|  |
| --- |
| Evidence template:  AS91902 - Use complex techniques to develop a database  AS91907 - Use complex processes to develop a digital technologies outcome |

|  |
| --- |
| **1. Planning** |

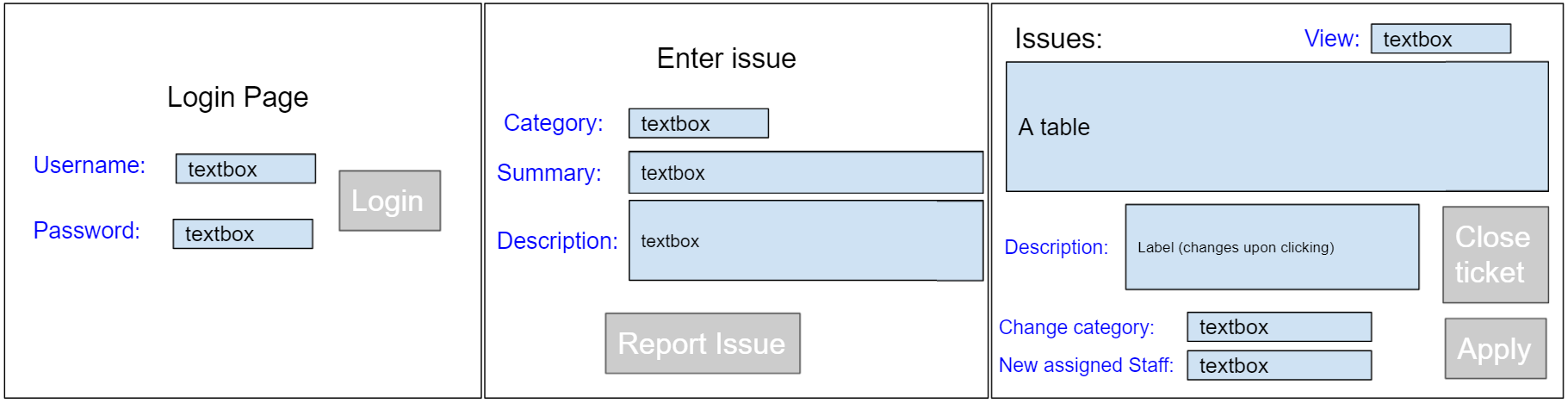
**What am I making? What is its purpose? Who are my end users? How does this affect the design?**

I am making a program in Visual Studio so that teachers and students can send in IT issues. Admin staff is then able to see these issues and assign them to different staff members. The end-users are the students, teachers and admin staff that need to be able to use this program. This means I need a design and instructions that are easy to follow, rather than having a messy layout that no one can understand.

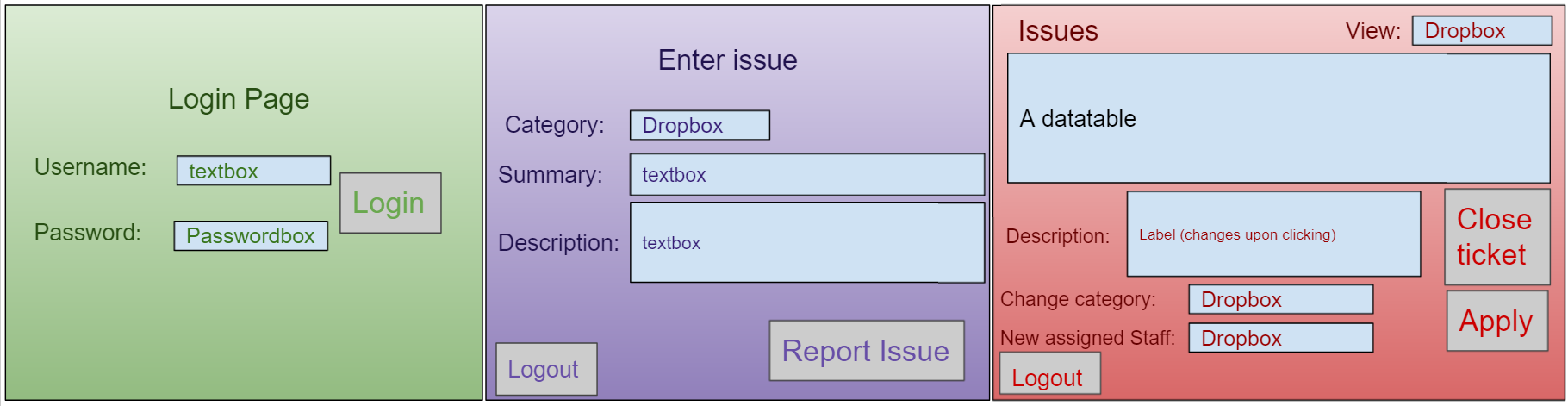
**Program interface design**



Upon reading about design principles, I realised this layout is very hard to read and very messy i.e., it’s not aligned or balanced, it has a lot of different colours, and the headers and labels aren’t very clear, they are all different sizes and same counts for textboxes, so I’ve tidied it up a bit:



After the first change I realized there’s a lot of empty space and the colour of the labels don’t match the textboxes, so I’ve changed it to make it more colourful. Furthermore, I realised logout button would be nice and that some textboxes could be a drop box and the textbox for the password in the login page could be a password box since therefore other people can’t really know what you’re typing. So here are the screens that I plan in using.



