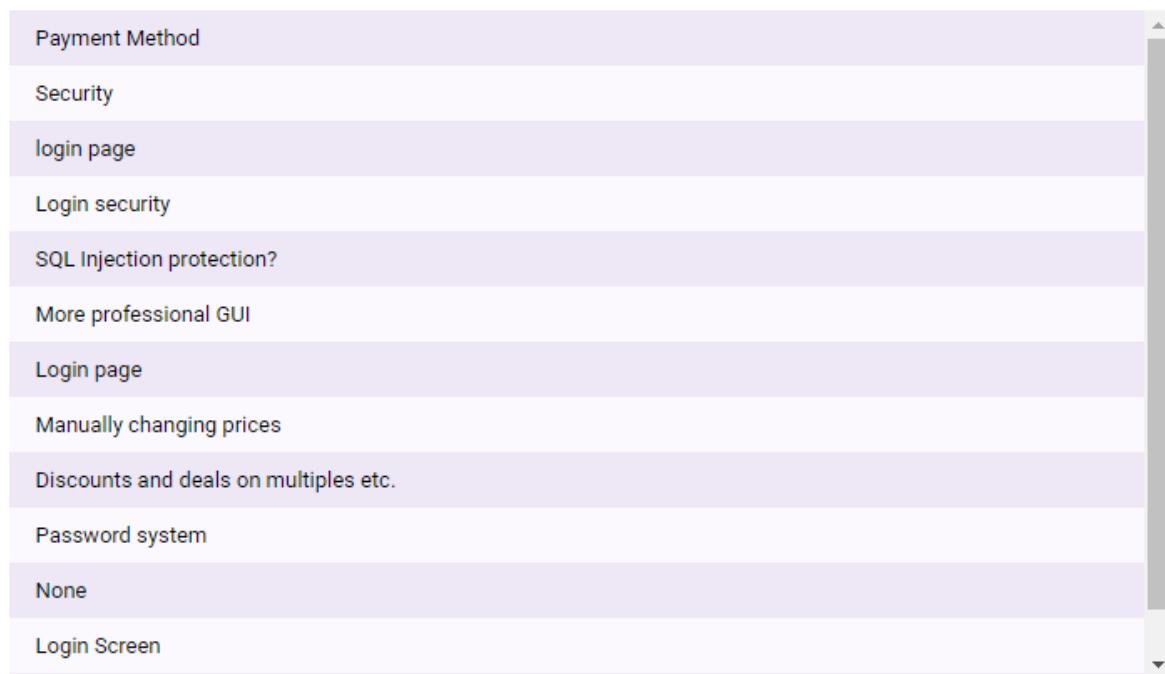
This could be referring to the ability in the settings for the user to have a background image on the home screen - which is not inherently necessary to the function of the solution, which is to book and organize rooms. While inherently not necessary, I felt it was still a feature which would be relevant to optimization - though the feature is still completely optional to appeal to every member of the demographic. Perhaps pertaining to the user more clearly that it is optional would be more helpful in a future iteration cycle.

- Transparent background colours & Transparent check boxes

Both of these ‘features’ picked up on by the stakeholders, and were recognized as being unnecessary and irrelevant. Upon looking into where they found the problem, I discovered that the program had some bugs in regards to transparency, which happened once I allowed the program to have an independent colour. The checkbox controls in the home screen are default to show the colour behind the form, hence they are transparent and can be quite confusing to click if the background is a solid black colour, for example. As well, the user can also select a transparent background to the forms, meaning the controls appear to be hovering with no real form to hold them - which can be very confusing to a user if accessed accidentally. From this, it is clear that deeper bug testing would need to occur, as well as patching these bugs in future updates - especially those in which may be discovered months after releasing the program (I will discuss this in maintenance later on).

Do you feel there are any features missing, or required? If so, what are they?

13 responses



Unlike the previous question, users felt there were quite a few features missing from the program to be standard to their liking. This is positive, as there are now many direct ideas which would directly improve the program in another iterative development, and are directly from the user's opinion meaning they would all be relevant features.

