OCR GCE A LEVEL COMPUTER SCIENCE PROGRAMMING PROJECT H446-03

The barbershop reservation system

Analysis

Introduction:

My client, Liam smith has asked me to create a booking system for his barber shop. This software will allow customer to sign in and book an appointment for a haircut.

The program will begin to prompt the customer to login or create an account. The reasons why I have accounts is to allow the customer to save preferences for future appointments. This will save their email, phone number and other preferences regarding their hairstyles. If the inputs match the details within the database, they will be promoted with to enter a security code that will be sent their email address. After login in, the customer will be directed to the main menu, the customer will have an option to create a new reservation and another button will allow the user to view or delete a reservation through a simple button.

If the customer decides to create a reservation, the software on the customer side, will present with the barbers that are available. After picking the barber, they will choose a from a broad range of hairstyles they want with the options to upload an image of their hair and an image of the haircut they want. Also on another page will be a message box that allows the user to input additional information the barber may want to know. If the customer is unsure, the program will ask the customer a series of multiple-choice questions to reduce the number of possibilities and give a recommendation with inputs from the customer. Some examples of questions my programs may ask is like what hair type or what head shape they have. This data will be used to pick a recommended hairstyle and show some photos of hairstyles as an example. The styles will be picked from the previous selections presented. My client wanted this feature to further differentiate this booking system from other similar barber shops as he believes customers will utilise this feature and further make the process efficient for his barbershop. The customer will now have a choice of the recommended hairstyle or something else they may like. The program will then present the customer with available times that the selected barber is free. At the end will be a confirmation screen with the barber selected, hairstyle, pricing, and any images the customer has uploaded. After the reservation has been created, the program will then automatically send a receipt with all the reservation details to the customers email address.

On the barber side, the software will prompt the barber to login to their account. After logging in they will be presented with their calendar. This calendar will show which clients are due, with the ability to click on their tab. The tab will then show the customers personal details like phone number and email. The tab will also include the details of the haircut and the image the user has uploaded as reference. One of the other barbers in the shop, Lucas, wanted the tab to include the client's personal detail because they will then have the ability to contact their customer if they have any questions or if the barber is not going to be in on that day.

This system will replace a paper-based system where the customer would normally have to call the barbershop and any barber available would have to manually input the information into a ledger. This system had many issues that I intend to solve. The issues include barbers during haircuts having to pick up calls that last on average 5 mins further wasting time for both the barber and the customer getting their haircuts, customers unsure of what hairstyle they want leading them to be unsatisfied with their haircut. Also preventing any human error with barbers inputting the data wrong into the ledger further causing delays and confusion.

My client, Liam, has especially requested that my system remind customers of their reservation shortly before their reservation as the main cause of lost business is customer forgetting their haircuts or coming late. My program will ensure that this does not happen as they will remind customer prior while warning them of a late fee if they turn up late. If a client happens to miss an appointment without cancelling, their next reservation will automatically be charged extra as a missed reservation fee.

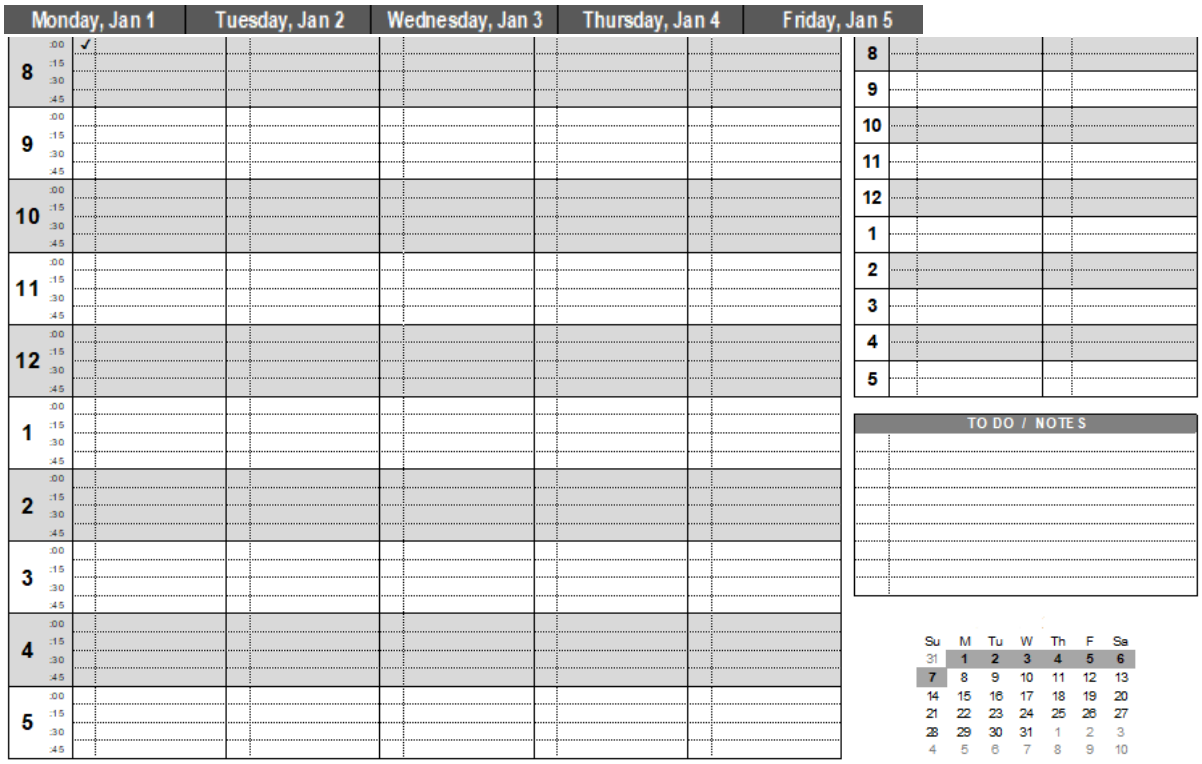
I aim to program the system by using python programming language and AppJar, whilst also incorporating SQL to interact, access and manipulate the database. I will need to update the database with the reservations made by using SQL to ensure that the barber is unavailable at that specific booked time. This also allows me to save client details while also being able to modify or review at any time. I will use AppJar to create the interface which will ensure that the user will be able to easily create a reservation. My aim is to make my program an intuitive user centred design that will require little understanding to use. A minimal interface with clear linear process that is simple to follow and understand. This will benefit both customer and barber as this design will reduce any further human error and make the entire process as quick and efficient as possible.

How is this computationally solvable?

I know this is computationally solvable and it will be the best approach compared to the telephone/paper-based system as it comes with many problems such as human error. To ensure my program is flexible and prevent it from becoming redundant, it is essential that I think ahead as my program will reflect real life scenarios which is dynamic. Using computation methods will allow me to a superior solution than the telephone/paper-based system.

Computation method	How will this be used?
Think abstractly	I will use abstraction to streamline the booking process as the user does not need to know the all the details of the barber shop such as seating arrangement as this does not affect the outcome of the program and should not be included. They will only be presented with essential data such as availability and information with no value will be removed from the design. For example, Barbers available at certain time slots will be visible as this will affect the outcome of the program. I will also have to think abstractly while I create my user interface. Due to my lack of experience and time, I will not be able to create a complex UI, but I will only be able to create a functional simple user interface that still provides all the relevant information that the customer will need to create a reservation easily.
Thinking ahead	To create a functional program, I will have to ensure that all predetermined services/hairstyles will be amended already into the database. This will include information about the service such as the name and pricing etc. More examples of tables I intend to create are an account database for both customers and employees as well as a database on the reservations made. These records will be made during the use of the program, so it does not need to be predefined by me. It is also important to consider future changes made to the services, as otherwise my program will become redundant. To do this, I will have to communicate with my client prior to development to ensure all the essential information is accounted for. To prevent my system from becoming redundant I

	will also be able to accommodate for any changes such as an introduction of a new service/popular hairstyle or pricing changes. Hence, I will make it possible for employees only to make update the services listed easily. Customers should be able to input their account details in and expect the menu page to be outputted if the details match, if not, output error message. If they want to create an account, they must input all details that fit the requirements set, this will output a message depending on if the requirements are met, either creating a new account or rejecting the inputs. Next, input their reservation details that will result in the program outputting the reservation to the corresponding barber calendar.
Thinking procedurally	I will split the problem into separate components. This will include a calendar screen and a booking screen etc. Each of the screen can be split into their own individual function where I can re-use code through inheritance. I will be able to inherit distinct functions from various parts of the system to increase efficiency of my program while also saving a lot of time. Inheriting will help meet my client's deadline while also making it easier for me to create this system. As my program will be built into a modular way, it will be easy to read and follow. This will also allow other developers and myself to maintain/amend the system without affecting the rest of the program as it is split into manageable chunks. I will especially need to think procedurally when validation the system as I am using multiple tables. I will need to ensure referential integrity by ensuring that the user inputs data in the correct format to prevent errors and providing the wrong data.
Thinking logically	It is essential to think logically when developing my program as it ensures the user will be able to reserve a haircut in the most efficient way possible. I will be gathering information about the features that my clients wants so I can meet their requirements and produce a program that functions well for my client. I will be developing my code in a logical order that allows feedback from my client to be fully utilised. Throughout the development process I will keep in close contact from my client gathering further feedback to ensure that my program meets my client's standard.
Thinking concurrently	While my program is running, there will be multiple tasks within the program running concurrently. For example, while my user interface is running, my service database will be accessed and queried with concurrently when obtaining information or updating the database. While my program is running, some parts of my program will also depend on each other. For example, when a reservation is made, another part of the program updates the database which than affects the calendar as the it is no longer available for other users.



Above an example of the template my client uses

Existing system:

The barber shop is currently using a timetable with the work hours on the and days of the week. The barbers each have their own timetable that the print every week and when a reservation is made over the phone the barber will mark their name and time of the appointment on the timetable with the corresponding time and date. All the available times are listed along each row and the weekdays are listed at each column. The barber must amend the booking which can be difficult due to the number of rows/columns. The barbers are prone to error as they usually mark the reservation as they are simultaneously on the call with their customer. The barbers have also made complaints that the timetable can be difficult to navigate and can be time consuming as it would be much quicker for the customer to make the reservation than calling the barbershop, wasting both customers and barbers' time. This is the reason I intend to find faults within the existing system and think of solutions to the new systems to improve efficiency while making the system clear and concise as the current system is overwhelming to the barbers.

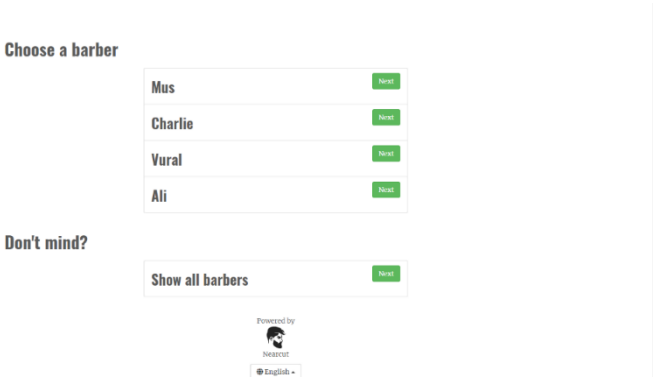
Issues	Drawbacks for my stakeholder	Workable solutions
The layout is difficult to navigate	The barbers are prone to more human error and may schedule customers at wrong times.	Making the software a user centred/accessible design that can be intuitively navigated
Barbers are pulled away from customers, when receiving a phone call for reservations	Barber wastes the customers time as call can lasts up to 5 minutes, also can cause barbers to be behind schedule affecting the next customer	Barber will no longer need to manually mark reservations as it is all done on the client's sides.
Customers being unsure of the haircut/style they want as they sit to get their hair cut	More time is wasted as the customer decides on the chair and it increases the chance of the customer being unhappy with the haircut, causing them to lose the customers business	A recommended haircut/styles that the program decides if the customer is unsure with a series of multiple-choice questions
Customer often may forget/come late to their reservation	Barber losing business if a customer misses a reservation or if a customer come lates, delaying the barbers schedule further affecting future customers	A day before the reservation, an email is sent to the customers email reminding them of their reservation. If they happen to miss their reservation without cancelling their appointment, a fee is applied to their next reservation
Barbers often lose and damage their paper calendar. Ink can often smudge making the text illegible.	If the paper no longer becomes legible, since there is no back up, the barber will lose track of all their reservations causing more confusion for the barber.	A digital copy cannot be damaged/lost as it saved on the database. This prevents reservations being lost allowing the barber to continue stress free.

Analysing existing software:

1) **Legends barber shop website**

This is a website designed for booking a haircut as an alternative system to the telephone/paper-based system. This system is well designed as it does what is intended, it allows users to easily reserve haircut with their favourite barbers but lacks many features that I intend to bring that overall will be bring a better user experience due to the extra features that entice a wider range number of customers.

Choosing the barber: .



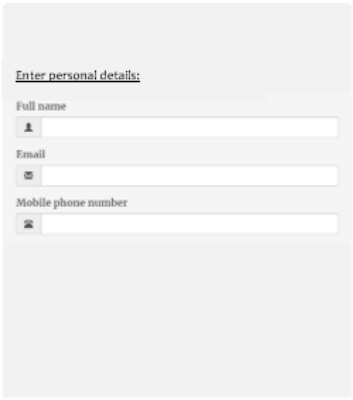
Use:

The first page the customer is presented with a choice, what barber they would like to book a reservation with. They have a choice, their favourite barber, or they can choose any. This is presented in organised way making it clear to users which barber is available.

Is this useful for my program:

My program will benefit from this multiple-choice screen for the customers. This is so they can easily filter what barber they like the most because most people usually go to one barber as they are more comfortable with them while being pleased with the result. This is important to implement into my program as it overall improves customer experience.

Personal information: .



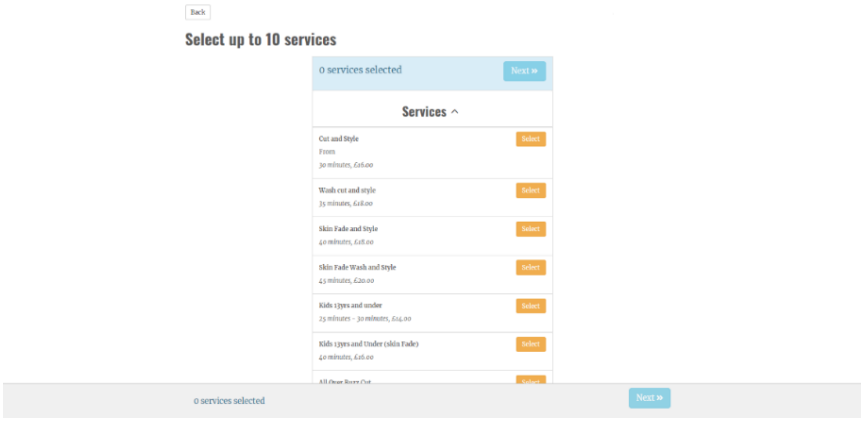
Use:

The next stage presents the customer with a text box to enter their personal details that allow the barber to contact the customer.

Is this useful for my program:

This part is not useful to me as I intend to create an account system where customers can create account to allow them to save preferences, prevents the customer having to go through all the stages of the program to book a haircut if they decide to get the same haircut. With the saved preferences they can quickly book an appointment by only choosing a date/time.

Picking the service:



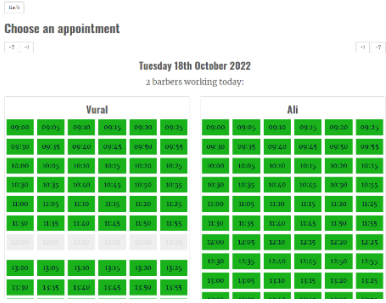
Use:

After entering their personal details, the user must choose what type of services they want, they have a range of options that describes all services provided by the barber shop. Each service has their price listed below with an option to pick multiple different options. This page is presented in a clear and organised manner making it easy for users to choose a service.

Is this useful for my program:

My program will need to provide a list of services available for the customer to choose from. From here my program will differ slightly as I will introduce a feature where the customer has a choice for the program to decide a recommend hairstyles with the answers from the user. This is also an important feature for thinking ahead as to keep the program relevant, the services will need to be modifiable so they can update parts such as pricing. This allows my program to be easily maintained as well as not becoming redundant.