After this design I asked some friends for feedback. One of it is that they want the ‘Login Page’ label to align with the ‘Enter Issue’ label. Another one is that they would like that the ‘Report Issue’ button and the ‘Logout’ are bottom aligned. However, they did like the colour scheme. Another person would like if the ‘Close Ticket’ button would be somewhere less likely that you would accidentally click it. Another person said to capitalize all the labels/buttons, to be consistent: Graphical user interface, application

Description automatically generated

Note that I considered aesthetics into this as the outcome needs to have an appealing quality, for it to be used. You don’t want to use something that’s messy and unclear what everything is. I addressed this by ensuring that the screens are in such a way that’s clear and everything looks good. The colours match each screen functionality i.e., green is used for trust, while red is used for anger and hence problems.

**Database structure**

**Fields**

|  |  |  |
| --- | --- | --- |
| **Field name** | **Data Type** | **Justification** |
| Username | VARCHAR(15) | The username shouldn’t be longer than 10 characters hence I’ve put a maximum of 15. VARCHAR opposed to CHAR since for CHAR the username must be 15 characters, but everyone wants different username not which are necessarily 15 character long. |
| Password | VARCHAR(45) | VARCHAR has opposed to CHAR because password is varied length and CHAR would make it that length. I put an upper limit of 45 because if the password is any longer, the user is likely to forget. |
| occupation | VARCHAR(45) | This is put in manual in the database and there are only 3 options, and this is default one and hence easier. VARCHAR is used because the 3 options are varied lengths and CHAR fixes the length. Hence, it’s VARCHAR(45). |
| Ticket/idticket | INT(unassigned) | This is an integer that can go up if the school is big, so I haven’t put on a limit for this. I’ve put int so that it’s number only which is what we want. |
| Category | VARCHAR(45) | All the categories aren’t longer than 45 characters so 45 is fine. VARCHAR is used because the categories don’t have the same amount of character so CHAR can’t be used. |
| Summary | CHAR(100) | This is a textbox that users can put in, so it needs to be somewhat longer than the usual 45. I used CHAR because I want the user to have a decent summary so I want it to be 100 characters long, as opposed to VARCHAR which would allow less. |
| Description | VARCHAR(255) | This is optional for the user to put in so I’ve decided to leave this one VARCHAR so the user can put in nothing. I’ve put in 255 because if the issue is difficult to explain they have somewhat more characters to explain. |
| Assigned | VARCHAR(15) | VARCHAR to allow different lengths for username. 15 because it’s the maximum length of username, and the only other option is unassigned which is also less than 15. |

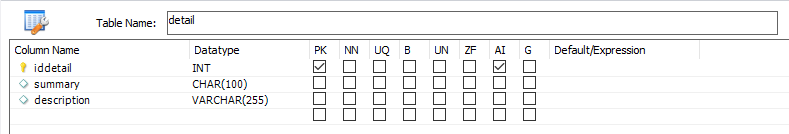
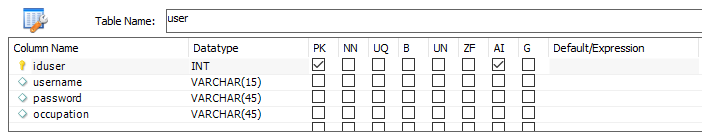


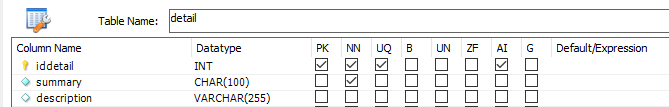
This table is very long and has nulls, so I decided to split it + not very clear primary key. This table has a few nulls and is extremely hard to understand. So, in this case more tables would be better so that it’s clearer and separated into two.



Upon learning on 1 normal form and 3 normal form I decided to split the ticket table once more because they aren’t allowed to relate to each other, but summary and description clearly do, and username and occupation are also related so I’ve decided to add an iduser so that is clear which user entered the ticket.



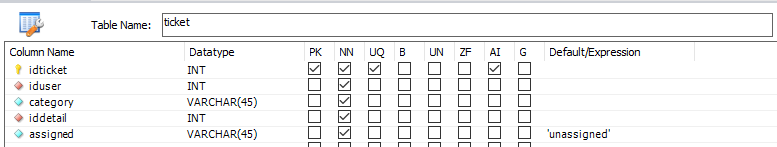


I have just put in the columns; the ticked boxes are the default one... Hence, I decided to double check everything and just check it: The description isn’t compulsory so it can be null as opposed to the summary which is compulsory and hence can’t be null. Same for iddetail this one isn’t null since it’s the primary key, and therefore it’s unique as well. This one auto increments as new tickets gets added so that it gets autoassigned. Note summary and description aren’t unique as different teachers/student can enter the same problem.