- Payment Method

This would involve a desire to further embed the program into the whole hotel system - this example being the payment method. Whilst the current program does create and display a final cost, as well as the ability to display when a room has been paid for, it does not directly link with a card payment terminal or cash register - it just goes off of the user's inputs. Hence, this is a request from the user to improve the program's level of embeddedness, and to manage more parts of the entire booking system within one solution - perhaps to enlarge the program's aspirations and to manage more items in one space.

- Login System

Many stakeholders felt a login system would be necessary, perhaps that a login system has become a standard for many professional business softwares, for the matter of organizing staff and security by assigning admin roles to certain members of staff, etc. A login system may require

the solution to become web based to better secure accounts, however a local database would also be viable. I suppose many of the stakeholders feel there would be a benefit to a login system, as it would make it easier to see which member of staff has assigned which rooms, as well as for the security of their own accounts - and a login system would be the next step to elevate the program beyond just a local solution, and to turn it into a web based solution.

- Security & SQL Injection Prevention

Adjoining with the login system, many users have shown to believe that the program lacks general security and other specific preventions of data loss, such as SQL Injection prevention. If I were to release the program for use in a professional environment, adding this SQL injection prevention would be essential to protect the hotel visitor's data from being lost or misused. This shows that users felt the features of the hotel booking were there, just lacking the additional features of protecting this system.

- Manually changing pricing, & Discounts

As mentioned during testing, the ability to manually change a room's price, overriding the booking's calculated price, was not implemented. It seems users noticed this, and noted that it was an important feature, especially alongside discounts. Allowance of discounts is another feature the test users seemed to want, for example, allowing a program admin to state that perhaps bookings that last over a specific week may become 50% off - which would then need to apply to the final price of the booking. This is a good identification, as with modern hotels being quite a competitive market, there are many deals which need to be accounted for using my solution.

Reflection on Function testing (Usability Features)

I now move onto evaluating how well each of the initial planned features were implemented into the final solution. I will frequently use quotes from the original feature plan and the test results to evidence the feature's success or failure.

Fundamental Features

- Initial Setup

I would deem this feature a certain success, as the initial plan described the initial setup as to ‘take in relevant information about the hotel’, and then subsequently be able to ‘change the initial given information’. Both of these main points were proven to be doable during the testing, as well as matching the criteria that ‘it will only need to be ran once’, which is proven in the first tests where I ran the program twice, and the initial settings only appeared on the user’s first time running the program - thus matching the initial plan. Whilst the initial plan does not mention input validations, the initial setup testing shows that this step does include deep validations which successfully prevent any errors from erroneous data, which is a fundamental underlying feature between all the features. However, the initial plan did mention ‘storing the settings in a database’, whilst the end product only stored them in a permanent text file (whilst still successfully acting as a database alternative). If I were to improve this feature for the next iteration, I would store these settings in a table in the main database, as this would better organize the information by keeping all the permanent information in one single file - the database file. Whilst currently small, if the project did grow bigger with more databases, it would definitely benefit from storing these settings alongside the other hotel data, to ensure the relevant data is not strewn across the entire computer’s file system.

- **Ability to input room and customer information and indicate a price**

The program was successful in this endeavour I feel, as the initial plan required the user to be able to ‘seamlessly’ book a visitor into a room, which was shown during the end user testing where I tested a range of speeds of booking and visually navigating the program, and the upper limit of these speeds was 7 seconds, which I would argue is quite seamless from a newcomers perspective. Whilst the feature was successfully implemented to endeavour this seamless nature, the feature itself wrote that it only required the allowance of inputting the visitor’s:

- Length of visit
- Size of room required
- Visitor’s name and phone number
- Any special requirements.

These 4 main categories were tested successfully in the first test, whereas I was able to input all this information into the database for a number of visitors - hence the fundamental nature of inputting room and customer information was successful. Also, validation was performed to ensure that these 4 categories were of their actual nature.