Choosing the slot:



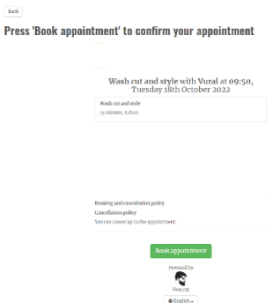
Use:

This allows the customer to input the date and time that works for them. This then updates the calendar on the barber's side and prevents other users from picking the same date and time. It is presented in a clear and colourful way that clearly shows what slots are available in green and different shades that indicate how free that day is. This page is presented in an accessible way as green is usually stands for go/available making it easy for user to quickly identify free days.

Is this useful for my program:

This is essential for my program as customers will have to input what time they want their reservation. I will also display the available dates in a calendar form as it will be most familiar for customers. My program will differ slightly as I will hide the unavailable times from the customer to further abstract as the user does not need to know what times are unavailable, therefore making the program interface less cluttered and making it as concise as possible.

Confirmation screen:



Use:

At the end of the booking process, the customer can review their reservation, confirming that they are happy with the reservation and price. The user clicks confirm, this updates the calendar on both customer/barber side making the slot unavailable. The appearance of this page is concise allowing users to walk away with information they only need, also saving time.

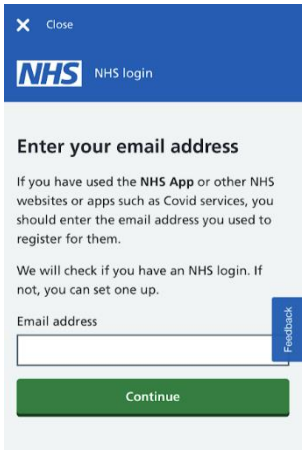
Is this useful for my program:

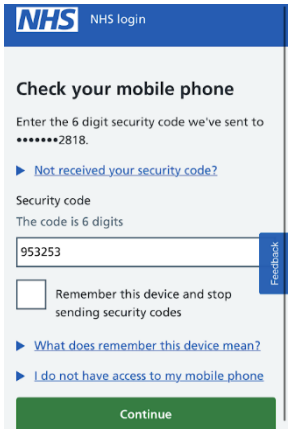
As this will benefit the customers, I will implement this into my program. I will also include the extra information such as the image uploaded so the user can review all the important inputs made. This will also notify the user of an fees due to missing the reservation and that the customer will be notified prior to the reservation.

2) NHS mobile app

This is a mobile app that allows users to view/create appointments for their GP, view their medical records and order repeat prescriptions. This mobile app is like my program as both can book/modify reservations and has a login system.

Login:





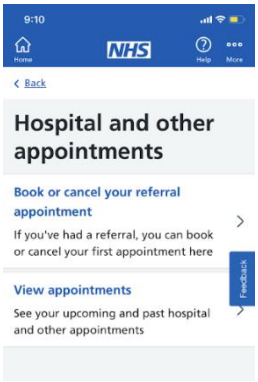
Use:

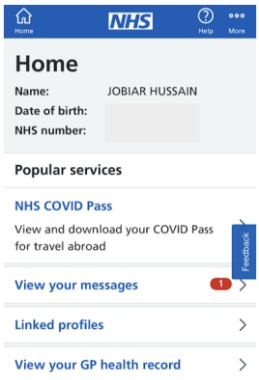
The first screen the user is presented with it the login/register page. The user has the option to login to their NHS account with the detail when they first create an account or create an account. The user is then asked to check their text message for a security code to securely login to their account. The presentation of this information looks cluttered making it overwhelming for some.

Is this useful for my program:

This will be essential for my program as users will be able to login into their own accounts that have their preferences saved while employees will be able to gain access to their own calendar more securely. I will attempt to implement a security code system where the user will be sent a code through their email inbox. Due to lack of skill and time, this feature may not be possible to implement but as my client wanted this feature, I will priorities the two-factor authentication.

Dashboard:





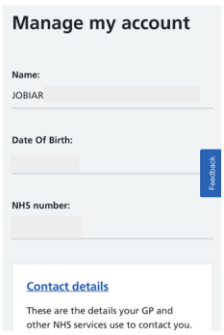
Use:

User through this page can access all the features the NHS app has to offer such as booking/viewing appointment. This allows users to easily navigate the software making it accessible to more users as they no longer must search for every feature.

Is this useful for my program:

As this will also benefit the user, I will implement this feature as it is essential to allow access to all the features the program has to offer on a single window that is easy to access. This will prevent the user from having to jump from window to window looking for features, further creating frustration.

Managing your account:



Use:

This allows the users to edit their details on their account. This ensure relevant details are stored ensuring data integrity. The appearance of this screen is clear and concise also allowing users to easily manage their account.

Is this useful for my program:

This is not essential for my users but is however beneficial for my user. This feature will ensure that barbers will be able to contact the customers with the correct details in case they need to reschedule. If I have enough time, I may implement this to allow customers to update the details within the account database.

Reminders:

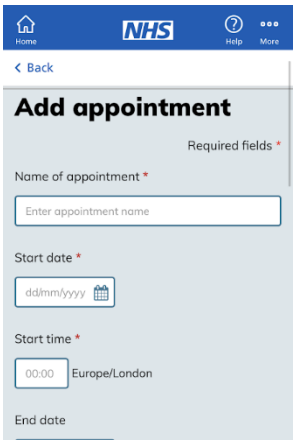
Use:

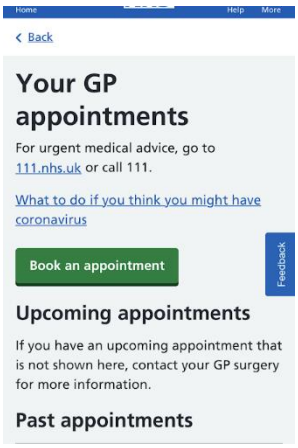
The NHS app will remind user prior to an appointment through text message or email. This ensures that users do not forget their appointment, reducing the amount of missed appointments.

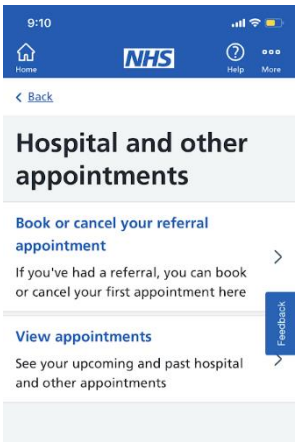
Is this useful for my program:

This is a feature I intend to implement when creating my program as it was requested by my clients. The users will be sent an email by my program before the reservation. This ensures that customers do not forget and prevent frustration from my clients.

Booking appointment: .







Use:

This page allows to users to create new appointment with the option of booking with their GP or the hospital through a menu system. Users can view their current appointments, view past appointments and book/cancel appointments. When creating an appointment, user instead must pick a start/end date/time. This differs from my plan as I intend to create an interactive calendar system where users can visually see available times.

How is this useful for my program:

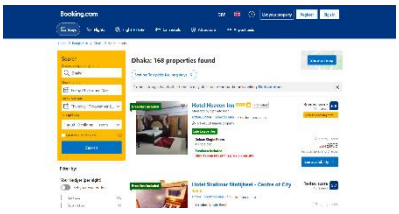
This is essential to my program as my program is based around this. However, some features within the NHS app, I would not be able to implement into my program such as viewing past reservations as I plan on deleting old reservations. My booking system will not use input boxes like the NHS's but instead use an interactive calendar.

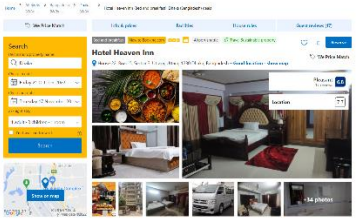
3) Booking.com

This program is designed for booking hotels and homes for users on holidays. This program is primarily designed to book everything a user may need during a holiday such as plane tickets and car rentals. Allows user to easily plan a whole holiday in one website.

Searching/reviews: .

Use:



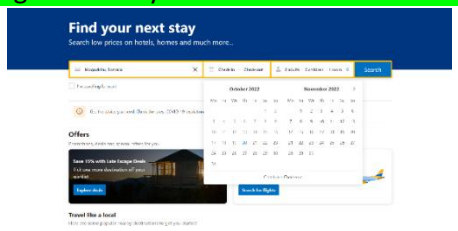


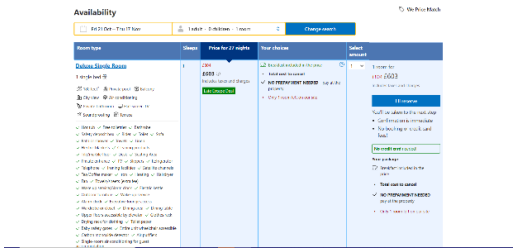
This page allows users to search and compare other hotels with a filter system that can reduce the number of possible choices. The options are filtered by factors such as location and booking period. Each hotel has information listed on the pages, this includes pricing and reviews. The users have the option to also view available hotels on the map allowing users to visually see hotels on the map so they can better understand if the locations are ideal for them. This is presented in a somewhat clear way but is cluttered with too much information/images.

Is this useful for my program:

This isn't as useful as my program job is much simpler. My program however will have a similar page where all the different services are listed but unnecessary information such as the reviews will be removed. My program will not include a filter as it isn't needed when choosing a reservation at a single barbershop.

Checking availability:





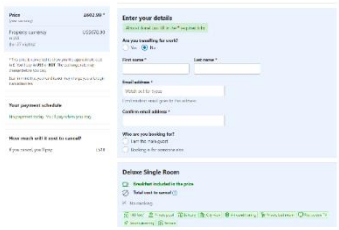
Use:

On these pages, the user chooses the date, with a clear interactive calendar, that they plan to stay at. The program then checks the availability for the hotels picked. On the second page all the services and bundles are listed allowing the users to learn more about the hotel services/pricing and see if it will work with them. This is presented in a confusing way to the user as they are overwhelmed with information.

Is this useful for my program:

This is needed for my program as the customers will need to pick an available time for their reservation. My program will also have to check with the database to see whether if a picked time is available. The additional information will be abstracted from my program as it will end up slowing my user down with information that is unnecessary.

Checkout:



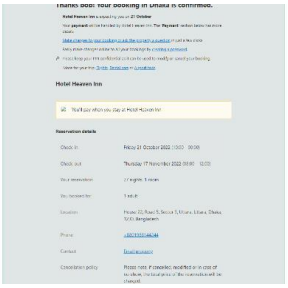
Use:

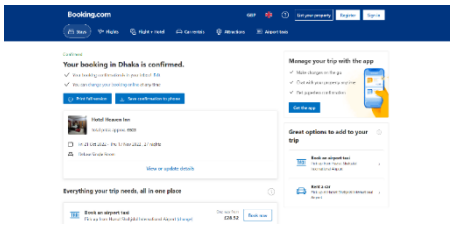
The first page ensures the reservation is checked out with the personal details of the users. The user must input the required fields prior to continuing to prevent any errors own the line. This interface is again cluttered making it even more overwhelming for users.

Is this useful for my program:

A checkout system isn't needed for my program as my users will be creating accounts that already save their personal details. This prevents them from repeatedly inputting the same data and therefore saving more time for the user. Also, no bank details are saved as users pay on the day.

Confirmation screen:





Use:

This page presents users with the summary of their reservation. This provides all the information regarding the reservation allowing the user to review the purchase to see if any changes are needed to be made.

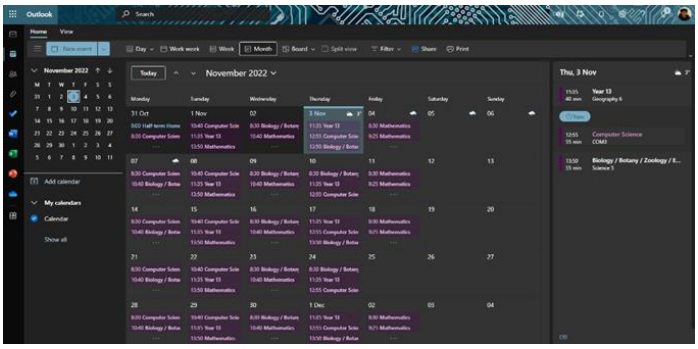
Is this useful for my program:

This feature isn't essential for my program, but I will however still include this. This is to prevent any more human error by giving the user a chance to find any mistakes and prevent unwanted reservations.

4) Outlook calendar

This is a pre-installed software on windows that allows users to create task and keep track of their schedule/calendar. This pre-installed app will have similar features to my calendar system where employees can view their schedule.

Calendar screen:



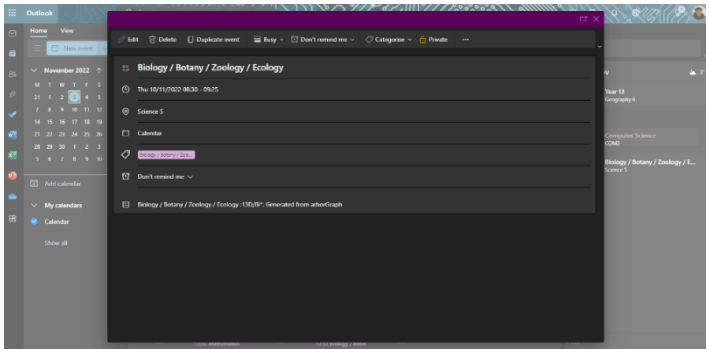
Use:

This page displays the month graphically with each day having a box with the users' schedule/tasks. This provides all the necessary information the user may need. For example, in a school environment, a student's calendar will present information regarding their timetable. This will only show abstracted information such as the lesson the student has, room number and time of day their lesson is in. Removing unnecessary information such as the number of students in the class to allow the user to quickly gather the information they need at a glance.

Is this useful for my program:

This feature is important for my program as I will present a calendar to my users and employees. This will be most useful for my employees as they will be able to easily see when they have their next reservation at a glance. Instead of a monthly view, instead I will do a weekly or daily view of their schedule as barbers will usually not need to see their reservations for the next month, further abstracting the information allowing the barber to quickly see information they need.

Addition information / drop down box:



Use:

This box appears when a user clicks on a specific task on an any given day. This box provides all the additional information that a user may want regarding the task, allowing the user to still have access of all the information needed while still having a clear/abstracted calendar.

Is this useful for my program:

This will be useful for my program as the barbers should be able to see the additional information that the customer has inputted during reservation. Instead of a drop-down box, I will do simpler button on each reservation that directs the barber to the information the customer sent, such as the image upload and the message the customer left in the message box. This feature will be added as the barber will not need this additional information at a glance at their reservation further keeping the calendar tidy.

Key features I plan to include:

Discussing with my client, I will outline all the features in my program that satisfy his needs and realistic for me to achieve. The features revolve around the database and different screens/pages.

Database:

-Reservation table with all the reservations that were created by customers. This is needed to ensure referential integrity, so no double reservation is made at the same time.

-A table with all the services available for the barber shop that allows the customers to choose between. This will need to be modifiable so that the barber shop can modify the prices and the services available.

-User table with all the personal detail required to make an account. Allows the customers to reuse their account while saving their preferences.

-Employees details will be saved within the user table where a field will confirm if the record is an employee

-Employees will be able to easily modify the service database to keep up with current trends and prices

Pages/screens:

Account system:

-There will be an account system to for employees that ensures only authorised staff can access my program. This will allow them to view their calendar, customers upload images, customer personal details and other relevant information needed.

-There will also be another account system for customers where they can save their preferences for reservations. This will allow them to view their reservation details while having the option to modify/delete their reservation.

-The page will include two input boxes for the username and password with another button down below to transfer employees to their login system that will also consist of two input boxes. This will be compared to a database and if correct allow them to access the rest of the program. Email system to confirm identity.

Menu page:

-The customer will be presented with two options on this page

-One button will direct the user to create a reservation

-The other button will direct the user to view /cancel current reservation

Choosing the barber page:

-This page will allow customer to choose their favourite barber or having the option to choose them all. These buttons will direct the user to the available reservations for that barber.

Picking the service:

-The customer will be presented with all the service available with the prices listed blow them. Below each service with be a brief description of what the service is. Each service will have a button that selects that service allowing the customer to choose multiple different services.