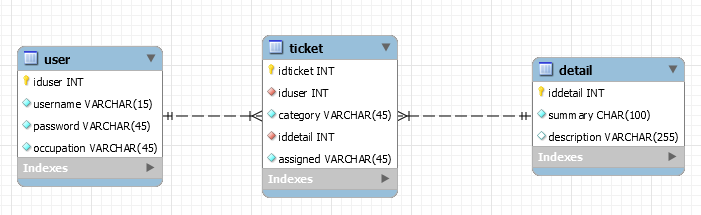
Graphical user interface, table

Description automatically generated

Iduser is the primary key and hence unique and not null. It’s important for the program to know the username, password, and occupation so that we know whether the credentials are valid and know to which screen they should go to, hence they are all not null. Username is unique as we don’t want any confusion, however, people can have the same password (though unlikely) and occupation.



Idticket is the primary key and hence can’t be null and must be unique, it auto-increments as a new ticket gets added, it’s needs a new number and auto increment is the default one. The category, iddetail and assigned all can’t be null since they all need to be visible to the admin staff. In this case, iduser isn’t necessarily unique as the same user can resubmit the ticket, however, iddetail is unique as it is fit to the description. The column assigned default is ‘unassigned’ so that upon each new ticket it’s clear to the admin that it isn’t assigned yet.



|  |  |
| --- | --- |
| Inserting a new user through the database. | INSERT INTO `student2021130434`.`user` (`iduser`, `username`, `password`, `occupation`) VALUES ('1', 'gabe', 'admin', 'admin'); |
| Inserting a ticket. I’ll use the textboxes/drop down menus to insert the details but for now this is just an example. | First add to ‘detail’ as otherwise it won’t work. Then add to the ‘ticket’ table:  INSERT INTO `student2021130434`.`detail` (`iddetail`, `summary`, `description`) VALUES ('1', 'the wireless internet isn\'t working. ', 'None of the computer in the classroom are able to connect to the internet');  INSERT INTO `student2021130434`.`detail` (`iddetail`, `summary`, `description`) VALUES ('2', 'The computer has shut down and doesn\'t want to turn on ', 'The laptop said it had 5% and a few minutes later it just shutdowns out of nowhere'); |
| **Upon realizing that CHAR100 would fill with spaces so that it is 100 characters long. Also 100 characters isn’t a lot. So, I’ve decided to make summary to VARCHAR(200) and increase the description to VARCHAR(750) to allow for somewhat more description for complicated problems.** |
| INSERT INTO `student2021130434`.`ticket` (`idticket`, `iduser`, `category`, `iddetail`) VALUES ('1', '2', 'Wireless', '1');  INSERT INTO `student2021130434`.`ticket` (`idticket`, `iduser`, `category`, `iddetail`, `assigned`) VALUES ('2', '2', 'Software', '2', 'gabe');  note how column 1 says unassigned – this is showing that the default expression works |
| Updating category/assigned columns: | UPDATE `student2021130434`.`ticket` SET `category`='Hardware', `assigned`='unassigned' WHERE `idticket`='2';  UPDATE `student2021130434`.`ticket` SET `assigned`='gabe' WHERE `idticket`='1'; |

**What project management and version control tools will I be using?**

I’ll be using a Kanban board on Trello so that I know what I still need to do. Also, I will be using subversion to see what I’ve done and to see where I can improve.

**Decomposed list of the components of my project:**

Graphical user interface, application, Teams

Description automatically generatedGraphical user interface, application, Teams

Description automatically generated

|  |
| --- |
| **2. For each component:**  These steps will be repeated for each component of your outcome. Copy and paste the table below as many times as you need. |