In terms of price, the plan states that the user can choose 'how much the hotel charges per room', which was successfully tested by showing a user could input money prices between 0-2 decimal places - thus validating the price nature of the input as well as allowing a monetary price to be assigned to a room. As well, the ability to 'accumulate the room and visit length to produce a final price' was successfully tested, as I tested the same room between 3 different length visits, to a successful result.

However, the plan designated that the user should have a 'manual input of price, in case of required discounts'. This part of the plan was unsuccessful, as tested above to no avail. As well, this missing feature was noted by the end users in the survey, showing its importance. In a later version, I would improve this by allowing the user to input a proper price using the same validations as previously tested, where this control would be checked first before writing to the database, and if the control has a value, it will overwrite the database price of the booking with the manual price. As well as this, I would allow for a 'percentage discount' option in the booking form, which would automatically take a percentage of all the room's prices once they were searched, and this percentage would then be noted and stored in the database once the room was booked to allow special hotel discounts.

- **Ability to input room and customer information from a range of different input devices**

This feature was 'simply the method in which the information is received' into the program, as my original stakeholders felt a range of devices should be able to access and navigate the program. This feature was successfully implemented, in that I tested a range of devices and if their input was valid enough to book a room in the software. As shown, testing a mouse, keyboard, touchscreen and puff suck switch showed that they all were valid devices to navigate the program with. Hence, the need of 'meeting the needs of our staff' was a success in conclusion. As well, the majority of my end stakeholder surveyors claimed that the

program was comfortable to navigate, which further evidences that this feature was a success.

However, if I were to improve this feature, it would be to design the program slightly more inclined toward these devices, as the program is built mainly for mouse and keyboard input and ;while these other devices work, it is slightly uncomfortable to use. For example, the keyboard requires tab inputs, so I would better align and count the tab indexes correctly in the program so that the navigation really makes sense using this device.

- **A Graphical User Interface**

In the plan, I stated that the research material were all successful because they all used some sort of ‘colour scheme to represent customer information’, and as shown from my tests, I successfully used a range of colours in the software to succeed this professionalism of using colour effectively, as the research material did. Also, just the fact that the final product was a GUI, shows that this feature was fundamentally successful.

However, the plan also desired the use of images to help show the use of buttons to the user (eg, using a ‘cog’ symbol to represent settings). From testing, it is shown that this visual element is missing from the buttons and comes at the sacrifice of being initially unclear to users what the purpose of the buttons are. To improve this for the next iteration, I would include these buttons using ‘picturebox’ controls in Visual studio, and perhaps research deeper into which icons and symbols would better suit the different buttons (EG, what would best represent a ‘book room’ button?). Also, another problem with this feature was that the colour usage; while included, was not as effective as initially planned - as shown from the user survey results. The results were unclear as to how users felt the colours were effective to their navigation, hence if I were to improve this, I would do more research into the psychology of visual colour cognition, and how it can be used to help guide and navigate users visually. From this research, I would then incorporate the findings appropriately into the forms of the project.

- **Calendar Display**

The plan detailed that the calendar will be of the form of a ‘2D axis array with time being on the ‘x’ axis and rooms being on the ‘y’’. As shown from the first test, the booking input data is presented in this

exact described format, and correctly shown. This means that this feature was a success in that the display successfully worked as planned, and no shortcut was taken in displaying the information. The plan describes this as a ‘fundamental feature of booking software’, showing that this feature allows the program to be on a similar level to professional software - which was the initial aim. As well, colour is described in the plan as to use colour effectively to make ‘certain elements of the calendar stand out’, and from the first tests this is shown, where green cells display bookings yet to be booked, and so on, showing that most of the plan for this calendar display was a success. As well, each cell is able to be interacted with and deleted if necessary, and has a genuine physical aspect of interaction with it, aside from just being a visual aid.