-If the user is unsure, an unsure button will be below directing the user to the multiple-choice question that will then present the user with a recommended hairstyle. They then have a choice if they want to choose the recommended hairstyle or one of the other services.

Booking page:

-The days available will be presented to the user allowing user to pick their specific day. They will be then directed and presented with all the times slots available to that day.

-The boxes may be coloured to make it clear to the user which days are the most busy and which days are free. Once they choose a day/time slot they will have to press confirm to continue

Additional information:

-The customer will be presented with a text box where they can add additional information about the haircut to get a better haircut.

-They will also have an option to add an image of the hairstyle that the customer wants. But if the user wishes to continue without any additional information, they can continue to confirmation screen.

Confirmation page:

-At the end, all the relevant information will be presented such as the service, price and the additional information and the option to go back to previous pages to make edits

-After the customer reviews the reservation, they will have to press a confirm button and once it is clicked, this will amend the database, by updating or deleting a record in the reservation table.

Calendar:

-For both barbers and users, a calendar system where employees will have be able to view all their reservation in that week or day. Another button in the calendar that directs the employees to a page with all additional information that customers input.

-Customers will be able to visually see what days/time slots are available in the calendar when making a reservation

Limitations:

This project will come with multiple limitations including:

-Lack of time: As this project has a limited amount of time due to my client's deadline, I will not have enough time to iterate on my program multiple times to create the most useful/efficient program that I possibly can. To meet my deadline, I have decided to take a rapid application development methodology. This will allow me to gain lots of feedback from my client which will help me create a program with great usability.

-Lack of skill: As I have little experience programming, it will be a challenge for me to successfully produce a program with all the features I intend to add. This added with my client's deadline might have to force me to cut a few features leading to a functional program with a few requirements not being present.

-Simple user interface: Due to the deadline, I will not have enough time to fully perfect/polish the user interface. As the user interface would not be too aesthetic, I am planning to lean into the idea of a simplistic user interface and therefore embracing it and focusing more of my time other parts of the program. This may lead to my user interface look more bare bone but as my focus is to perfect the usability to the best of my ability, by prioritizing on easy navigation.

-Reminder system: implementing a reminder system where the customer will receive an email prior to the reservation may be difficult for me to implement due to my lack to skill. But as this was a requirement that my client made; I am going to try implement this into my program as this was the main cause of loss of business.

-Lack of encryption: Again, due to lack of time and skill, I most likely will not have enough time to implement or enough skill. This will lead to the system to be less secure as if an unauthorised person gains access to the database, they will be able to cause major harm to the program.

-Resetting password: if an employee or customer forgets their password, they will need a way to recover and create a new password. I may not have time to implement this feature.

Computer requirements:

-These systems will be able to run on most low-end pc's/laptops. This will allow my program to be as assessable as possible to customers. To run python the requirements are:

•34/64-bit processors or 1GHz

•1GB of disk space

•Compatible with Windows 7 and later, macOS and Linux

•Installed python packages onto the system

All the barbers have access to a Lenovo IdeaPad flex which has 2.8 GHz intel Celeron N4020 processor with 4GB of RAM. This laptop comes equipped with windows eleven with 64GB of secondary storage that is more than enough to download Python. This computer system is more than capable to run this program and will be accessible to most customers.

Success criteria

No.	Criteria name	Explanation	How will I ensure this	Quantitative or qualitive?	Achieved by prototype	Importance
1	Functional account system/user login	The system should prompt the user for login details, or they have the option to register. The details will be validated by the program and certain access rights to different pages will be given depending on the user. 2 factor authentication to verify their identity.	I will save all accounts made within a database and when an attempt to login is made, it will be compared to the data within the database. Have a separate button for employees only, that prompts for employees account details instead that then gives access to the program	Quantitative as I will test the system by attempting to gain access without the correct password	1/2	
2	Follow the design language of the barber shop	The program will imitate the barber shop mainly through colour and font. This will help aesthetically to produce a seamless transition from the program to the barbershop itself.	I will use the same colour pallet that the barbershop uses from its logo and from the interior design of the barbershop. This was requested by my client as he likes the barbershops aesthetics	Qualitative as it will be approved by my client	3	

3	Functional system to make a reservation	The customers should be able to create a reservation for a haircut with a date and timeslot they are happy with	I will make the system intuitive where the available time slots are displayed only without timeslots that are already been booked out. This will be done to make a more concise user interface and avoid confusion.	Qualitive as the client has to be happy with the page layout	1	
4	Database containing all accounts, reservations, and services	Multiple tables within the database will be made to store the data such as account details and reservations.	Using database languages such as SQL to create and amend data into the database. I will regularly contact with the client to ensure the data being saved is relevant and not excessive.	Quantitative as to ensure these function as intended, I will have to test the program	1	
5	Process to find a recommended hairstyle	If a customer is unsure of the hairstyle they want, they have the option to answer a series of questions designed pick a recommended style based on my client's feedback	I will create questions to learn more about the customer. This can be questions about face shape and hair type that will be used to determine a type of hairstyle. The hairstyles will be from the services database and each hairstyle will have tags that will be used to reduce the number of possibilities of hairstyles.	Quantitative as I need to test each path and see if they produce a sensible result	2	
6	Employees have the ability to update the services database	The employees should be able to update cells such as pricing and be able to add new hairstyles to keep up with the latest trends and prices. Prevent the	The employees will be able to update the database with a series of questions that my program will have. This will prevent the employees needing to know SQL, so they will be able to use the	Quantitative as I will need to test If the database will save the updates made to it properly	2	

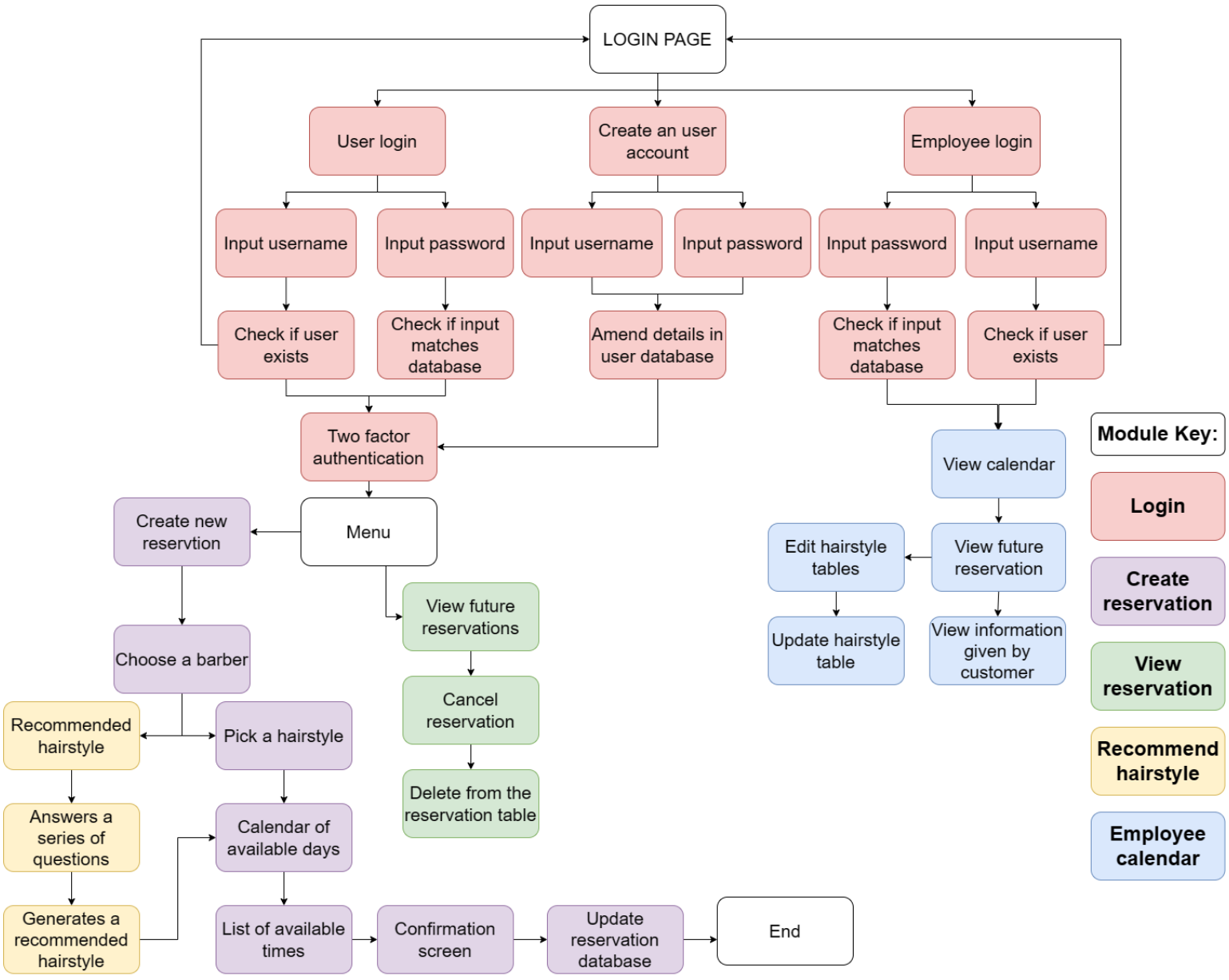
		code becoming redundant.	database with little training			
7	Able to run on employee's hardware and majority of customers hardware as well	The program must at least run on the barbers' laptops so they can access the software	I will ensure that the program doesn't require unnecessary computational power by making the code as efficient as possible	Quantitative as it must be tested on a range of hardware including the barbers' laptops to see if the program works as intended	1	
8	Program able to prevent double bookings in the same timeslot	After selecting the options during the reservation process, the program should prevent the user from being able to book an unavailable timeslot	I will create a function that will filter out all the unavailable time slots so it can prevent being shown to the customer. I will ensure by attempting to create reservations with unavailable times	Quantitative as it must be tested if that timeslot become unavailable to users once someone's books and reservation	1	
9	A dashboard /menu for both customers and barbers	An option to go to the homepage where the customers can view current reservations, or be directed to create a new reservation	I will create a button that allows user to be directed to the homepage where they can navigate to their current reservations	Quantitative as I will need to test if the correct features/window is being presented to the user	1	
10	Ability to cancel their reservation	While viewing their current reservations having the ability to cancel their reservation to avoid a missed reservation fee	The customers will be able to update the database with a delete button in my program. This will now allow other customers to pick that time slot as it is now available. This will prevent the customers needing to know SQL, therefore being more accessible	Quantitative as I will need to test that only that record gets deleted, and no other is affected.	2	

Design

My program is going to be made up of five distinct functions. This includes a login page system. These allow users to sign up/login to the main menu. On the menu page there are three distinct functions, one essential and two non-essential. The essential function in the menu page will create a new reservation where the other two non-essential functions will allow customers to cancel/view future reservation. The last function will allow users to pick a recommended hairstyle based on the users' inputs.

Top-down diagram:

This is my top-down diagram designed to show each function and each subsequent subroutine in the same colour as the function. It is a decomposed view of my whole program and further decomposition of each individual function. In the corner there's is a key with the name of the function coloured coded with the diagram.



Why is each chunk justified?

-Login pages: This page will be further decomposed into three smaller parts. This will be the first page for the users. This ensure that users access further pages that they have authority for and prevent any security risks as it ensures the system is reliably maintained. This also ensures only registered users' login successfully to access the system, next users will be presented with 3 buttons; User login, create new account, employee login. Once the user chooses their options will be then presented with a minimal screen with two input boxes and a couple more for creating a new account.

-Reservations: Users can book a reservation/view reservation/cancel reservation after successfully logging in. Each function will be separated clearly, presented by buttons giving clarity for the user while also making it easier to navigate and for future developers to understand/edit the program.

-Viewing reservation: A function that reads a reservation that matched with the customer's detail. Presenting inputs, the customer made when creating a reservation, such as time and images uploaded. I expect this to be a short and simple function. Responds to changes made to customers reservation. This includes deleting a reservation. It is to ensure that the time slot becomes available for customers when a reservation is cancelled. This ensures data integrity in my database, helping to prevent my program becoming redundant if a customer decides to cancel their reservation. I expect this to be a smaller function as I will need to delete a record only.

-Creating reservations: The core of my program is a function that amends a new reservation to the database. I will need to ensure that saves the inputs the customers makes while also allowing employees to view certain information regarding the reservation, including the hairstyle and images uploaded. It is paramount importance that once a reservation is created, that specific time slot for that barber on that specific day is no longer available for customers after the reservation is created. This is likely to be the largest part of my program that will require the most time to ensure it functions correctly as it is essentially the whole function of my program.

-Recommended hairstyle: This non-essential feature will have its own separate function. Each hairstyles will need to have tags describing the hairstyle, an examples of tags could be long hair, short hair, narrow face shape. Reducing number of tags and presented list of hairstyles through the of process of elimination. I will have to amend these tags within the hairstyles database. I don't expect this function to require too much time, but I may take longer due to lack of knowledge

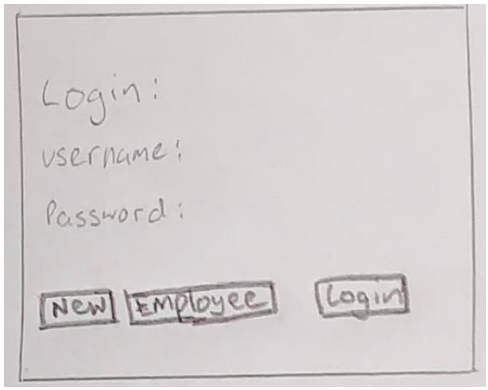
Initial designs:

On the next page, I begin creating initial sketches for the user interface for each page of my program. I will take an iterative design approach where after I draw rough sketch, I consult with my client, and he provides with feedback for each page. After receiving my client's crucial feedback, I will further iterate on the design once and repeat the process after my client is pleased with the final design. This feedback will prevent me from fixating on my design and help me achieve a user centred design. Due time constraints, I will only have time to create 2 iterations and then a final design.

1st iterations

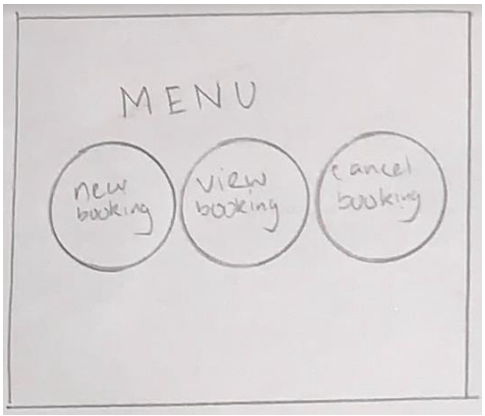
Login screen:

This is my first sketch for the login screen. My initial approach was to keep the design as simple as possible. It consisted of 3 button that read: new user, employee login and login. On top, there's a 2 input boxes with username and password written above them. I presented this to my client, and he requested that the barbershop logo will be present in the top left corner. He also asked that the colour palette will match the barber shops aesthetics providing a uniform design to the barbershop.



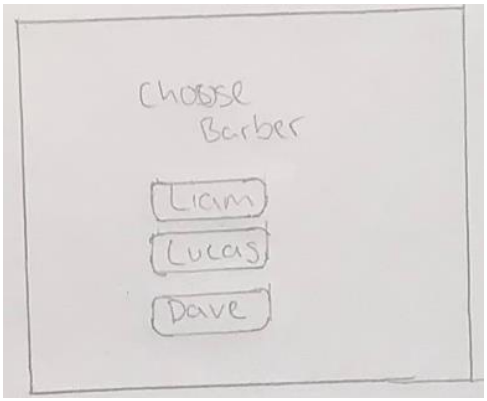
Main menu:

Initially, the main page with have a menu with 3 options: create a new reservation, view future reservation, and cancel reservations. Initially I had each button in a circular shape to navigate to each function in my program. I presented this to my client, and he asked me to remove the circular shape, so for my next iteration I will add the rounded boxes.