Graphical user interface, text, application

Description automatically generated

|  |
| --- |
| Component: Setting up the screens (Login, reporting issue, and managing.) |
| Show *trialling* of alternative techniques/components. Explain what you are trialling and justify which techniques you are going to use. How is this going to improve functionality? (Include screenshots) |
| 1. The combo box/dropdown menu I wanted to add it through the C# code:   However, it can also be done manually through the XAML file:  Therefore, I’ve decided for the combo boxes for category (on the Enter Issue page and Issues page), to try to do it manually rather than coding, so that my C# code is less dense and hence more easily readable.  However, for the viewing drop box, it has the usernames of the admins, which isn’t fixed, plus 2 other options (all and unassigned). Hence, I’ve decided to put in the ‘all’ and unassigned’ manually and add in the usernames using c#, to ensure my C# code is as short but clear as possible. For the new assigned I will do it similar; I will put in the unassigned manually and add in the admin staff using C# since I need access to the database. However, I will do this in a private function so that my code is clear and easily readable.   1. For the password box I inserted a textbox so that I can what I’m typing: *.* However, now other people can see it to so I changed it to a password box, so that people can’t see your password, however that also mean that I can’t see what I’m typing. Hence, I’ve decided to add a show button (checkbox) that would reveal the password, using a textbox, when checked and hide the textbox, and show the password box it if unchecked. This is to ensure that the password as default can’t be seen but it if needed the user can see it.      1. For the display description label, that show the description of the selected ticket, it can be very long, and labels can’t scroll. Hence, I’ve decided to change it to a Read Only textbox so that if the description is long, the admin staff can still read the entire description, not just part of it.  A label also doesn’t have a wrap function that puts all the letter within the left and right boundary. Therefore, textboxes are better since they have word wrap and you can add a scrollbar. Scrollbar can be set to hidden, visible, auto or disabled. I’ve chosen for auto, which means that if there are unseen lines the scrollbar is visible, but if the description fits within the bar is hidden. |
| Show *testing* of your selected technique/component. What did you find out? What do you need to do as a result? How have you improved your design? |
| 1. The checkbox for view password only works one way. It doesn’t work if it gets unchecked. Hence, I went to check the XAML code to discover it only had a checked function but not unchecked. Hence, I’ve added the unchecked with the same function:  to . After having retested this function, it does work both ways now. The individual components of the textbox and password box also work. 2. Also, as you can see above, the combo boxes are filled with the right names, that I’ve put into the database. *.* As you can see at only has 2 admins, with the usernames gabe and xxx. 3. Lastly, the description textbox now has a scroll bar if the content is overflowing. This is how I want it to function i.e., this works.  * When I ran the program, the screen was not completely visible i.e., the output screen is slightly smaller than the designer screen:  Therefore, I’ve added 15 to both the height and width so that everything fits within the output screen: |
| What implications are relevant to this component, and why? |
| * Functionality – This program needs to function well to be successful. Hence, I need to ensure that it does what the brief asks and that everything works the way I want it to work. * Privacy – People put in their password, which is how they can access either the report or the view issues tab. Since password need to be secret, they need to be private. * Usability – This program is going to be used by different people so it needs to be easily usable so that people of all experiences can use it. * End-user considerations – Since the end-user will be students, teachers and admin staff and need to ensure that they want to use the program and that it isn’t a hassle to use. * Future proofing – This program might be used in the future, so I need to ensure that everything works and that it is clear what everything component does. |
| How did I address these implications? |
| * Functionality – I ensured that the dropdown menu works, and that the description can be seen well, so that the user can see everything. Furthermore, when the unchecked button wasn’t working, I looked at why and fixed the problem. This is to ensure that the program functions correctly without bugs. Hence, I test all the functions to ensure it works. * Privacy – I ensured that the password could be hidden on screen so that it remains secret so that other people can’t look over the shoulder and see the password and as a result use their login. * Usability – Some of the buttons were hidden and hence weren’t very usable since you had to click them very accurately. I addressed this my slightly increasing the width of the window from 450 to 465 and the height 800 – 815, to ensure that everything is visible and hence can be used. * End-user considerations – I asked for some feedback from some students and a teacher and here’s what they said: the view password checkbox should be underneath the password box rather than besides it. Then the next end-user told me to move the login button closer in.   Before:  and After:  On the report issue page, the user found it a bit cramped between the summary and description with the buttons. Hence, I’ve moved up the header to make more space for it and align the top of the description with the top of the textbox for description.  Before:  and After:   * Future proofing – I’ve named every component on the way clearly so that future programmers, that want to look at my program, know what everything is and why it’s there so that they know what is happening at any point in the program. |
| My project management, version control tools, and any feedback on your program at this point in the project.: |
| Note that I intended to do the combo boxes along with the other C# coding, however, I found another way to do halve of it, hence, I’ve decided to do it simultaneous so that I ensured that they are full completed. Furthermore, I didn’t realize I could set a character limit on the textboxes so thought I would need a trigger but turns out I don’t need a trigger to set the max lengths.  For subversion log please refer to the word document titled subversion\_log\_3.3\_3.8 |