However, if I were to improve this feature, I would go beyond the fundamentals and delve into the personalization of the calendar, in which I would hopefully develop settings which would allow a user to have a custom layout to the calendar, such as choosing how many rooms should be displayed at a time, other than the value being fixed (which would be useful to those with less hotel rooms, as their calendar would then have less waste space). Developing this sort of feature would not be directly necessary, but would just be a feature of professional quality.

- **Saving and Restoring Data**

I felt this feature was a success, as the plan aimed to ‘save all essential data to a database, and restore it upon the program being restarted’. This feature was tested and proven by saving three bookings, and they successfully reappeared upon the program restarting. This was successful by using a database file as permanent storage to store these bookings. It also required to load and store information with ‘minimal hassle’, and this was arguably met as the user does not even need to choose a location to store the database file, as it is default to save with the program files. In terms of improvement, it would appear that this feature does not need improving, though would likely need to be updated heavily if any new features were added to the program - as this data would then need to be saved and loaded correctly. It could also be tested to see if the saving and loading is optimized, to see if the accessing times could be minimalized in the program to reduce any possible lag.

Advanced Features

- Keypad Scanning and unlocking upon validation

This feature was unsuccessful, as due to time and feature complexity limitations, it was unable to be implemented. As no other fundamental features depended on this feature being present, it did not cause much of a problem to drop the development of the feature - as it was simply an addition to the fundamental system to prove the system was able to 'embed'. As shown in the survey, the users felt the system required some form of embeddedness in the system to branch out beyond its basic functionality. Hence, if I were to improve the project, I would perhaps choose a more simple fashion of proving it can embed, before then advancing to higher levels of outer functionality, such as the keypad system. Also, I learnt from this that sometimes it would be simpler to introduce a pseudo-system to the project, (as to feed it artificial inputs from an 'artificial' keypad scanner) and to then program it accordingly to handle this pseudo information correctly, so then if the artificial information came from a *real* source, it could then handle it all the same - which I will describe further in the 'safeguarding' feature below. If I were to implement this keypad feature, I would research Python as a language to then learn to write a program to handle keypad scanning information (as that was the language the affordable scanner was written in), and then discover some fashion of passing information between C# and Python, perhaps if they shared the same databases. I would also designate more time to flesh out this feature, as it seems this feature alone would take quite a lot of development to achieve.

- Safeguarding (Storing Unlock Attempts)

This feature was also not successful, in that it relied on the keypad system being active to fully be available to develop. However, regarding the above 'pseudo system', if I were to develop the software further, I would have added a system to the program to handle keypad information and implemented the safeguarding feature - despite not requiring an actual keypad scanner. It would handle artificial information from a simple input source, as to handle the keypad information the exact same way even without there being a scanner. Then, I would be able to have a functioning safeguarding log implemented which was not reliant on the keypad feature being implemented - thus allowing

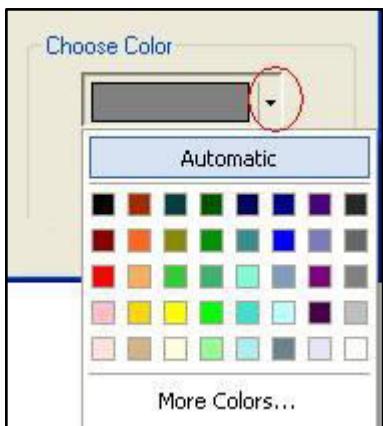
development to skip complex parts of development, allowing backtracking to occur to complete these parts.

- **Responsive ‘Beeping’**

Once again this a feature unable to be developed, though if I were to improve the program and implement this feature, I would focus on researching the Python language to be able to communicate with the keycard side of the system, as that is the language used to communicate with the scanner I purchased. I would then need to link it with the function of detecting either ‘scanfailure’ or ‘scansuccess’ to then act on different sounds, depending on the function outputs. I would also need to research electronics on how to wire the buzzer and scanner together, to be read by the same piece of code.