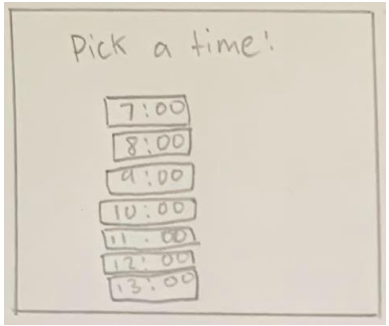
Booking a specific barber:

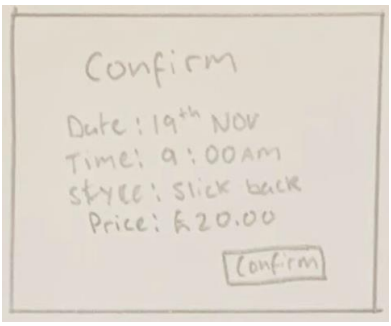
First page will be a page where the user picks a barber. This will be another mini menu where each barber will have their own button. The initial sketch was bare, with only 3 buttons. Feedback from my client said that he wanted to add his company logo, corners around the boxes, and change the text to select a barber for a more professional feel.



Reserving a day / hairstyle / time slot / confirmation :

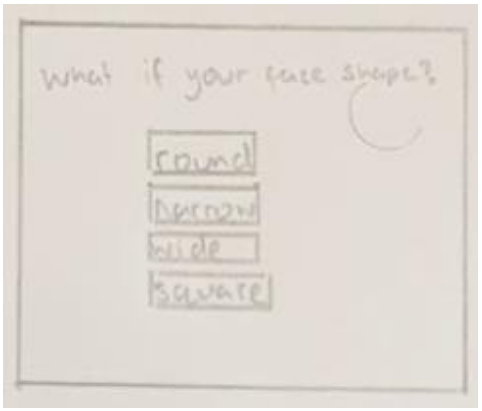
The user will now be presented with the next 1 week where they have the option to click any day of these days, choose a hairstyle, pick a time and confirm. These initial designs were bare as well with just boxes around the different boxes, indicating as a button. Feedback from my client, he requested again that the company's logo was in the corner of the pages along with the name. Also requested that instead of a small circular button on the side, that each button had rounded box that would act as the button instead.





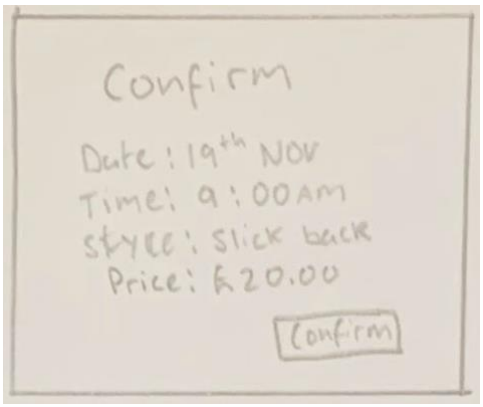
Recommendation:

If the user is unsure of a hairstyle, the button on the previous page will direct them to this page were. On this page the question is asked on the top where then the user has to pick an option provided by below. Each box itself will be a button. My client requested again that company logo/name was there with also rounded boxes. He requested that another button 'next/ was below as he didn't want people to continue the questionnaire with accidental answer, giving the user to confirm their option with next button.



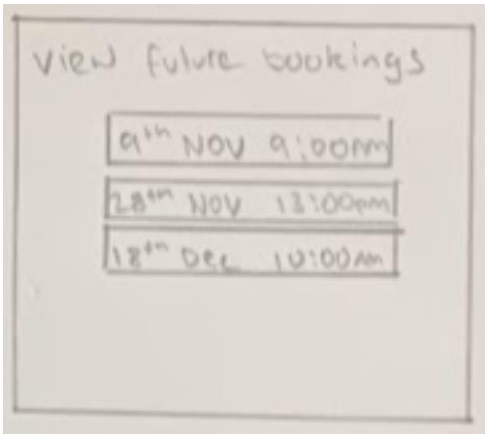
Confirmation:

This page will provide a short summary of what the customer has reserved with a button to confirm the reservation. My clients was happy with this simple page as it suffices it purpose. However, he did again request the company's logo was present in the corner with a rounded boxes, here he also remined me that a home button was required if a customer needed to make changes, so in the final design I will ensure this is present on all pages.



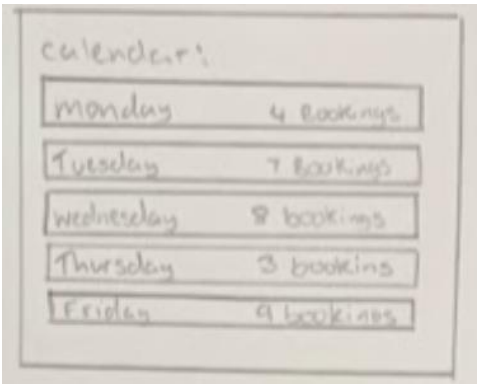
View / cancel reservations:

This page will allow the user to view their reservation with another identical page that will allow them to cancel their reservation. Again, my aim was to keep the page simple as possible, but due to my client's request, I will also implement his logo and rounded boxes to each box.



Calendar:

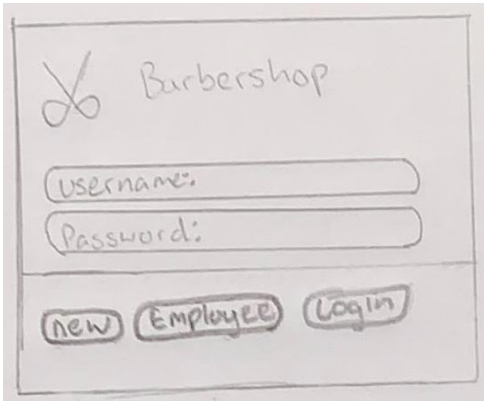
This page is the employees only, where it provides a weekly calendar where they can see how many bookings they got in that day. The original plan was when a user clicks on a day, it will direct them to another page that shows all the details within each reservation. But by my client's request, he asked me to create a drop-down box system where all the reservation information will be visible. He also requested that company's logo was present and that the boxes would be rounded.



2nd iteration

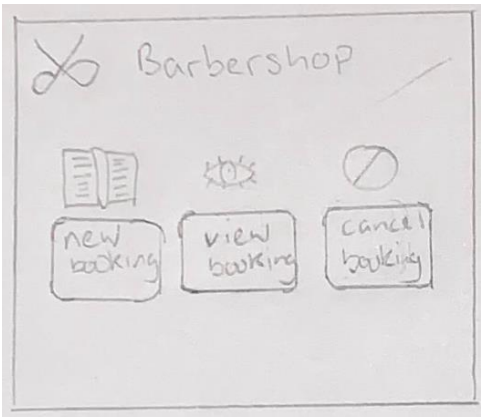
Login screen:

After considering my client feedback, I iterated another design implementing the barbershops logo and colour palette into my design further. I also decided to put rounded boxes around the area where the user needs to interact with, further making the design accessible to range of users. This also provided a 'friendlier' look to my page with less harsh corners. At the end of initial sketches, my client requested to implement this throughout my whole program.



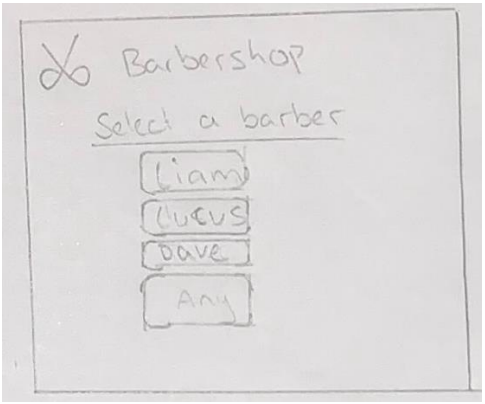
Main menu:

For my second iterative design, I decided to add icons to each button. This is to further make the design accessible as icons are universally recognisable compared to text, I however will also have text below the button to also provide a short description. This page also contains my client's logo only with 3 buttons, making the page simpler.



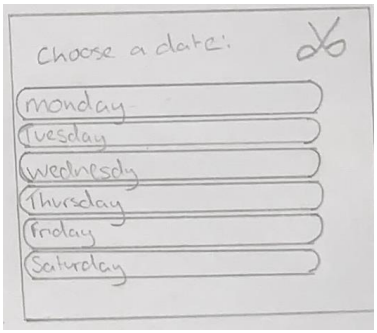
Selecting a barber:

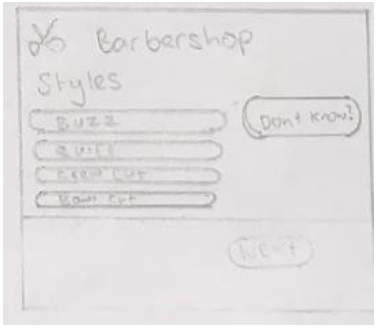
I changed the text on the next version while also adding the barber shop logo and rounded corners on each button. I also decided to add an any button where the user can pick a time slot between any barber.

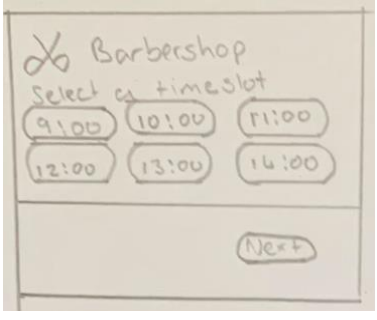


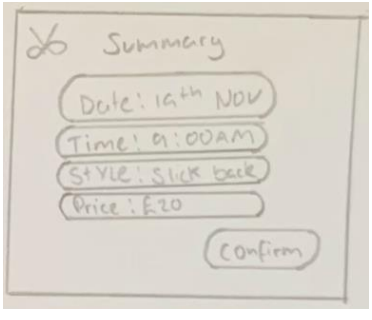
Reserving a day / hairstyle / time slot / confirmation :

I implemented my client's feedback into my 2nd iterative designs by adding the companies branding while rounding the boxes. I decided to move the button of centre to see if my client preferred this look, but he decided that he would want the text to remain in the middle. On the page to selecting a style, I added another button to the side where the user can be directed to the questionnaire for a recommended hairstyle. For choosing a time slot I decided to list the times horizontally so more information is on the screen and to reduce scrolling.



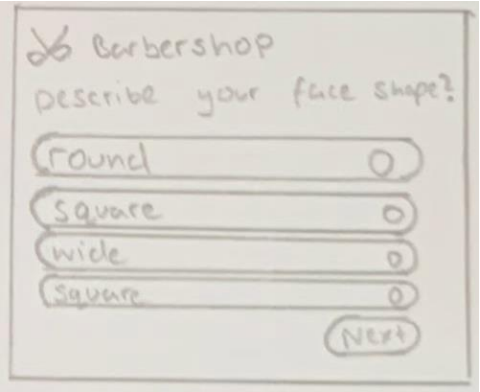






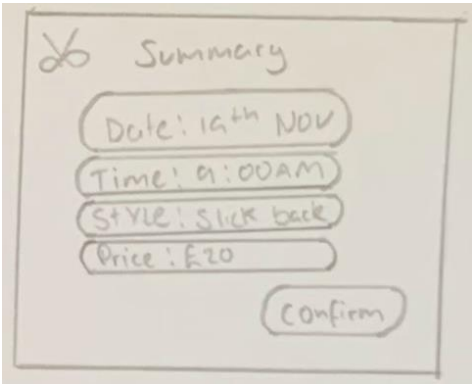
Recommendation:

Again, implemented my clients request of adding the company logo and rounded boxes. I also then added a next button and an option to select multiple options with a smaller side button. This next button allows the user a chance to confirm their answers preventing them from starting over if they made a mistake.



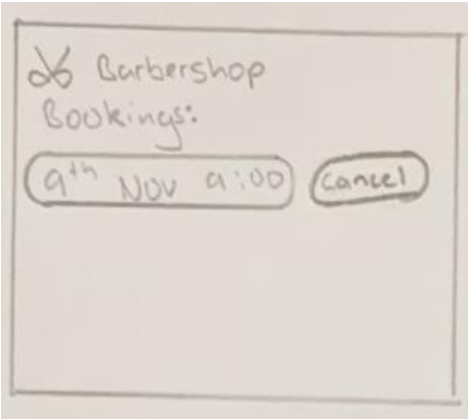
Confirmation:

This page was also kept simple and similar to its first design. Implemented the company's brand and added rounded boxes into the page. My client requested for the final design to instead have one larger rounded box instead of smaller ones as he believes it unnecessarily divides the page.



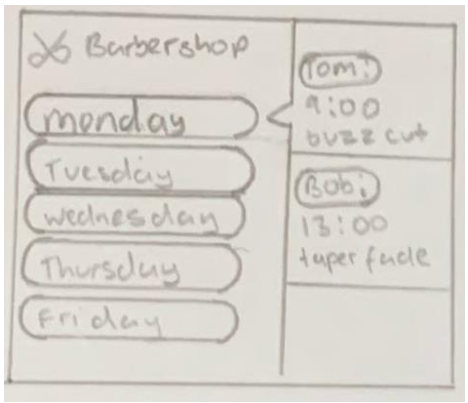
View / cancel reservation:

This page also had minor changes made, such as implementing the barbershop brand and rounded boxes. My client also suggested instead of 2 separate pages to cancel and view reservation, instead put it into a single page to reduce prevent from pages to pages and further streamlining the process.



Calendar:

Taking my client request into consideration, I condensed the information down into a single page. I implemented a drop-down box system where a barber can select and see all their reservation. In my sketch, I forgot to implement another button that allows them to see all the additional information for each reservation so I will include this into my final design. I also added the barbers request for his barbershop branding and rounded boxes.



Comments and conclusions on iterated designs:

Being that my program was simple in its core nature, me and my client could only make mostly smaller refinements to the iteration 1. While some larger changes were made that overall made the process more efficient for both my client's employees and customer, majority of these changes were smaller refinements that made the program more aesthetic while still manages to keep the program look more approachable and simpler.

Final design

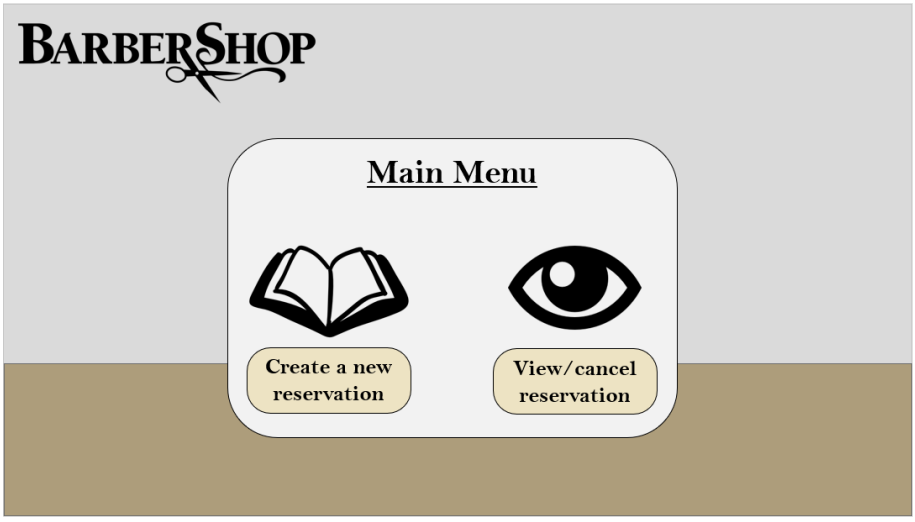
Login system:

My client has requested this colour palette as it is similar to interior design of his barbershop. I decided to colour the bottom section to create some contrast to make it more visually exciting. The accents match with the grey painted exposed brick wall while the light golden and brown colour match with the vintage chairs and moody lighting at the barbershop. In order to provide a uniform experience, I have implemented this style/colour palette throughout the whole system to ensure that the user associates the booking system with the barbershop itself and further make the program as attractive as possible.

I designed to keep the login in system as simple as possible, showing the logo and the different login options. This has been designed to be accessible from all age ranges where each type of login is clearly separated, this is perfect especially for the elderly who still enjoy getting a fresh haircut at the barbershop.