|  |
| --- |
| Component: SQL database |
| Show *trialling* of alternative techniques/components. Explain what you are trialling and justify which techniques you are going to use. How is this going to improve functionality? (Include screenshots) |
| 1. Upon realizing that admin want to see the description and might think something is wrong with the program if the description box is empty. I could either add a trigger as shown below or have a default value:   .  Both the trigger and the default expression works, if it’s entered manually (through mysql). however, when using C# code for later on, the text will be empty (“”), not NULL. Hence for later on, it would be easier to use a trigger, so I need to do less if-else statements in the C# code and it’s easier to troubleshoot in the future if something is wrong (future proofing). Hence I’ve decided to go with the trigger.   1. I could go with 3 users, 1 admin/teacher/student or with 6, 2 admin/teacher/student. One will give a clearer outline however, for testing purposes it only shows if it works once not twice. Hence, I’ve to add extra users into the user database so that when I test, I can ensure that it works twice rather than just once. This is because if only one works it might be just the specific code whereas if you have 2, you need more general code:      1. I could either use a normal password, like I have now or a hashed password that no one can see. Currently the password for the user can be seen by everyone. What I mean is, it isn’t hidden and you can easily find it. Hence, I’ve decided to go add a hashed\_password column for privacy reasons with BINARY(32):   Next, I needed to ensure that every password that already exists has a hashed password so I’ve decided to run the following code:    So now I have to add a trigger that when a new password is added that it is hashed immediately as the hashed\_password shouldn’t be null and then set the password to null. Hence I have to change the setting for password as it is set to null after the trigger:  Finally, the already esixtant password are still visible, just the added are hidden. So I’ve decided to run the following code:   1. Summary and description could be kept at 100 and 255 characters respectively. However, this isn’t very long as 100 characters is about 20 words. Hence, I’ve decided to increase summary and description to 200 and 750 respectively. Also change summary to VARCHAR instead of CHAR as CHAR just fills up with spaces if it isn’t 200 characters long: |
| Show *testing* of your selected technique/component. What did you find out? What do you need to do as a result? How have you improved your design? |
| 1. As you can see when enter just an id and summary to the details table, a description is automatically added. This means that the default expression works. However, when added in c#, it is empty: However with the trigger, it’s added, with the same code, this happens: 2. The table has been updated to contain the extra users: 3. I rechecked my trigger to see if there were any bugs in it. Turns out, that I needed to add the END IF; up so that the 2 SET lines would always run. Now I reran the same code as above and it worked: 4. Summary can now have 200 characters and description 750 characters. I’ve confirmed this by entering a 200 and 750 long string:  * User Testing: One of the end-users looked through my MySQL database and asked what would happen if you would update a password, since the trigger is only for BEFORE INSERT:  As you can see the password is visible and the hashed\_password is probably not updated. So, I checked out how to have a trigger for both BEFORE INSERT and BEFORE UPDATE. So, initially I added a trigger for BEFORE UPDATE with the same code as BEFORE INSERT to ensure it is correct: However, I looked for alternative methods as copied code isn’t ideal for a program. I tried procedures which can be called to run specific code. I created a procedure with the same code between BEGIN and END as BEFORE INSERT. Hence, I’ve decided to use procedures as this would prevent copied code and makes it work. Then I had to change both the triggers so that they would call the procedure and I had to ensure the code worked:   The program doesn’t recognise that at the end hashed\_password will be set. Hence, I’ve decided that hashed\_password can be NULL in the table, as it will always be generated through the PROCEDURE/TRIGGER and hence it doesn’t really matter if the NULL box for hashed\_password is ticked or not:   I reran the same code as above and this time it worked. The procedure worked as password isn’t ‘bbb’ and the hashed\_password isn’t NULL. |
| What implications are relevant to this component, and why? |
| * End-user considerations: This is relevant because the people who are using the program want it to work and ensure that every part of the database is clear. Hence, I need to consider what the end-user would like it to look like * Functionality: I need to ensure that every part of the program works, i.e., is functional. Hence, I need to consider the different functionalities of the program to ensure it doesn’t crash. * Privacy: We need to ensure that all data is stored and is secure. The password shouldn’t be easy to find or figure out, as we want that to be private, so that not everyone can login. * Futureproofing: The program is likely to be used for a few years, hence I need make sure that everything works and makes sense so that future programmers understand every component of the database. * Accessibility: I need to ensure that the database is easily accessible so that it can be easily updated, to ensure that IT staff can add and remove members. |
| How did I address these implications? |
| * End-user considerations: I allowed end-users staff members and other students at the school to test the database and give feedback so that I can ensure it makes everyone happy. Secondly, I considered what I would like it to look like in the end so that end-users are happy to use it. * Functionality: I tested all components of the database alongside with end-users testing so that there are no bugs in the program, so that everything works as functions as it is supposed to. * Privacy: I ensured that the password isn’t easily found, by hashing the password so that you can’t see it and hence it’s kept private to you, as no one can see it. * Futureproofing: I considered what would be easiest to understand, in terms of altering and understanding everything, to ensure future staff can easily edit it and know what each component is for, rather than just guessing and making unnecessary changes. * Accessibility: I added triggers and functions to ensure that everything runs smoothly when it’s updated or new data is entered, to ensure staff need to do minimal checks. |
| My project management, version control tools, and any feedback on your program at this point in the project.: |
| For subversion log please refer to the word document titled subversion\_log\_3.3\_3.8. I am on track since I have still 4 weeks to complete the C# code and for testing, which should be plenty of time. |

|  |
| --- |
| **At this stage I ensured that I followed all the conventions that I need to follow. For example, I had to add \_ between id and singular table name. Secondly, I had to change the names of the foreign keys in the ticket table, I had to add fk to the name. So, it changed from iduser to fk\_user and iddetail changed to fk\_detail. Thirdly, I did all my component names as Pascal Case, while it should’ve been camelCase. Hence, I changed all the buttons, label, combo boxes etc, to camel case.** |