- **Colour scheme organization**

I described in the plan that the user should have ‘total freedom’ over the colour scheme of the program, and that ‘the key elements of the program are high contrast, and stand out beyond the colour scheme’. As shown during the testing, ‘total freedom’ is followed, as the program gives complete control to the user as to which colour the background of the forms should be. As I was able to change this colour to a maximum range of colours, this is arguable to total freedom. Also, by testing using an obtrusive colour, it shows the visibility of the elements of the form, and if they are vivid and high contrast. When testing this, it appeared that the black text was subdued and unable to be read, however, the key elements (calendar, buttons) were still highly visible despite this, thus showing that they are kept visible and at a viewable level, despite any level of visibility. However, if I were to improve this feature, I would fix the text so that it may read with any background colour chosen - similar to treating it as a key element. As from the testing, it is shown that the text is subdued, so perhaps putting a fixed background colour on the label would make all the text permanently visible to any background colour. Hence, I would only label this feature a partial success - as it still requires validation on the colouring to prevent these logical errors.



The image above is the fashion of picking a colour in the solution, highlighting the true freedom of choice.

- **Search Function**

The purpose of the search function was to ‘pick out specific rooms from the entire directory of hotel rooms, depending on a criteria decided by the user’. I feel this was successfully done, as during the tests, it shows that multiple rooms which all met the conditions could be picked out by the function successfully, and that it ignored all the rooms when it did not meet any conditions - hence showing that the search function worked correctly, and as expected. The search function was important to decide that using the computation solution was more beneficial than a paper based solution, as a paper based solution could not identify a list of rooms from given criteria as fast a computer could with definite results - and as the survey results showed that the majority of users felt it was hugely more efficient than a paper based solution, this shows that the users were confident that this search function’s original purpose has been met to outmatch paper based solutions. If I were to improve this feature, I would include a ‘word specific’ search, which would read all of the ‘extra notes’ in the room record, and find certain specific words and return the room if their notes included the searched word. This would improve the program and would require heavy string manipulation to read the notes string. It would improve it by given more in depth search, and perhaps give more purpose to using the ‘notes’ string to store specific, bespoke information which may make a room unique.

Success Criteria

To evaluate if I met the terms of the success criteria, I analyzed each individual point of the original success criteria, and compared it to my

final tests and survey. Using quotes from the original criteria, I then made my final decisions as to whether or not the criteria was met:

- Input Peripheral criteria

This criteria involved the input methods of the program, focusing mainly on ensuring ‘a range of different input peripherals’ could interact and navigate with the program, including peripherals designed for those with a physical disability for the purpose that the ‘majority of the demographic’ would have comfortable access to the program. I would deem this criteria as a success, as during testing, I tested the functionality of 4 input devices; keyboard, mouse, touchscreen and puff suck switch: with the puff suck switch being designed for those with a disability. With all these devices being able to successfully book a room (albeit the uncomfortability of using the keyboard, whereas would be improved next time by better planning the tab indexes, as described in the feature evaluation.), this evidences that a range of devices successfully worked with the program and met this criteria. As well, the survey question about the comfortability of navigating the program using their input device had high results, showing that not only the testing evidences that these devices work, but a range of devices used by my stakeholders were applicable thus also succeeding the ‘majority of the demographic’ aspect of the criteria.

- Colour and Images criteria

This criteria dealt with the visual aspect of the program, and ‘using colours and images effectively in the aims of making the program simpler to navigate’. The original criteria would be met by using a survey questioning how simple the program was to navigate, and following that question by asking how much the colour scheme played into making this navigation simple (with 7 being the minimum score for success).

Through testing, I would say that this criteria was only partially successful, in that users responded to the survey about the simplicity of navigation, where the score for this question had an average of 9.4 - showing that the users felt the GUI was used effectively to communicate the program’s usage, and the average was above 7 - clarifying this success. However, the images were not implemented into the program’s GUI and the survey showed that many thought that colour was not effective in aiding them navigate the program - showing that; whilst the GUI was still successful in being ‘simple to navigate’, it was not entirely

the planned images of colours in the GUI. To improve this for next time, I would do research into colour cognition, and how to use colour effectively in design, as well as imagery. I would then act on this research, and implement it into the design of the program for the second version.