From my 2nd iterative design, I moved email address and password out of the text box as that may cause unnecessary confusion due to not knowing where to input their detail. This is the primary reason why I added a rounded box making it clear for users to input into that box. In doing this , making the system more user centred

Main menu:



For each button on this screen, I previously added icons to each action to further make the design accessible to a range of people no matter their background. This hopefully can assist people whose English isn't their first language. This was implemented as from conversation with my clients, I now know that many customers are not proficient at English, they may benefit from more icons that can universally explain the functions of the program. I placed the buttons in the centre of the screen as my client requested it, he said that it made the buttons stand out more, making it clearer. This with the larger font may help visually impaired in using the program easily. Reduces the button count to two as I decided to combine both viewing and cancelling booking to reducing jumping from page to page.

Selecting a barber and day:

These pages have very few differences from their previous counterparts. The new changes that I have made was a larger box around the area that the user needs to focus on. This creates separation between the background and the foreground further increasing usability as it reduces confusion to where they need to focus on. Another change I made was when selecting a day, the customer can now see the full date instead of the just the day, this avoids any potential confusions regarding what day the program is talking about.

Selecting a hairstyle:

On this page, some changes have been made. Firstly, my client remined me that prices must be listed along with the hairstyles provided. I implemented this into the program and came across a problem. All the services wouldn't fit onto one page. I could have presented the hairstyles horizontally but decided to use another approach as the page would become too overwhelming due to a lot of information. I displayed time slots horizontally as each box contains less information overall making the page less overwhelming while provide much more information. The other approach I done was to implement a scroller that lacked in my initial's designs.

Each box has a marker at the end that will change colour indicating to the customer that that has been picked. This allows the user to choose multiple services, for example the user can now have a buzz cut with a skin fade.

On each page corner, I added a home button as my client requested. This button directs the customer to their main menu allowing them to back out if they make a mistake when creating a reservation.

The recommendation button still remail but now on top of the continue button, this will help people from missing the button as if they want to continue, they will have to click 'continue' naturally guiding their eyes to that button. This wasn't added to the list of services as this feature would easily be lost in all the options, so it has been separated for everyone to have quick and easy access to it.

Selecting a date:

This page has little change from the initial design. The times slots are still displayed horizontally to fit as much information onto a screen without overwhelming the customer. The only changes are the scroller and home button that adds further functionality to this page.

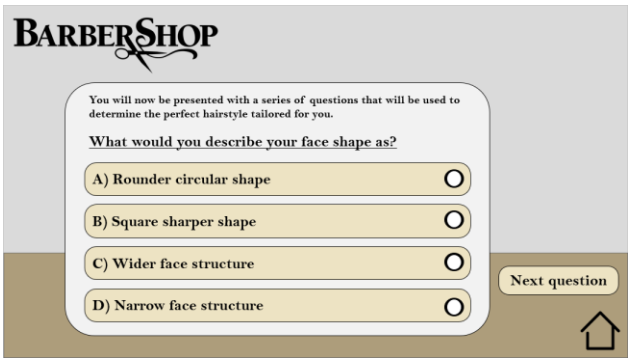
I changed the time format into the 12-hour clock instead of 24-hour time. This was requested by my clients as he believes that 12-hour time is easily read by more people, he said it will benefit the immigrants that regularly visit his barbershop as they wouldn't have to convert the time into 12 hour. To differentiate between past/after midnight I abbreviated it after displaying the time to avoid any confusion.

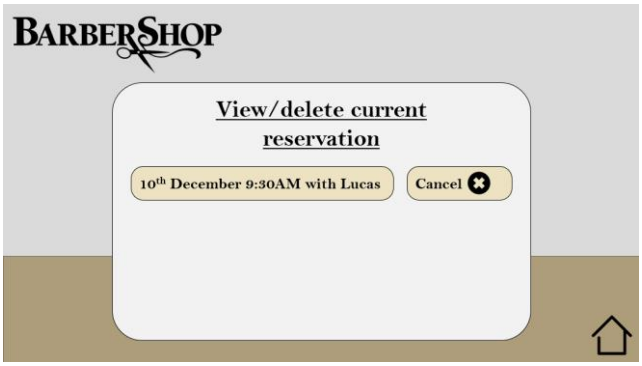
Confirmation screen:

On this page I decided to add the input message and upload an image onto this page. This is due to lack of space on other pages, so I decided to add these features onto the emptiest pages. In my initial designs I forgot to add these features in as there are the least important. Additional 2 extra boxes are optional for the customer allowing them to continue without having to input anything. These were implemented as my client said it forces the customer to think of what they want at home instead of in the chair, further saving more time.

In the summary section, I added the warning of the missing fee as my client requested it. I reminds the user that there a fee if they happen to miss an appointment without cancelling it on the program. I also added more information regarding the reservation giving the user time to check if there's any mistakes.

Recommended hairstyles:





These pages are essentially the same to my initial design with a few differences. On the top of first page, I added a short description of the process letting the customer know what there are getting into. I also have a marker at each question allowing the user to choose multiples answers as a person can have multiple attributes to them. On the second page I decided to keep it relatively the same to keep it simple as possible. The only change I made was I added more information when viewing a reservation. Apart from that all the design elements are present on this page further creating a uniform experience.

Employees calendar:

On the employees' calendar, further refinements were made to further streamline the experience. Compared to my first iteration, it contains a lot more information regarding the reservation. This caused a problem of overwhelming the user but to prevents this, it only displays the necessary information when each day is clicked on. To access even more information, such as the images uploaded, the user intuitively has to click on the view more button that will direct them to another page with the additional information. There is also scroller that also allows the barber to access more reservation in that day, preventing me from have to direct the barber to another page.

Each day has now a date attached to it, ensuring the barbers do not confuse different reservations with other weeks. This page continuous to follow the design language of the whole program ensuring the user gets some off the experience they will find at the barbershop.

Flowchart:

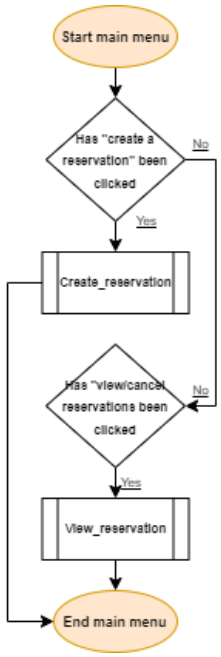
Down below are flow charts for each subroutine and function in my program. This will be used to aid development of any other program and any other developers to replicate my program.

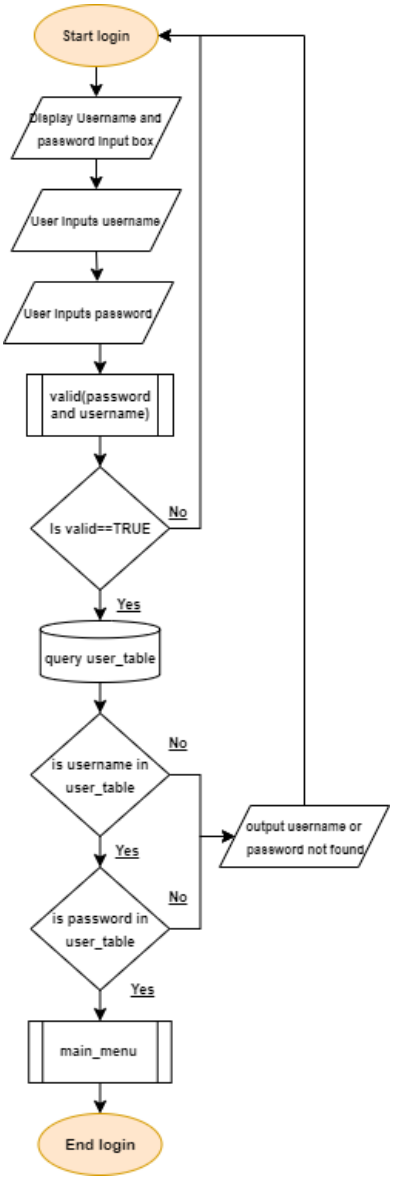
Main menu pseudocode:

```
Function main_menu ()
If create_reservation button clicked, then:
        Run create_reservation ()
elif view_reservation button clicked:
        Run view_reservation ()
else:
pass
```

Comments:

This is the flowchart for my main menu where the user will be able to pick what page they want to navigate to. The program will run a specific function depending on what button the user has clicked. As this is an event driven page, nothing will happen if no input/button has been clicked.





Login pseudocode:

```
Function login()
username=input(enter your username: )
password=input(enter your password: )
validate(username,password)
if valid==true then:
        load user_table
        for each record in user_table
                if username in user_table then:
                        if password in user_table then:
                                run main_menu()
                        else:
                                output (username or password not found)
                                run login()
                else:
                        output (username or password not found)
                        run login()
else:
run login()
end
```

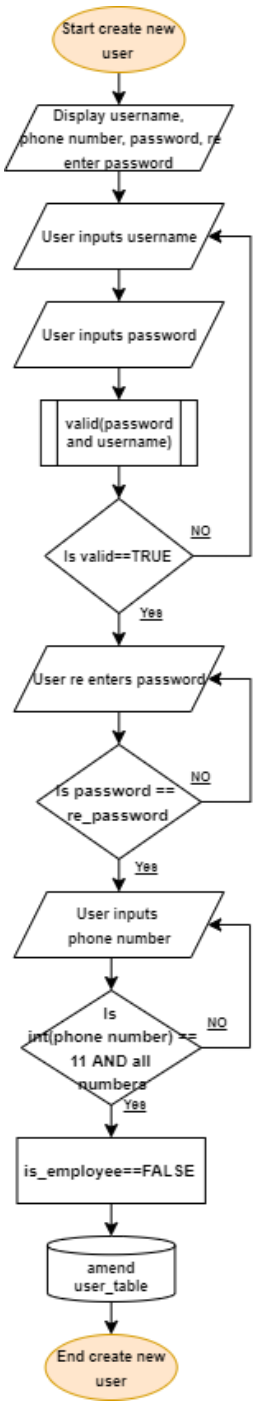
Comments:

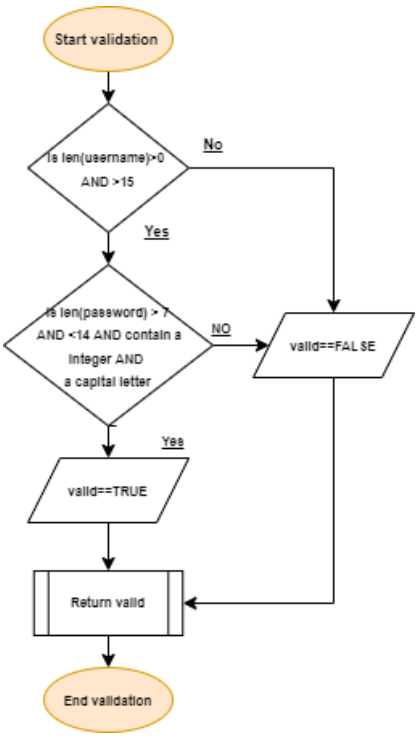
This function ensures only registered users can login and allows them to proceed to the main menu. Prompts users for their username and password, then my validate functions ensure the details fit the parameters required. The program will then load the user_table to compare the details provided to the details in the table, if no matched found or details are not valid, runs login again and vice versa.

Creating a new user pseudocode:

```
function create_user()
display username, phone number, password, re-enter password
username=input(enter your username: )
password=input(enter your password: )
valid(password, username)
If valid==true then:
        re-enter_password=input(re-enter your password: )
        while password=! re-enter_password then:
                re-enter_password=input(No match, re-enter your password: )
        endwhile
else:
        output (username must be 0-13 characters, password must be 7-13 and contain a number and capital)
        run create_user()
phone_number=input(enter your phone number: )
phone_number.replace(" ", "")
while len(phone_number)=! 11 AND phone_number.isdigit()==false :
        phone_number=input(Must be 11 digits, enter your phone number: )
endwhile
is employee==FALSE
update user_table with username, password, phone number, is employee
end
```

Users will be prompted to add their username, password and phone number. Once they input their details, my validating subroutine that I used for my login will also be reused to check if the user inputs fit the parameters for the database. After validating username password, I ensure the password is what the user intended, by making them re-enter it to match with the previous password and after check if the phone number has 11 numbers which is standard for all UK numbers and at the end, amend it into the user_table





Validate pseudocode:

```
Function validate()
valid==false
if len(username)>0 AND len(username)<14 then:
        if len(password)>7 AND <14 AND contain number AND contain capital letter then:
                valid==true
                return valid
        else:
                output(password is incorrect)
                return valid
else:
output(username is incorrect)
return valid
```

Comments:

This function validates username and password when login in and creating a new user. This ensures all username have 0 to 13 characters. Checks if the password has character lengths between 7 to 13, a capital letter and a number in it. Depending on the parameters inputted, it will either return a true valid or a error message and a false valid. It will also be more efficient as it will ensure only valid data is searched within the database.

Creating a reservation pseudocode:

function create_reservation()

load barber_table

display barber_names from barber_tables

barber_name button clicked

display dates for the week

date button clicked

load hairstyle_table

display hairstyles,prices from hairstyle_table

if don't know button clicked then:

    run recommend_hairstyle()

else:

    hairstyle button clicked

output reservation_time from reservation_table

where barber=!barber_name clicked

available time clicked

update reservation_table with date, reservation time, barber name, hairstyle

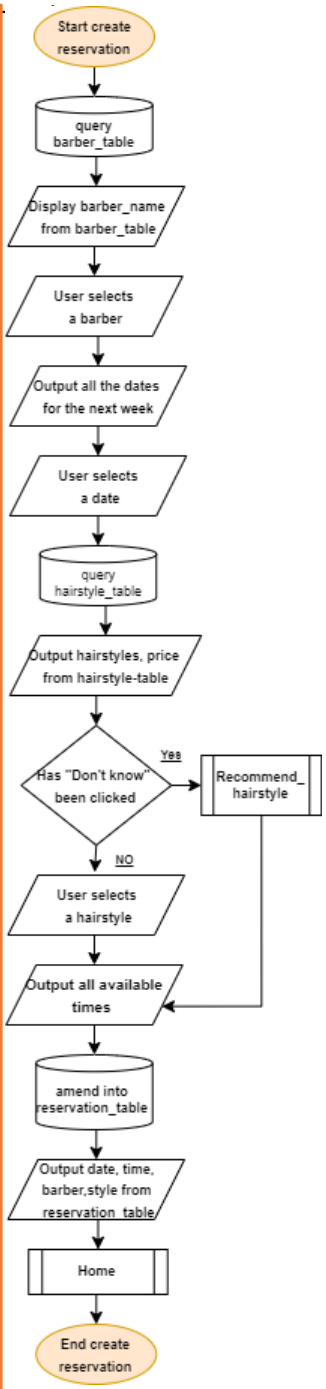
display date, time, barber, hairstyle from reservation table

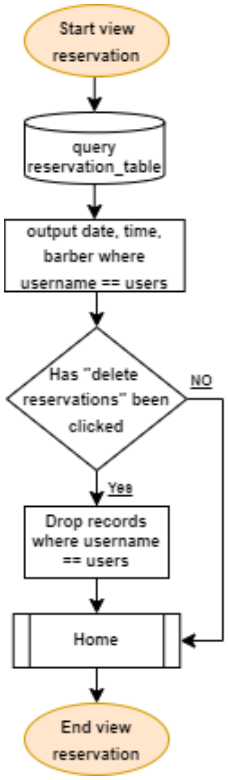
run home()

end

Comments:

This is one of the main functions of the program where the user will be able to create a reservation. As I have decided that the user mostly enter data via buttons, it prevents the need to validate every input, thus making the program much more simplistic and approachable for the user and allowing me to have more time to focus on core features of the program. All data must be inputted as at the end, all date will be added into the table and then will output the key details as a reminder for the user.