|  |
| --- |
| Component: C# code |
| Show *trialling* of alternative techniques/components. Explain what you are trialling and justify which techniques you are going to use. How is this going to improve functionality? (Include screenshots) |
| * When people login, I need the user id and their occupation. I could either use Execute Scalar two timer or use Execute Reader to get both: Execute Scalar is very easy to use, however it’s quite a bit of code, with some part being repeated, and it has 2 commands instead of 1: Execute reader is a bit harder to use but once you’ve got the hang of it it’s easier to understand the code. Furthermore, execute reader allows a more flexible approach in case you want more data from the database. * I could have an if else statement to know whether, the textbox or password box is currently used and add the right parameter, to the MySQL code:  This is rather a long way of doing it, it works fine, but it can look a bit confusing. Alternatively, I could use a statement that checks the box, so that the password box is always correct, so that I don’t need the if-else statement:  The second option, is easier to use and understand, as it’s clear that I uncheck the box and as a result use the password box to get the password (especially when I add comments). Hence, I’m going with the unchecking the box as this way I have less code and it’s easier to understand, for future proofing, what’s happening, because there are less statements and you don’t have to think about how it works. * I need to store id\_user if it’s a student or teacher because the ticket needs to know their username and occupation. I could either make iduser a public variable as shown below and set it to the value I want:  However, I could also use an extra 2 labels, one saying ‘User # and one displaying their user id.  This approach allows the user will know whether they are logged into the right account and I’ll have less global variables. Hence, adding labels is better since it has better coding styl and the user can check if they have logged in correctly. * I have two logout buttons, 1 on the View Tickets Tab and one on the Report Ticket tab. I could use 2 separate method that reset everything: However, I could use 1 method for bout logut button, so that there is only 1 method for logging out:   This allows me to have less repeated code and it’s clear that it does the same thing. Hence, using 1 method is beneficial because less code and the outline is clearer and if you need to change the logout method slightly it’s easier to do it once rather than twice. So, I’m using the singular method.   * For the View Ticket apply changes, the user needs to be able to change both the category and who’s assigned. I could either go with 1 change button or 2 with 1 for each. Apply changes with 1 button, I need to set the combo boxes automatically to the selected row: Nevertheless, I could also use two apply changes button that changes one column at a time with 2 functions: Using 1 change button is preferable because it doesn’t need as much code and it only needs 1 method, which prevents some repeated code. Also, this will allow the user to change both at the same time if wanted, and it gives a better overall overview of what is in the database. * When a ticket is deleted, I want the associated detail row also deleted. I could either use two delete statements or 1 using inner join. With two delete statements I will need to command and run it, with the ticket running first due to the foreign key constrain then the detail: This isn’t great as it requires some code to store the foreign key needed. So, I needed to have 3 commands. However, I could also use just one command: This is easy to use as it deletes the right rows with 1 statement, and I don’t have to store any values. Hence, going with INNER JOIN DELETE statement is easier because I only need 1 command and it’s easier to understand what’s going on. Furthermore, for future proofing this is the better option if you want to change something, you only must change 1 statement as opposed to 3. * For my commands, I could either use {} (curly brackets) or parameters that I add in later. For the once with parameter I could use @name, and then use Add With Value to add the needed value: This is a method, that’s useful if you don’t know what the parameter is, however, if you know what the paramers will be, it’s a bit finicky, as you have to use a standin name and then define it, even though you already know what it is. With the {}, the problem is that if you don’t know what the value will be, it’s isn’t great, however, in this case I can just move the line of checking the checkbox up, so that I can set the parameter already:  Hence it’s better to use the {} as it is less code and when someone else reads, they know what you’re using rather than having to go back in forth to see what each thing does. |
| Show *testing* of your selected technique/component. What did you find out? What do you need to do as a result? How have you improved your design? |
| * Execute Scalar works just fine. It selects the correct occupation, and then redirects to the correct tab. Same goes for execute reader. However, with execute reader, I need to ensure that only 1 user is selected, as otherwise I might get the wrong user. As a result, I added in an extra column and only go to the appropriate screen if 1 is selected: * Both the if-else statement and checking the password check box work. They both allow the user to login, and then redirect to the right screen, without any problems. I tested the second method by checking the box and entering a user and see if it works, and it did. * Using a public iduser variable works, it gives the correct user when I report the ticket and double check in the database. I logged in as Matt (#3) reports a nonsense ticket and checked the database: (details)(tickets) Also, as seen above (in the trialling box) storing the number in a label also works and this will also give me the right user when the ticket is reported. * Both the logout button methods work i.e., reset the wanted screen, I checked first by just letting the code run, then I temporarily removed the enabled=false line to double check whether everything was truly removed. Furthermore, the singular method for both buttons also work, as they also reset everything. I double checked my temporarily removing the line that disabled the screens. So, both methods work, only using 1 method is ideal as otherwise the code is repetitive and this way if you something needs changing, you only must look at 1 function. * The 2 apply changes methods work. When you click the button, it’s clearly visible on the DataGrid that the column changes, and hence the methods work. Same goes for using 1 method, it’s shown what changes. However, with the change Staff combo box, the value isn’t always seen since the value isn’t an option, unassigned. Hence, I’ve decided to add an extra item to the box: * The 2 delete statements work. The first one with the delete ticket, obviously works as it is immediately removed from the database. I checked the 2nd delete statement of details by going to the database and checking if it was deleted. This was the case. With INNER JOIN, the ticket is obviously deleted as it isn’t seen in the database anymore. The details are also deleted, as I have checked in the database itself. Both are great options, however, 1 statement is better to handle then 3. * Both Parameters.AddWithValue and {} in the command works, as it doesn’t crash and still does everything right. However {} is a better option as this makes the code more readily, in my opinion, and it’s clearer what everything does/is. |
| What implications are relevant to this component, and why? |
| * Functionality: The program needs to work without crashing, so it needs to function to be usable. * Future proofing: This program might be used in the future, so I need to ensure that the code is understandable and that everything is clear. * Usability: the program needs to be usable as if it’s hard to use no one wants to report a ticket, which is great for admin, less work, but frustrating for |
| How did I address these implications? |
| * Functionality: I tested this part of the program with all the new things added. I ensured that it worked as it supposed, and if it didn’t work, I would troubleshoot and find the problem and fix it, so that it functions perfectly. * Future proofing: The program needs to be able to be adjusted easily so that it if changes are needed, it’s easy to do so. Hence, when I was trialling, I thought about what way would be easier to understand/change the program. This would make, it easier for future programmers. Furthermore, I added comments to ensure that other programmers know what each part of the program does, so they can modify it easier. * Usability: I ensured that everything is working, in the program and that’s easily usable and everyone knows what everything does by ensuring the text is clear and that the methods work, so that the buttons are usable. |
| My project management, version control tools, and any feedback on your program at this point in the project.: |
| Feedback: I had another student use the program; however, they couldn’t come up with it. Further user-testing will also be done in the next section.  For version control refer to the document called subversion\_log\_3.3\_3.8. |