- Embedded criteria

This criteria was to be met if ‘the program could be proven to be embeddable with another system’. Originally, this proof would come from implementing the keycard system, however, this feature was not implemented due to limitations later discussed. Subsequently, as there was no other proof of the program ‘interacting with another system’, this criteria was ultimately not met. However, many of the survey respondents felt that the program; in its current state, would well integrate with other systems in the entire hotel system - yet was lacking this embeddable nature. If I were to prove this criteria on the next development iteration, I would perhaps choose a less daunting feature to implement this embedded nature, such as (learnt via relevant survey responses) a payment method system, which linked to a physical card payment machine. Thus, it would only need to communicate the end prices to the card machine and be much more simpler, yet still prove legibility of the entire program’s embedded nature.

- RFID Efficiency criteria

As mentioned before, this criteria failed completely as the feature which the criteria required to test on was not implemented (keycard system). Thus, I was not able to follow through on the relevant tests to prove a ‘strong and reliable RFID scanning system’. However, if I were to incorporate this system into the next iteration, I would have tested intently to show that the RFID scanning was reliable, to match the professional standards of scanning security. I would have also researched different available RFID scanners to ensure that when this efficiency was tested, it matched and passed the original planned criteria test of having ‘a 100% efficiency’.

- Simple Visuals criteria

This criteria described that an ‘inexperienced user must find the program simple to learn’, in which I would test a new user’s reaction to being asked to read and decipher a part of the calendar in a timed reaction question. Then, a success for this criteria would be counted if

the mean time of all the reactions came to below 5 seconds. I would claim this criteria was met, as during the survey I trialed this test to record the timings which resulted in a mean time of 3.9 seconds - which is quite well below the 5 second criteria. Thus, as the criteria test was performed with the stakeholders and was below the limit, I would claim this criteria is met, meaning the program is successfully simple to inexperienced users.

- Simple Access criteria

This criteria was to test how well users could adapt to 'successfully booking a room to customer'. It would test the simplicity of how the GUI taught the user where certain booking information was to be inputted - where I 'aim for a 100% efficiency'. Rightfully, during testing, I gave the task of booking a room to the stakeholders, and 100% of the users were able to book a room successfully - thus matching the efficiency aim and clarifying that this criteria was strongly met, in that the program does not confuse users and all the visual information is relevant.

- Searching Accuracy criteria

This criteria ensured the accuracy of the search feature, testing that the search 'should be an accurate feature. The original test hoped to have multiple rooms searched for, which every search successfully returning the expected room with the matching room in the database. I deem this criteria to be met, as during the search feature testing, I searched for multiple rooms where there were multiple rooms matching the criteria, as well as no matches at all - and all of these tests were successful as shown, showing the efficiency and integrity of the search function.

- Validations criteria

The important validations of the program were successfully tested at the end of each key stage, to ensure that the necessary comparisons were put in place during each function which required manual input, to ensure no errors enduring data types occurred when, for example, placing the data in the database. The original plan wanted to use 'black box testing to input the wrong and correct data types into data fields', to show that all the inputs would be handled successfully. These black box tests were done during the key stages, during the original key stage tests to ensure each individual stage's inputs were validated. Also, the money inputs were successfully validated during the final black box tests - thus showing the level of testing that has gone into confirming that the

entirety of the program's inputs are validated. Thus, I would claim this criteria is met.

- Overall Black Box Efficiency criteria

This original plan to black box test was met, as shown during the in depth feature black box test - in which robustness was tested and analyzed successfully. Whilst some of the outer features were not completed, the basis of the program being able to "show final prices, take bookings and visitor information, and display information effectively" were all shown to function, showing that this criteria was met, and shown to work and 'pass' this black box test.