Viewing/cancelling reservation pseudocode:

function view_reservation()

load reservation_table

display date, time, barber from reservation_table

where username==username inputted

if delete reservations button been clicked then:

    delete records from reservation_table

    where username==username inputted and reservation_time==reservation_time clicked

else:

run home()

end

Comments:

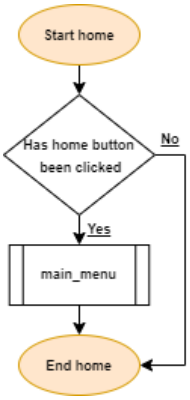
This will output all the reservations under that user's username, while also providing the option for them to delete specific reservations made. I also reuse the home function to give them the ability to jump to the homepage depending on if they gave clicked the button

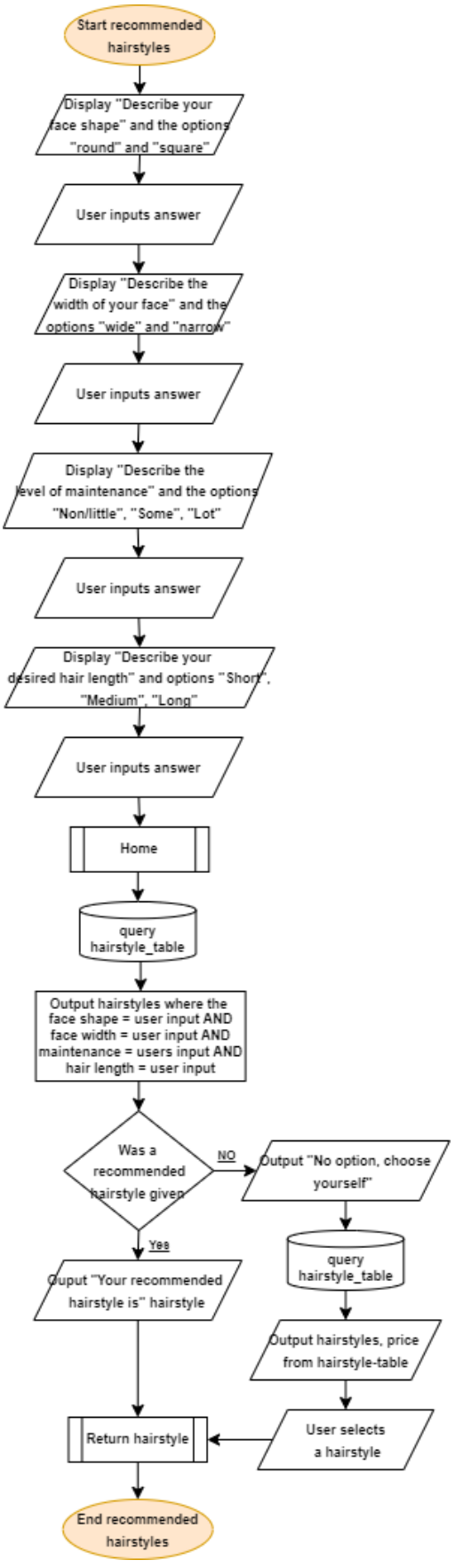
Home pseudocode:

```
Function home()
if home button clicked then:
        run main_menu()
else:
        end
end
```

Comments:

This is a simple procedure where if the home button has been clicked, it will take the user to the main menu, or nothing will happen is no event like pressing the button has taken place. This procedure will be reused multiple time throughout my whole program as I need a way for my user to go back to the menu





Recommend hairstyle pseudocode:

```
Function recommend_hairstyle()
display describe your face shape, round or sqaure
user_shape==button clicked
display describe the width of your face, wide and narrow
user_width==button clicked
display describe the level of maintenance, non/little or some or lot
user_maintainence==button clicked
display describe your desired hair length, short or medium or long
user_length==button clicked
run home()
load hairstyles_table
output hairstyled from hairstyles_table
where   face_shape==user_shape
        face_width==user_width
        maintenance==user_maintenance
        length==user_length
if hairstyle_table output null then:
        output (no results found, please choose yourself)
        output hairstyles, prices from hairstyle_table
        hairstyle button clicked
else:
        output(your recommended hairstyle, output picked styles form hairstyle_table)
        hairstyle button clicked
return hairstyle
end
```

Comments:

This function is optional for the user, where through the process of elimination, the program will decide a hairstyle based on the user inputs and then give the option for the user to choose what they like best. If no results are found, then the user will be presented with a list of buttons for every hairstyle in hairstyles table. They will then choose from their options and will return the hairstyle they have picked.

Employee calendar pseudocode:

Function employee_calendar()

display all the dates for the current week

user clicks a date button

load reservation_table

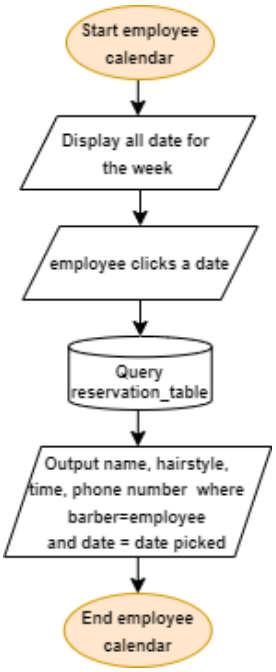
output name, hairstyle, phone number from reservation_table

where barber==employee name inputted AND date==clicked date button

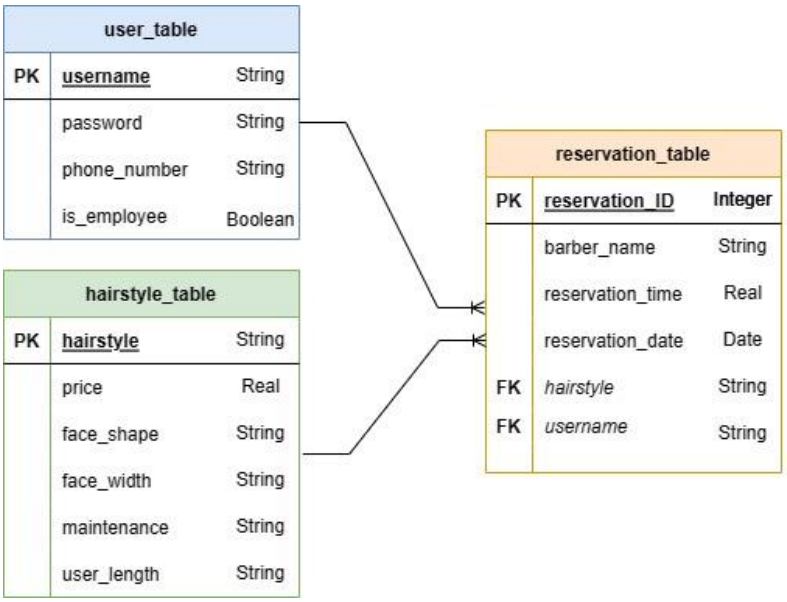
end

Comments:

This function is used after the employee logins into the program. They will be presented with all dates and have a choice in which day they are interested in and after that see all their reservation in a drop-down box. This will utilise the reservation table as all reservation are saved there. As there's little information, I decided to scrap the "view more" button as it is unnecessary with little information provided.



Entity relationship diagram:



This ER diagram is a visual representation of my database where it shows the relationship between the tables and the fields within each table.

To maintain the consistency of the data and to ensure that the database reflect reality that it represents, I will ensure ACID (atomicity, consistency, isolation, and durability)

My database will consist of a user table where it will store the username that doubles a primary key, a password to ensure only authorised access to users account along with their phone number to allow barbers to contact the customer if needed. Another field I will include is is_employee, this will be used for employee login where it will query records where is_employee is true, this prevents unnecessary table with just employees' details as they can be easily merged into user table. This is a one-to-many relation to the reservation table as users can have multiple reservation, but a reservation can be booked by one user only. The primary key, username, will also be a foreign key within the reservation table to join both user_table and reservation_table.

Hairstyle table consists of the primary key, hairstyles, as each hairstyle within the table is unique. It stores the information regarding the hairstyle such as the price of the haircut and along with its attributes that will be used in the recommend hairstyle function. Hairstyles has many to one relationship as a hairstyle can be in many reservations, but a reservation has only one hairstyle. The primary key, hairstyles, will also be a foreign key withing the reservation table to join both hairstyle_table and reservation_table.

Reservation_table will be most important table within my database as this is essential for the core function of the program. The primary key, reservation_ID, will be generated so that no two reservations have the same ID. It will contain all the information regarding a reservation as the barber_name. This will also be used to in the employee calendar as it will output reservation with barbers with the name they input into during login. It will contain the time and date whilst also having the hairstyle the customer wants, allowing the barber to prepare in advance and their username so they will be able to recognise regular customers and further prepare for needs prior to their reservation system. My program will also need to ensure that one barber isn't double booked at same, by removing the time slots that are already blocked.

Variable table:

Variable name	Data type	Scope	Purpose	Is it validated	Screens/where used
User_username	String	Local	Uniquely identify customers to allow them to access their account	Validated within validate subroutine to ensure its between 0-13 characters	-Login -Reserving -Employee calendar
User_password	String	Local	Allow authorised users to login to system	Validated within validate subroutine	-Login
Reservation_ID	Integer	Local	Primary key for reservation table	No	-Reserving
Re_enter password	String	Local	Ensure that correct password is inputted the first time	Checked against the original password	-Validate login
Phone_number	String	Local	Customers phone number for barber to contact	Validated to ensure it has 11 digits	-Employee calendar
Valid	Boolean	Global	Checks if username and password is valid	No	-Validate login
Final_hairstyle	String	Local	Haircut the customer wants	No	-Create reservation -Recommend hairstyle -Employee calendar
Final_barber	String	Local	Customer preferred barber	No	-Creating reservation
Final_price	real	Local	Price of haircut	No	-Confirmation -Create reservation -Recommend hairstyle
Final_date	Integer	Local	Customer preferred date	No	-Create reservation -Confirmation -Employee calendar
Final_Time	Integer	Local	Customers reservation time	Checked if timeslot is taken	-Create reservation -Confirmation -Employee calendar
Hair length	String	Local	Customers desired hair length	No	-Recommend hairstyle
Face shape	String	Local	Customers face shape for best hairstyle	No	-Recommend hairstyle
Face width	String	Local	Customer face width for best hairstyle	No	-Recommend hairstyle
Maintenance	String	Local	Customers desired level of maintenance for best hairstyle	No	-Recommend hairstyle
New_Uname	String	Local	New account username	Checked if between 0 to 11 characters	-Create account
New_Pword	String	Local	New account password	Checks requirements	-Create account
New_Pnumber	Integers	Local	New account phone number	Check if 11 digits	-Create account
Tbl_results	Strings	Local	Stores results searched from table	No	-Validate login -Filtering timeslots -Confirmation
Employee	Boolean	Local	Set employee as false	No	-Login

Software development methodologies:

As mentioned prior, I have decided to take a rapid application development. I've decided to take this methodology on as I needed to gain lot of feedback that will help me make the program a success. Through getting regular feedback about my designs and features, this had allowed me to design a system that meet the requirements my clients had set for whilst making my program as usable as possible. I will be creating successive prototypes of the software and further present them to client to help identify early issues giving me enough time to tackle them on over the duration of the development. This methodology has also allowed time for my client to thoroughly of the requirements instead of just rushing through them early on. I've also taken aspects of the waterfall development by also heavily documenting every part of development that will then in the future, give future developers the information they need to modify my program by adding new features and to further refine.

Plan for testing my program:

I will incorporate a variety of testing from alpha, beta, white box and black box. Alpha test will help me reveal errors within my program while also helping to ensure all requirements are met, I will be using alpha testing as I produce successive prototype to ensure no mistakes are carried over to the next iteration, thus helping me produce the program to meet all stakeholder requirements. Near the end of the development, I will distribute my program to clients and peers to help beta test my program. This will benefit the program as I know some users may try certain actions that I haven't anticipated, causing the program to crash or fail. This will again help me identify any errors within my code, helping me to solve them quicker as I have more information from a variety of users regarding the issue, further increasing chances of success for launch. To implement both white and black box testing, during development I will attempt to test every path of my code to prevent even more errors. As this is a smaller project, it will be feasible for me test majority of the possible paths. I will test each module independently to help me isolate the problem faster and test the code logically. Once I've developed the program, I will compare each part of the system against the original requirements in analysis, and present to my clients' which requirements where met and which were not.

Test	What is the test	Expected outcome	Why am I testing it?
Fit design language	Test against documents and interior design of the barbershop to see if there is a coherent design language. Compare the colour shapes and feel	I expect the design language to be similar. My aim is for users to be able to glance at my program and unmistakably identify it as The Barbershop	This was a requirement made by my client as he wanted his booking system unique from the competition
Run on variety. of hardware	I will run my program first on my clients' laptops within the barbershop to see if the employees can access their calendar. To test this further I will run it on older, less computational powerful hardware to test the lower end to ensure its compatible with a variety of devices	As this is a simple program that requires little memory/power. I expect it to run on most devices	This further helps the goal to make the program as accessible as possible no matter the device available to customers. Helping reach as many users as possible

Have high quality images for icons	As I use the program, I will inspect every image/icon to ensure they are at a high quality	If I find high quality images, I expect the images should be clear enough	To further be accessible, to prevent blurry images that can cause human error whilst giving the program a “clean” and professional look
Maintain data integrity	Check the data within hairstyle table match with barbershops service With correct pricing	I expect there to be no mistakes but there is a potential for one, however this will be easy to spot and solve so the issue isn’t large	To prevent confusion to customers for when the prices do not look right or have services that are not listed
Enforce user access levels	Test the login system by trying to login into both employee/customers logins with invalid details like “j@(“(ss”	I expect my program to have no issue preventing false details	To ensure there’s no unauthorised access the system
Usernames validate	Test if new account created with usernames not within 0 to 15 characters. Test also with no input.	I expect the system to prevent usernames with the incorrect format	To ensure username fits the parameters and allow the database/rest of the program to function properly
Passwords validate	Test if the program allows access with passwords without capital letters and number like “jobiar”. Test invalid passwords with both creating a new user and login in. Test also with no input	I expect the system to reject invalid passwords that do not fit the criteria and only accept boundary and invalid inputs	Help the user by providing them a message if their input is wrong. This will reduce confusion in customers where they are not sure why the system isn’t letting them login in
Re-enter password match	Test if a new account created if I enter a password that doesn’t match with the password made before. Such as “JOBiar.123” and “jobIAR:123”	I expect the system to notify the user if the re-enter password doesn’t match the one, they input prior	Help reduce human error where user accidently inputs the wrong password to the one, they expect
Test the login/create a new user button functions properly	I will test this by clicking the button and seeing if it directs the user to the correct page or not	I expect this to function properly	Prevent the user from being directed to wrong parts of program
Create a new user successfully	I will create a variety of users to see whether if the program creates an account with the correct information. Will also try to create identical accounts with the same details to see if the program accepts it	I expect this to create new unique users that also stores all their correct information that they have inputted	This is essential as all users using the program will have had to first create an account. If it doesn’t store the correct information, it can cause issue down the line for the barbershop
Menu page that allows user access different parts	I will test if the buttons on the menu page led to the correct part of the function.	I expect this to allowing user access to different parts of the program.	Important to test as this be essential page that allows users to effortlessly navigate the program
Reservation successfully save on the database	I will create a multiple reservation whilst also trying different options to ensure everything functions correctly	This part of the program is the main function, as this is vital, I would have spent the most time on it and expect it to create a reservation correct	The program will have to save in order for the barbers to prepare for reservations prior and prevent confusion