|  |
| --- |
| **DOUBLE CHECK IF MY TICKET TABLE IS CORRECT** |

|  |
| --- |
| **3. Finishing up** |

**Final testing:**

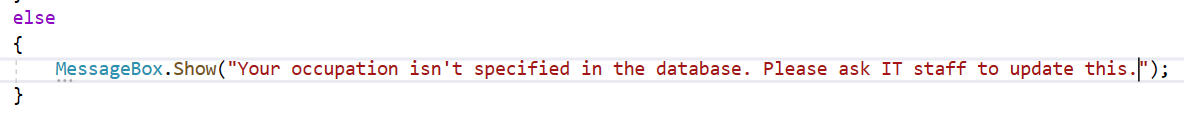
What **user testing** did I do on my completed outcome?

1. As user 1 was testing, the C# program, they noticed that the DataGrid doesn’t get updated if you login, logout, and login: Graphical user interface

   Description automatically generated with medium confidence Table

   Description automatically generated with medium confidenceHence I edited the code that would reset the combo boxes and description box after logging out. This made me wonder if this also happens if I change the view and/or close a ticket:  Graphical user interface, text, application

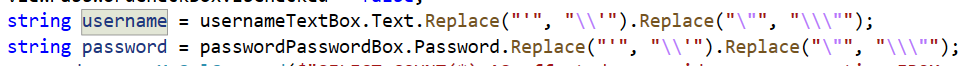
   Description automatically generatedSo I just had to ensure that I reset the description, category and assigned staff in the filldatagrid method: Text

   Description automatically generated. Also, they noticed that if I delete a ticket, the numbers are out of order, and my program crashes due to having a duplicate primary key with my id\_detail/id\_ticket so I removed the counting the rows. Now the columns just auto increment, has that box is checked in the database.
2. I allowed user 2 to add in new users. The problem that they noticed was that there are no restrictions on occupation i.e., you can fill in anything and the C# program would go to the report issue tab since it says, ‘if occupation = admin, else’. Hence, I needed to add in a restriction, or give feedback in the C# program: 
3. My 3rd user added in apostrophes into the summary. However, since were using MySQL, apostrophes make the program crash: A picture containing text

   Description automatically generatedText

   Description automatically generated This led to checking if it likes emojis, since students, and teachers, might use them: emojis. The emojis also made the program crash. To fix the apostrophe I used replace to insert a back slash before the apostrophe or quote so it would ignore it: Now I rechecked for my summary and description and now it does accept ‘ and “:A picture containing text

   Description automatically generatedHowever, this led me to think about the login page as the user might want to insert those as well. So, I double checked this, and it crashed: Text

   Description automatically generated. Hence, I used .Replace once again to ignore the apostrophes and “: . For the emoji part, I could either delete them from the string itself, however, they are there to help the admin understand the problem. The other alternative is going into the MySQL database and setting the default character set to utf8mb4 instead of utf8, since utf8mb4 supports emojis. Hence, I’ve decided to change the characters, which allows emojis in varchar, I will set this for the users table as well as the details table so that usernames, passwords, summaries, and description can all emojis: Graphical user interface, text, application

   Description automatically generatedFinally, I had to set the charset in C# as well. Now it would accept emojis 👍👌: Graphical user interface

   Description automatically generated

What **functional testing** did I do on my completed outcome?