Maintenance Issues

- Forms Service Updates

One maintenance issue would concern the platform in which the code is written on. As the code is written in Visual Studio Forms, if this platform were to stop receiving support from Microsoft, it would be difficult to maintain the security or publishing of the program, especially if I were to develop the program further. As Microsoft are actually considerably giving less support to Forms, and more into another GUI based coding service they are building, it is likely that this problem may occur, and the maintenance of porting the program to another language or platform may become a reality, if the program were to be able to keep receiving updates or security inspections. This maintenance could be acted on by 'backing up' another version of the program to a more modern coding platform, to ensure that it can keep being updated with relevant info and help from the support platform.

- Dedicated Storage

While unlikely, there is a possibility that the program requires more memory or permanent storage than a client can provide - such as if a user keeps adding rooms and bookings, the databases may grow to a size which is too large to store permanently. There are a number of features I could provide to counteract this maintenance, with the first being to recommend storing the program on a larger disk or storage device, to accommodate the larger database. Also, I could research deeper into how better to store data in the database, especially if I add more features to the program, as if there is a fashion of lossless compression, this would reduce the need to more the database, and also

reduce the maintenance from occurring altogether, if the database cannot grow larger in size due to good compression.

- Discovery of bugs

As mentioned previously during the survey analysis, there are some bugs in the software which may not be discovered by me, but by clients upon releasing the program, as there is no chance of being able to identify every possible bug in the program without testing extensively beforehand. Hence, to maintain the program in the chance of a bug, I would need to offer a forum of sorts to clients, which would allow them to voice any bugs they noticed - which will allow me to communicate and learn about these bugs for a lot less effort than beta and alpha releases. From then, an update could be performed on the software - upgrading it to a new version which fixes these bugs, and then releasing this patched version back to my clients. This cycle could continue until the program is ultimately robust.

- Portability to newer OS'

There is a chance that the program becomes outdated to modern operating systems in years to come. This is a problem, as users who may want to use the program on new operating systems may run into portability errors, or perhaps the program may not run at all. This would be counteracted, by ensuring that the program is tested on modern operating systems, and if necessary, port the program using the necessary tools to ensure it is able to run on the newer operating systems, using the same update method as described in the maintenance above.

- Database Updates

Many times during development, I had the problem with the database service being out of date, which meant the connection was denied between the program and the database. This is a problem, as it would cause definite errors if the database service is out of date. To ensure this problem would not occur, I would need to perhaps check for updates whenever the program is ran, to ensure the user has the latest version of the service which works with the local databases. This would need to be included with every program, though may cause a problem to offline users who cannot update the database service. I would then, likely need to research how a user would maintain their databases for optimum

usage despite this ability to update - as there is likely a solution to counteract this problem.

Limitations

- Keyboard Interaction

A mentioned problem with the program was the interaction using the keyboard, in which the tab indexes of the program were in a strange and uncomfortable order, which had not been thought out and only realized during final testing. This was a problem through the limitation of time, as I suppose with more time, smaller problems with attentions to detail like this would be worked out and solved, as under the current time limit restriction, the priority was completing the fundamental features. I would fix this problem if solving for the next version by working out which order the tab indexes should be in on the forms, and then testing this order to ensure it felt comfortable to navigate the program. This would make the program better by being better accessed via a keyboard, and more ergonomic to users who prefer using the 'tab' key on a keyboard to navigate programs.

- Visual (Colour and Imagery) Aspects

Issues with the colour choices and the lack of images has been discussed previously, in that the program's colours (via the survey response) were not directly helpful in navigating the program, as well as lacking small 'icon' images to help give more personality and identifiability to buttons. This is a limitation through lacking a larger team or lacking resources, otherwise a larger team may be able to have split the jobs between them - such that an individual would focus on the visual aspects of the program, in that they would then have the resources and time to focus on designing small icons, and researching the cognition behind colour recognition to have the best, most effective visual aspects of the program. With these improved visual features, it would improve the program by giving a piece of professionalism and identifiability to the program, rather than being just a default, seemingly dull set of forms.