Displays all relevant information in employees' calendar	I will create multiple test reservation and through employee login, check if it correctly displays reservations for that employee only	I expect this to only show reservation only on days the barber has selected and reservation only relevant to that barber	This is needed for the employee to see which reservations they have coming up to allow them to work with a schedule
Does the program prevent double booking	Test if I can create a reservation on the same day with the same barber at the same time. Such as creating 2 reservations with Liam on the 14th of June at 7:00	This program should prevent double bookings by removing unavailable time slots from view when creating a new reservation	This is essential to prevent frustration in both customers and barbers as only one person can be with one barber at a time, therefore test to prevent loss in business
View reservation	I will create reservations with the same account and then see if the view reservation page shows reservations that have been created	I expect for all reservations to be displays; this includes past reservation as my client stated that it isn't required for only future reservations to be displayed	Needed as some customers often forget the time and date of their reservation and this reduces missed reservation further reducing loss in business
Test if reservation can be cancelled	I will create a variety of reservations from the same account and delete some reservation, check the table itself and see if the picked reservations have been deleted only	I expect reservation picked by the user to be deleted only and leave all the other reservation intact	Sometimes customers are unable to come for their reservations and will need to cancel, also important that it doesn't affect any other reservations within the table
Program displays correct dates of the current week during creating reservations	See if the program displays all the days that should be available	I expect the next 7 days to be displayed for the user and not any dates in the past	Creating a reservation in the past is impossible, I will need to ensure correct dates are displays so they have enough time for them to fill up with reservations
Recommend hairstyle products logically correct results	I will run the function multiple times and check if the responses make sense and are logically correct	I expect the function to produce sensible responses to the answers	It will be pointless if it produces responses that do not make sense. For example, I pick "short hair", and it producing "man bun". It will be impossible to have that hairstyle with short hair
Home button leads to menu page	Click the button on all pages to see if it directs me to the home page	I expect this to direct me to the main menu	This doubles as the back button as well as the home button. It will be important for it to function as users need a way to go back if a mistake is made.
Confirmation screen displays all the correct inputs from the user	I will create reservations and see if the confirmation displays the relevant information regarding the reservation	I expect the price, hairstyle, barber, date and time to be displayed for the user after inputting the details for a reservation	This must display the correct information to give the users a chance gloss over the facts and make sure they are correct, if it were incorrect, this will create confusion in for user
Update/ reschedule reservations	Will create multiple reservations on the same account and edit some reservations with different dates, then check the database to	I expect the reservation to be modifiable and prevent double books, it should only affect the	Some users must reschedule in order to attend, therefore important this feature functions correctly to allow customers to

	see if correct reservation were affected only	reservation being modified and no others	make a reservation that works for their schedule
Allow employees into employee calendar whilst denying users in	Will attempt to login to employee login with user's login details to if they work, test if invalid account "jobiar" can login and password "Jobiar123"	I expect only employees to be able to login into employees' calendar and only users to login to user's accounts	Must prevent users from being able to gain access to employee's calendars as this can cause complication for the business/barbers
Method to scroll through lists	Will attempt to scroll/flip through a list so everything can be seen on a single page	I expect the scroll feature to allow users to scroll through the lists of times and hairstyles	Users will need to be able to access parts of the list that would otherwise be too large to it onto one page
Updating hairstyle table	Attempt to update a price within the hairstyle table. I will change the buzzcut price from £10 to £12, and insert new hairstyle with its own attributes	I expect employees to be able to update the table only on employee window and successfully change required fields	Barbershop will need to change the services to keep up to latest trends and changes in prices to remain competitive

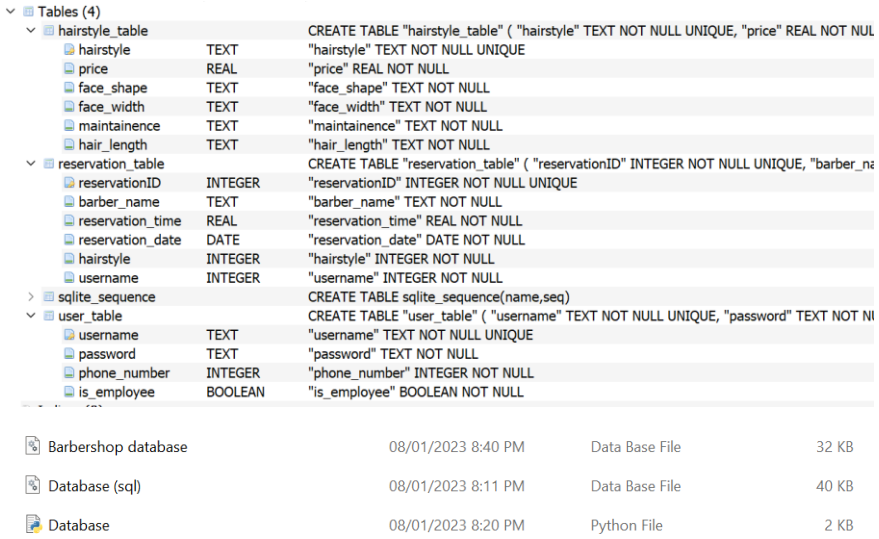
Development:

I will now be being to produce the code for the program. Now that I have all my requirements, designs, and template. I will begin by building my database using my entity relationship diagram. I will begin with the database first as this is the backbone for the rest of the program, as the rest of the program relies on this. I am separating development into 3 prototypes. In prototype 1, I plan on creating the all the core functionalities of the program, this includes creating the database, interface, validating logins, create account, and the creating reservation. In prototype 2, creating all the additional features that will improve quality of life. Lastly in prototype 3, I will solely focus on the aesthetics and final touches.

Prototype 1: Creating the core functionalities.

Creating an empty database:

```
def create_database():
    cur.execute("""CREATE TABLE "user_table" (
            "username"    TEXT NOT NULL UNIQUE,
            "password"  TEXT NOT NULL,
            "phone_number"      TEXT NOT NULL,
            "is_employee"       BOOLEAN NOT NULL,
            PRIMARY KEY("username")
    )""")
    #creates the user_table with all its fields

    cur.execute("""CREATE TABLE "hairstyle_table" (
            "hair_ID"    INTEGER NOT NULL UNIQUE,
            "hairstyle" TEXT NOT NULL,
            "price"     REAL NOT NULL,
            "face_shape"        TEXT NOT NULL,
            "face_width"        TEXT NOT NULL,
            "maintainence"      TEXT NOT NULL,
            "hair_length"       TEXT NOT NULL,
            PRIMARY KEY("hair_ID" AUTOINCREMENT)
    )""")
    #creates the hairstyle_table with all its field

    cur.execute("""CREATE TABLE "reservation_table" (
            "reservationID"     INTEGER NOT NULL UNIQUE,
            "barber_name"       TEXT NOT NULL,
            "reservation_time"  REAL NOT NULL,
            "reservation_date"  DATE NOT NULL,
            "hairstyle" TEXT NOT NULL,
            "username"  TEXT NOT NULL,
            PRIMARY KEY("reservationID" AUTOINCREMENT)
            FOREIGN KEY ("username") REFERENCES user_table,
            FOREIGN KEY ("hairstyle") REFERENCES hairstyle_table,

    )""")
    #creates the reservation_table with all its field
```



I created a new function, create_database, which will create a new file, barbershop database.db, that contains the actual database. As I am just creating the tables, they are currently empty. I plan on next populating the hairstyle table next with all the information my client has provided me. This includes the hairstyles and the prices. Within the brackets, I programmed the tables using SQL, importing the SQL library so that python can recognise and run the SQL code. As seen above, this function works as it created the barbershop database and allows me to see all the tables and fields within SQLite.

Populating the database:

```
file=open("Hairstyles spreadsheet.csv","r") #opens clients files with all the prices / services
for line in file:  #ensures every record is inserted
    line=line.strip() #removes any whitespace
    hairstyle,price,face_shape,face_width,maintainence,hair_length=line.split(",")
    cur.execute("INSERT INTO hairstyle_table VALUES (?,?,?,?,?,?)",[hairstyle,price,face_shape,f
    #puts the data from the spreadsheet into the database
con.commit()

file=open("Employee detail.csv","r") #opens clients files with their account details
for line in file:  #ensures every record is inserted
    line=line.strip() #removes any whitespace
    username,password,phone_number,is_employee=line.split(",")
    cur.execute("INSERT INTO user_table VALUES (?,?,?,?)",[username,password,phone_number,is_emp
    #puts the data from the spreadsheet into the database
con.commit()
print("Everything is fine")
#gives me feedback telling me its fine
```

username	password	phone_number	is_employee
Filter	Filter	Filter	Filter
lucas grant	eMphv6Ezb	07700900264	TRUE
timothy joshua	GunV4pNHX	01154960918	TRUE
liam smith	4rGCaiRUv	01314960927	TRUE
dave jackson	L3qsQaEIV	02079460136	TRUE

hairstyle	price	face_shape	face_width	maintainence	hair_length
Filter	Filter	Filter	Filter	Filter	Filter
Buzz cut	15.0	sqaure	narrow	non/little	short
French crop	20.0	sqaure	wide	non/little	medium
Curtains	30.0	round	wide	lot	long
Quiff	20.0	round	narrow	lot	medium
Slickback	25.0	round	wide	lot	medium
Crew cut	15.0	sqaure	narrow	medium	short
Man bun	30.0	round	narrow	medium	long
Bald	10.0	sqaure	wide	non/little	long
Shag	25.0	round	narrow	medium	long
Edgar	25.0	sqaure	narrow	lot	medium
Spiky	20.0	round	wide	lot	medium
Crop top	20.0	sqaure	narrow	non/little	short
Comb over	15.0	round	wide	medium	short
Ivy league	25.0	sqaure	narrow	lot	medium
Mohawk	30.0	round	wide	non/little	medium
Bowl cut	15.0	sqaure	wide	non/little	medium
Caesar	15.0	sqaure	wide	non/little	short

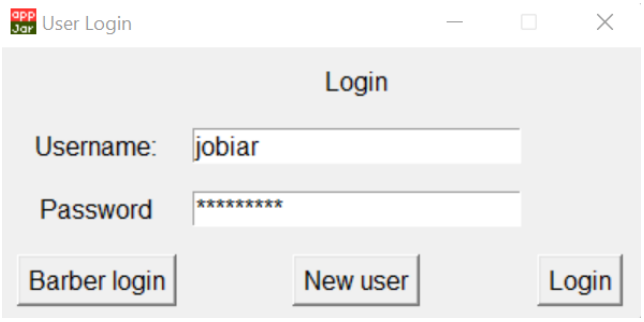
Outside the create database function, I created the code that populates the database with all the relevant information the client provided me. My client sent me an excel spreadsheet that had all the information regarding the hairstyles. In order to insert it, I had to convert it into comma separated values file as an excel sheet isn't supported. When I proposed the idea for the recommended hairstyles, I asked him to send me the characteristics for each hairstyle and using that data, inserted it into my hairstyle table. I also added another field into the user table is_employee and set the employee records only as TRUE. This will help me when creating an employee login where I can set the is_employee as TRUE in order to let them in.

```
Traceback (most recent call last):
  File "C:\Users\216104\OneDrive - Kingsmead\Barbershop project\Code\Database.py
", line 47, in <module>
    cur.execute("INSERT INTO hairstyle_table VALUES (?,?,?,?,?,?)",[hairstyle,pr
ice,face_shape,face_width,maintainence,hair_length])
sqlite3.OperationalError: no such table: hairstyle_table
```

Prior to populating the database, I received an error where the database file would not be created as the hairstyle wasn't created. To fix this error, I moved the inserting data code outside the create database function and this solved the problem as the tables where first created and then the program can insert data into the empty tables.

Login window interface:

```
##### LOGIN PAGE #####
app.setSize("400x150")#Sets size of window
app.setResizable(False) #Prevents window size change
app.addLabel("Login title","Login",0,1)

app.addLabel("user", "Username:",1,0)
app.addEntry("inputU",1,1) #Input box for username
app.addLabel("password","Password",2,0)
app.addSecretEntry("inputP",2,1) #Input box for password

app.addButton("Login",valid_login,3,3) #Login button
app.addButton("Barber login",barber_login,3,0) #Barber login
app.addButton("New user",new_user,3,1) #Create user button
```

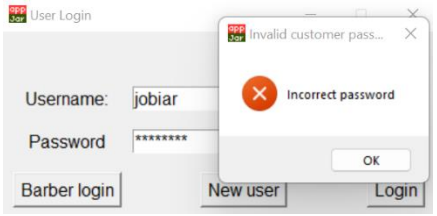


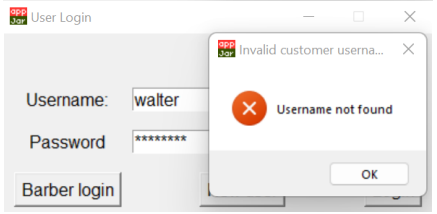
This piece of code is within my create_function() and it begins by displaying the main window of my interface, allowing the customer to login. From this window, I will be able to build upon this with multiple sub windows for the rest of the interface. This window consists of 2 input boxes that are labelled with what the customer needs to input. To provide extra privacy for the customer, I utilised password masking by hiding their input with *. This window includes all the features my client required, such as the barber login and new user options. I also utilised coordinates to place the buttons, labels and input boxes in spaces that feel natural to customers as users will be familiar with the layout as many other sites use this similar layout.

As this is prototype 1, my client decided to strictly focus on core functionality in prototype 1, leaving the aesthetics to later prototypes as my client said that a functional program was more useful to him than an attractive one. This also benefits me as it gives me enough time to focus on what's important, instead of fixating on the looks.

Creating the validate login function:

```
def valid_login ():
    user_username=app.getEntry("inputU") #stores username input as a variable
    user_password=app.getEntry("inputP") #stores password input as a variable

    cur.execute("SELECT password FROM user_table WHERE username=?", [user_username])
    #Searches if users input is within database
    tbl_password=cur.fetchone() #Stores one result as password

    cur.execute("SELECT username FROM user_table WHERE username=?", [user_username])
    tbl_username=cur.fetchone() #Stores one result as username

    if tbl_username==None: #Checks if username exists
        app.infoBox("Invalid customer username","Username not found")
    else:
        if tbl_password[0]!=user_password: #Compares if password in table match input
            app.infoBox("Invalid customer password","Incorrect password")
        else:
            app.showSubWindow("Menu") #Success, moves onto next page
```



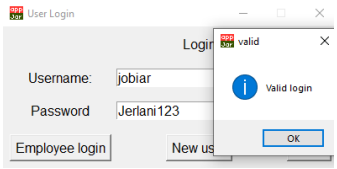


This function then checks if the customer inputs are correct by checking against the barbershop database. It starts by storing the customer inputs as variables that are then used next to query the database. When searching through the database, it first searches for any passwords that have the username given by the customer and stores any results. Next line of SQL searches for any usernames with the username given by the customer and stores it as a variable. With a series of selection statements, it checks if the table results are None and accordingly gives an appropriate response. I've added information boxes that alerts the customer if any problem arises with their details but if all details are correct, it will show the sub window, Menu.

```
T.py", line 81, in valid_login
    if user_username==tbl_username[1]:
IndexError: tuple index out of range
```

Originally, I compared if the all the results from the database search matches with user inputs. As this produced me with errors, I decided to check what the table results variables are storing by printing them out. In doing this I found out that if results are produced, the details would be correct already as the SQL code already filters out incorrect information. With this information, I then checked if the table results came out as None instead. This change successfully allowed access, but with a new problem. It allowed access with correct details, but with details from different accounts.

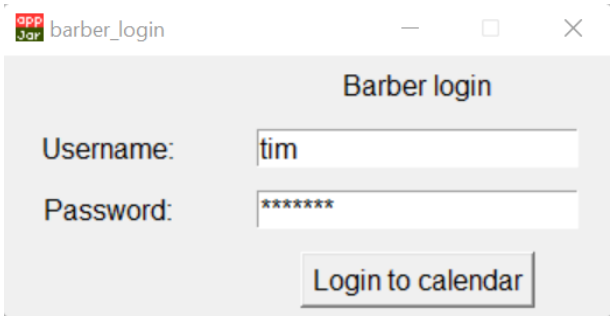
```
cur.execute("SELECT username FROM user_table WHERE username=?",[user_username])
#Searches if users input is within database
tbl_username=cur.fetchone()#Fetches only one
cur.execute("SELECT password FROM user_table WHERE password=?",[user_password])
```



As I tried to find my mistake, I realised that even if the details where incorrect for one account. It wouldn't produce a none result as it still existed somewhere else in the user table. To fix this issue, I adjusted the SQL code to find passwords in the table with the customers username input and find usernames in the table with customers username. Later during selection, I checked to see if the customer password input matches with table results. This ensures that only details for one account were searched for. This works, now only ensuring that all details that are compared to the customers input

Barber login window:

```
##### BARBER LOGIN #####
app.startSubWindow("barber_login") #Creates a new subwindow
app.setSize("350x150") #Sets the size of the window
app.setResizable(False) #Prevents change to size window
app.addLabel("employee title","Barber login",0,1)

app.addLabel("B_username", "Username: ",1,0)
app.addEntry("barberU",1,1) #Input box for username
app.addLabel("B_password","Password: ",2,0)
app.addSecretEntry("barberP",2,1) #Input box for password

app.addButton("Login to calendar",barber_validate,3,1)
#Button leads validate function
```

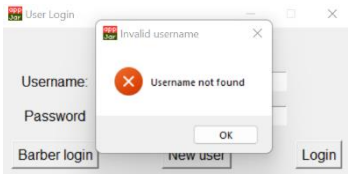


This section of code is within create_interface() and this window is accessed through the login page and only allows barbers to login. Again, as this is prototype 1, I've decided not to focus on any looks and strived for pure functionality. This page shares many similarities to my login page where they both contain username input boxes and masked password input boxes. Masked the password for extra layer of privacy for the barber and a button that leads to the barbershop calendar. I've also made use of coordinates to match the layout of my customer system to provide a more consistent design across the program. I also fixed the sizes of each window in my program to prevent distortion and further provide a uniform experience whilst also making the windows have a similar size. Even though I can utilise AppJar commands to prevent distortion when the windows are stretched, I limited the sizes as its unnecessary in most scenarios. When the barber clicks on the login button, it will lead to the validate function that checks their details against the database.