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Purpose of the test** | **Data entry for the test** | | | **Expected Result** | **Actual Result** | **Outcome of Test** |
| Testing Login page | | | | | | |
| Expected Inputs | | | | | | |
| An admin user | gabe, admin | Go to the view ticket tab and disable the other 2 tabs | | |  | Pass |
| A student/ teacher user | yyy, yyy | Go the report ticket tab, disable other 2 tabs, display user #5 | | |  | Pass |
| Boundary inputs | | | | | | |
| Putting in the hashed password | ‘student’ hashed is: 264c8c381bf16c982a4e59b0dd4c6f7808c51a05f64c35db42cc78a2a72875bb | | | Show a message box that says wrong username/password and reset the password box |  | Fail |
|  | Pass  (Forgot to empty the box) |
| Username with a different password | matt, admin | | |  | Pass |
| Different cases for username | mAtT, student | | |  | Pass |
| Having the checkbox unchecked | strang, teacher | | | Move onto the report ticket tab, disable the tabs, and say user #3 for matt and #2 for strang |  | Pass |
| Emojis in the textbox - switch to password box | 👀👀, in the password box add abc | | | Just have the same input regardless. |  | Pass  Note that password box has 4 characters. That is because 1 emoji = 2 chars |
| Unexpected Inputs | | | | | | |
| Random letters for both entries | abcd, zyxw | | | Show a message box that says wrong username/password and reset the password box  Go the report issue tab |  | Pass |
| Just numbers | 1234, 5678 | | |  | Pass |
| Testing Report Issue page (and triggers) | | | | | | |
| Expected Inputs | | | | | | |
| A decent summary (~80 char) with a description/report ticket | Alphabet with space\*3, alphabet\*1 | | | Give a message saying ticket reported + added to the database + emptied boxes |  | Fail  (Missed .Text) |
| Pass |
| Logout button i.e., press the button | Clicking it should disable the tab and return to an enabled login screen with empty boxes. | | | |  | Pass |
| Boundary inputs | | | | | | |
| 49 chars in summary | alphabet\*2 minus xyz, nothing (‘’) | | | Message asking for it to be longer than 50 char and keep text |  | Pass |
| 50 chars in summary | Alphabet\*2 minus yz, nothing (‘’) | | | Message say ticket reported and add to database + empty boxes |  | Pass  (Note that the default value for assigned column as well as the trigger for description works) |
| Fill summary + description | 200 char, 750 char. | | |  | Pass |
| Duplicate summary/description | Alphabet\*2 minus yz, 750 char (same as above) | | |  | Pass |
| Adding emojis into summary/ description | 👀\*60, 👀\*60 | | | Should work, i.e., report the ticket and empty out the boxes. |  | Pass |
| Adding between 100 and 200 in summary | 👀\*150 | | | It only allows for me to enter 100 👀. This is because C# counts emojis as 2 chars. | Fail |
|  | Pass  It allowed me to enter the 6 lines of emojis (25 per line). Note that I used enumarateRunes.Count instead of Length as length counts the emojis as two characters. Note that I added an auto scrollbar to the summary as well as a result. |
| Unexpected Inputs | | | | | | |
| No summary | ‘’, abcde | | | Message asking for it to be longer than 50 char and keep text |  | Pass |
| Number of emojis is above 25 but below 50 for summary | 👀\*30 | | |  | Pass |
| 200+ chars in summary and 750+ char for description | Alphabet\*8, Alphabet\*30 | | | Give an error message saying that there is too many characters \*2 |  | Pass |
| **Note I deleted all the new ‘tickets’ I made during testing to clean up the database.** | | | | | | |
| Testing View Ticket page | | | | | | |
| When I click on a row, description and combo boxes get updated. I will just be highlighting a row. | | |  | | | Pass |
| Apply changes button that will change the row. xxx, network | | |  | | | Pass |
| Filtering view, I’ll be filtering by unassigned and going back to all: | | |  | | | Pass |
| Testing close ticket:  Will be removing the 3rd row. Should delete the ticket, and detail from the database and say ticket closed. | | |  | | | Pass |
| If I logout and login again everything is reset. | | |  | | | Pass |

**How did planning assist in the development of a high-quality outcome?**

Planning allowed me to consider what I want the program to look like at the end. It allowed me to organize my thoughts so that it would turn into an organized project. Furthermore, Trello (Kanban Board), allowed me to layout the task I had to do for each component, and was a visual for me to see how much I had left to do, so I was able to complete the project with enough time for thorough testing to allow for a high-quality outcome. It also allowed me stay ahead of the project, so that I could take a break during the winter holidays so that I could have a fresh mind, going into the 2nd half of the project. This allowed me to think of several new ways I could approach a specific part, which would as a result improve the program. Moreover, I got sick for a week, and didn’t have to work on it because I had my Kanban board and knew I still had enough time to do the C# coding. This ensured that I didn’t have to code while being sick, which allowed me to focus on it and not make any errors, rather than being sick and writing bad code, that could lead to problems later. Lastly, planning allowed me to set up the Kanban board and as result I was able to structure my program and make it a high-quality outcome by ensuring everything worked together. Subversion allowed me to track back to what I’d already done so that when I took a break I knew where I left off, so I had fresh and old ideas working together to ensure that the best choices were made to complete a high-quality project. In conclusion, planning allowed me to take break and come back with fresh ideas, improving the program, it allowed me to create a structured program that has a logical flow, and hence a high-quality outcome, and it ensured I was on track so I didn’t have to rush things, that would leave the outcome uncompleted.

**How did testing and trialling assist in the development of a high-quality outcome?**

Firstly, testing and trialling allowed me to try out different techniques for the same part, like default expression vs triggers in the database, to ensure I got the most efficient and logical technique for every part of the program, resulting in a high-quality outcome. Secondly, I have done testing of every part to ensure that there are no bugs whatsoever in the program, so that it works without a flaw, because if I didn’t do any testing, the program for example would crash with apostrophes or the number of characters including emojis would be wrong, resulting in a high-quality outcome. Moreover, through code convention, allowed me to ensure my code has comments, follows all the basic programming rules like no magic numbers, like MinSummary, MaxSummary and MaxDescription, ensuring every part makes sense and is easy to change if needed, which creates a high-quality outcome. Finally, through continuous seeking of end-use feedback, by letting them test, I was able to create a program that is both pleasing for the end-user to use and ensures that everything works without bugs, like not assigning numbers to id\_detail/id\_ticket, instead let the database do it. It also allowed me to make some minor changes that would improve the program, by functionality, usability, and aesthetics, like at the start with seeking feedback for the original layout of the 3 screens, for the end-user. It allowed for a pleasing outcome without any errors, making a high-quality outcome.