- Missing Keycard Feature

A problem with the criteria highlighted that the program was missing the planned embed proof - the keycard scanning system. It is a problem as it missed a key point in the success criteria of the project - that the

program should be embeddable with other systems. It was not implemented via the limitation of time and resources, in that it would require a knowledge and research in RFID scanners, as well as researching how they are coded, and electronically manipulated. If I had more time for another version, I would definitely research into the scanners to see how one could be implemented alongside the booking system, and then implement the system to have a real use with their communication. Otherwise, I would research another system which could adjoin with the booking system, such as a payment system to prove this criteria that the solution can embed.

- Unclear Coding References

Throughout the coding of the project, I was generally vague with naming variables, forms and databases. This means that if I were to develop the project alongside a larger group of peers, or were to advance on the code in the future, it would be extremely confusing and counter productive to try and understand the variable purposes when they are so vaguely named (such as names like 'num1' and 'form1' not having a direct meaning to their purpose). In future developments, I would ensure my variables are rightfully named with a relevant name to ensure it can at least be universally understood and picked up on by a working peer, as this would help development by saving confusion when trying to understand which variables have what purpose - thus saving time on development. This was down to the limitation that there was only one person working on the project, hence I did not need to specify too much about variables, albeit it being common practise.

- Localized to the UK

By using only the English language to label items in the program, and also only using a '£' sign to describe money, this limits the program to only being used correctly in the UK. This means that my only market would be the UK. To improve this, I would need to add extra options in the settings menu to select either the language the program would write in, (followed by all the necessary validations depending on this language, different to English) and the currency the program would use, where the program would then take this option into account when loading the text in the forms.

- SQL Injection Security

Due to time and experience limitations, I did not include SQL injection security in the product, due to it not being an immediate priority to the success criteria in this development. It is a major problem, as if I released the solution without this security measure, the software would be a major risk of data loss, as anybody with malintent would be able to delete and perhaps access the databases, which includes private data of visitors which must legally be kept private. I would fix this by researching SQL injection prevention techniques, and including them in the validations of the program, to then release the product with no legal worries.

- Software Learning

Many users in the survey felt the program may be daunting to learn for some, so this is a limitation that the user is just expected to know how to best use the program upon running it for the first time. This is a problem, as the user then may miss out on certain features they were not made aware of because they were not taught. I would improve this problem, by providing an in-program tutorial, guiding the user through the program and highlighting every useable feature. Alongside this, there could also be a 'readme' manual with the program, detailing these features as well - though a on screen GUI tutorial would be much more effective at teaching through interaction, hence would be the likely option to correct this limitation.

- Professional Interface

Currently, the interface has a 'dull', default feel to it, in which it uses the default controls provided by Windows Forms as a service. Through the limitation of time, the full visual professionalism of the product was not a concern to the success criteria, though if I furtherly developed the program, I would have a larger team to deal with problem, perhaps a graphic designer to design bespoke buttons for the program to give it a more professional feel, which would better market the product to give the solution a unique feel. The designer could design special buttons and calendars, rather than the dull default designs to fix this professionalism limitation.

- Inefficient Coding

A problem with being an independent coder meant that the code I ended up writing is likely inefficiency, and likely has much better methods of doing the exact same purpose. With a larger team on the second

version, I would share ideas between the team to ensure the most efficient method of writing the function was thought up, as more ideas from multiple people would likely achieve a better thought out end product, with much more efficient and easier to visualize code - which also uses less memory when running.

- OS Portability

Right now, the program is likely only designed to run on the Windows operating system, meaning users who prefer Linux or macOS would not be able to use the solution for their hotels. This filters out a majority of the demographic, but would be improved if I ported the program to be universally understood by these operating systems. This could be done using Java, as Java can be intermediately ran on these systems. Hence, I would port the program to Java to allow this portability for more users, given I was improving the solution for an improved version, with more time and resources.