Creating the function that validates barber login:

```
def barber_validate():
    barber_username=app.getEntry("barberU") #stores username input as a variable
    barber_password=app.getEntry("barberP") #stores password input as a variable
    employee=1 #Variable used to check if details are for a barber account

    cur.execute("SELECT password FROM user_table WHERE username=? AND is_employee=?",[barber_username,employee])
    #Searches for passwords that only are from barbers accounts
    tbl_password=cur.fetchone() #Stores a single password value

    cur.execute("SELECT username FROM user_table WHERE username=? AND is_employee=?",[barber_username,employee])
    #Sarches for usernames that only from barbers accounts
    tbl_username=cur.fetchone() #Stores a single username value

    if tbl_username==None: #Checks if any usernames are found
        app.errorBox("Invalid username","Username not found")
    else:
        if tbl_password[0]!=barber_password: #Checks if tables passwords matches barbers input
            app.errorBox("Invalid password","Incorrect password")
        else:
            app.infoBox("Its good","works") #Success, onto next window
```

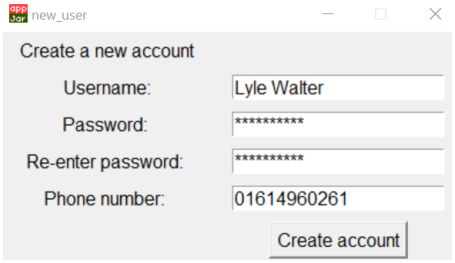
	username	password	phone_number	is_employee
	Filter	Filter	Filter	Filter
1	lucas grant	eMphv6Ezb	07700900264	1
2	tim	Tim123	01154960918	1
3	liam smith	4rGCaiRUv	01314960927	1
4	dave jackson	L3qsQaEIV	02079460136	1
5	jobiar	Jobiar123	07958642818	0
6	jerlani	Jerlani123	07700900101	0



This new function checks if the barber username and password input are in the user table. This function is identical to my customer, but it also checks if the record being searched has is_employee as 1. Previously I planned on using true and false for this field but as Boolean isn't supported by SQLite, I used the integers 1 and 0 as they can also represent as true and false. I used 1 to represent true and 0 as false. In order for this to work I had to create a local variable, employee, and set it to 1. This is so I can use it in the SQL code to search for records with is_employee as 1. This then prevents customer accounts from logging into the barber login and only allow barber accounts to login. As I haven't created my calendar window yet, I had to display an info box that lets me know it successfully let me gain access.

New user window:

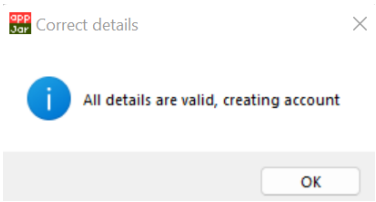
```
##### CREATE NEW USER #####
app.startSubWindow("new_user") #Creating new subwindow
app.setSize("400x200") #Sets the size of the window
app.setResizable(False) #Fixes the size of window

app.addLabel("new user title","Create a new account",1,0) #Title of window
app.addLabel("username1", "Username:",2,0)
app.addEntry("new_username",2,1) #Input box for new username
app.addLabel("password2","Password: ",3,0)
app.addSecretEntry("new_password",3,1) #Input box for new password
app.addLabel("repassword2","Re-enter password: ",4,0)
app.addSecretEntry("new_password2",4,1) #Input box to check if password matches
app.addLabel("phone","Phone number: ",5,0)
app.addEntry("new_phone",5,1) #Input box for phone number
app.addButton("Create account",create_account,6,1) #Button leads to validating / inserting
app.stopSubWindow() #Stops the window
```



This block of code produces the user interface on the right. This can be accessed via the login page that will then allow customers to create a new account. It consists of 4 input boxes: username, password, re-enter password and phone number. Most of these details will be inserted into the user table as a new record. The re-enter password is a feature that I've added, this is to ensure that the customers password input is what they intended it to be. This reduces chance of customers being locked out of their account due to remembering/inputting their password wrong. As I may not be offering a feature to change account details, my client suggested this idea as an alternative. The re-enter password variable will not be saved into the database as its unnecessary information and will be only used for validation. Again, focused on functionality than aesthetics as its prototype 1.

Creating function that create accounts:

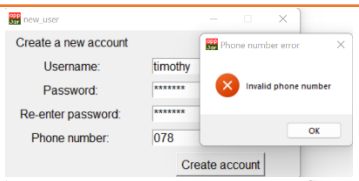
```
def create_account():
    new_Uname=app.getEntry("new_username").strip() #Stores username input as a variable
    new_Pword=app.getEntry("new_password") #Stores password input as a variable
    re_enter_pword=app.getEntry("new_password2") #Stores re enter password as a variable
    new_Pnumber=app.getEntry("new_phone").replace(" ", "").strip() #Stores phone number as a variable

    employee=0 #Creates a new variable to set is_employee field as 0
    validate_details(new_Uname,new_Pword,new_Pnumber,re_enter_pword)
    #Runs details through function to check validity

    if valid==True: #Checks if details are valid
        app.infoBox("Corrct details","Now it will be creating") #Meesage to let me know it works
        cur.execute("INSERT INTO user_table VALUES (?,?,?,?)",["new_Uname","new_Pword","new_Pnumber",employee])
        #Inserts details from user into user table
    else:
        pass #Nothing happens till the user fixes details
```

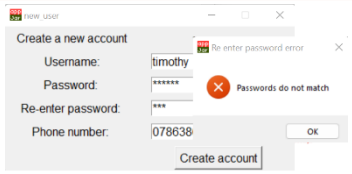


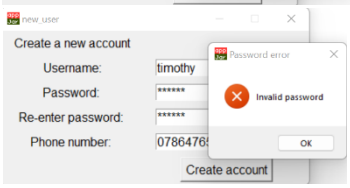
Once the details are inputted, I store them in separate variables. I stripped both username and phone number to remove any leading/trailing whitespace to reduce customers losing access to their accounts due to accident space bar clicks when creating their accounts. I have also removed spaces with .replace() within the phone number, I've done this as it is common for users to leave spaces in between numbers when typing their number out. Spaces in between their phone number will cause the validation to fail as the characters will exceed 11 with spaces in between. Therefore with .replace(), single spaces will not affect phone number validation as the program will remove spaces, causing the phone number to have 11 characters. I created a local variable, employee, that I will later use to insert into the record. As creating an account if for customers only, the program will automatically set is_employee field as 0 so it prevents them from logging in through barber login. Before inserting the data in, I run the details through the validation function to ensure only valid details are saved. If validation is successful, it will then create a new record. If validation isn't successful, the program will not do anything till correct details are inputted.

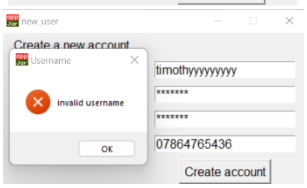
Creating the function that validates details:

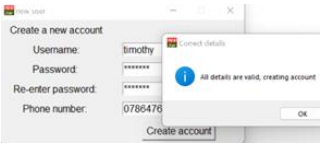
```
def validate_details(new_Uname,new_Pword,new_Pnumber,re_enter_pword):
    #Parameters are taken from the create_account()
    global valid   #Makes this variable global, usable in other functions
    valid=False   #Creates the valid variables and automatically set to false

    for x in new_Pword:   #Checks if any character are a capital letter
        if x.isupper()==True:   #Checks indiviual characters for capital letters
            capital_valid=True   #If capital letter found, set valid to True
            break   #Exits loop if capital letter found
        else:
            next  #Goes to next character
            capital_valid=False   #If no capital found, set valid to False

    for x in new_Pword:    #Checks if any character are an integer
        if x.isdigit()==True:   #Checks indiviual characters for integers
            digit_valid=True   #If integers found, set valid to True
            break    #Exits loop if integers letter found
        else:
            next   #Goes to next character
            digit_valid=False   #If no integers found, set valid to False

    if len(new_Uname)>0 and len(new_Uname)<12:   #Checks if username is between 0 to 11 characters
        if len(new_Pword)>6 and len(new_Pword)<16 and digit_valid==True and capital_valid==True and re_enter_pword==new_Pword:
        #Checks if password has a capital and interger and if between 7 to 15 characters and if both entered password match
            if len(new_Pnumber)!=11:   #Checks id phone number has 11 characters
                app.errorBox("Phone number error","Invalid phone number")   #If not 11 characters, error message
            else:
                app.infoBox("Correct details","All details are valid, creating account")   #Success
                valid=True   #Set valid to True, used in create_account()
        else:
            if re_enter_pword!=new_Pword:   #Checks if re enter password matches password
                app.errorBox("Re enter password error","Passwords do not match")   #If no matches, error message
            else:
                app.errorBox("Password error","Invalid password")   #If password doesnt follow requirments, error message

    else:
        app.errorBox("Username","invalid username")   #If username doesnt follow requirements, error message
```











This function is solely used to validate details inputted by the users during the process of creating an account. The inputs from create_account() function is passed through the function. The username, password, phone number and re-enter password are all passed through, ensuring they meet my requirements in order to get a successful validation.

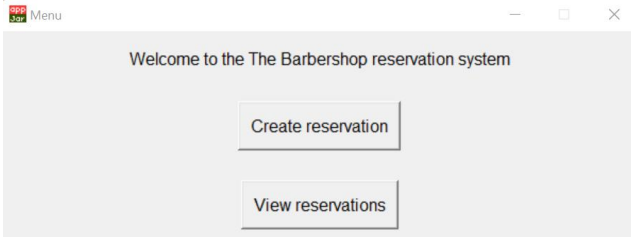
I start the function by making the valid variable as global. I converted the scope from local to global and this allows me to use this variable in create_account() to check validity before creating a new record. As the use of global variables is considered as a poor practise as it reduces modularity and flexibility. I plan on later prototypes, to remove the global variable and find another method to use this variable in other functions. I set valid as false automatically as this will reduce the amount of code I would have to repeat as there are more ways to make valid false than true. I then create 2 similar loops that check every character for a capital and integer. If one capital and one integer found, they both set there respected valid as true and exit the loop. If every character has been checked and no integers and capital are found, it sets there respected valid as false. Through a series of selection statements, it one by one checks if requirements are met, and if all are, sets valid as true. If requirements are not met, valid will not change and will let the user know what was wrong.

```
  File "C:\Users\jobia\Desktop\Jobiars work\Barbershop project\Code\2023 PROJECT
.py", line 124, in create_account
    if valid==True:
       ^^^^^
NameError: name 'valid' is not defined
```

Prior to completing the function, I came across the problem that the valid variable couldn't be used in create_account(). This generated the error. I then tried to return the variable, but I came across the same error. I then tried to declare the variable, valid, within the create_account() and then made valid as one of the parameters that was fed into the validate_detail(). After validation was completed, I tried to then return valid but came across another problem where valid within create_account() didn't change. I then tried another option by declaring valid as global and this worked allowing me to access them from both functions. I know that I can pass valid along as an argument to create_account() to make it work as a local variable or make return work, but as the global method works, I will keep the variable as global and in later prototype change it.

Main menu window:

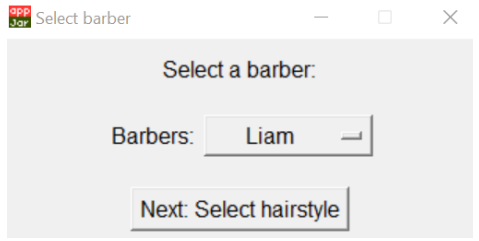
```
##### MAIN MENU #####
app.startSubWindow("Menu",modal=False) #Create a new subwindow
app.setSize("600x200") #Sets size of window
app.setResizable(False) #Fixes size of window

app.addLabel("Welcome","Welcome to the The Barbershop reservation system")
#Title of the window

app.setInPadding([5,7]) #Changes button size
app.addButton("Create reservation",tempbutton) #New Reservation button

app.setInPadding([5,7]) #Changes button size
app.addButton("View reservations",tempbutton) #View reservation button
app.stopSubWindow() #Stops the window
```

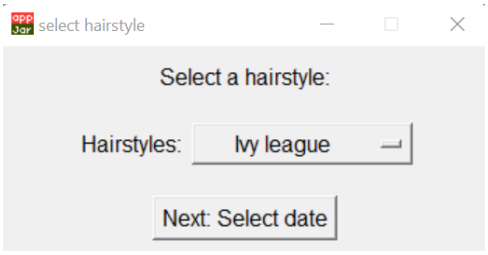


This is my main menu window that can be jumped to from the home button I plan to add and after the login windows. This window will only contain 2 buttons that will either lead a new reservation or viewing reservations. Again, kept functional only for now, but in later prototypes I plan on adding barbershops, logos, colours and icons.

Selecting barber/hairstyle window:

```
##### SELECT BARBER #####
app.startSubWindow("Select barber") #Creates a new subwindow
app.setSize("350x150") #Sets the size of the window
app.setResizable(False) #Prevents change to size window

app.addLabel("barber_pick","Select a barber:",0,0)
app.addLabelOptionBox("Barbers: ", [" Liam ", " Lucus ", " Dave "," Timothy "])
#Displays all barber options in option box
app.addButton("Next: Select hairstyle",select_hairstyle)
#Confirms selection and moves to hairstyle subwindow
app.stopSubWindow()

##### SELECT HAIRSTYLE #####
app.startSubWindow("select hairstyle") #Creates a new subwindow
app.setSize("350x150") #Sets the size of the window
app.setResizable(False) #Prevents change to size window

app.addLabel("hairstyle_pick","Select a hairstyle:",0,0)
app.addLabelOptionBox("Hairstyles: ", ["Buzz cut ", "French crop", "Curtains","(
#Displays all hairstyle options in option box
app.addButton("Next: Select date",select_date)
#Confirm hairstyle and moves to date selection
```



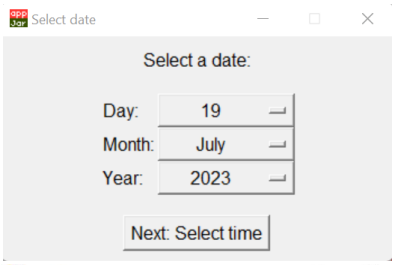


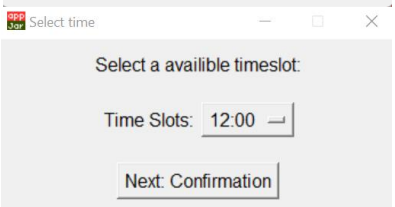
These are the two windows that are accessed after clicking “create reservation” button on the main menu. These windows consist of 2 option boxes and two button that both lead to the adjacent window. The option boxes, once clicked on, display all the options available for the user.

I decided to use option boxes instead of multiple buttons as this simplifies the windows by reducing the clutter that many buttons would have caused. This further makes it easier for customers to navigate as it will not overwhelm them with over 30 hairstyle options on a single window. Making use of the option box also benefits me as it reduces redundant complexity of the code, as each button would have required code to evenly space out and place each option button in the window.

Selecting date/time window:

```
##### SELECT DATE #####
app.startSubWindow("Select date") #Creates a new subwindow
app.setSize("350x200") #Sets the size of the window
app.setResizable(False) #Prevents change to size window

app.addLabel("date_pick","Select a date:",0,0)
app.addDatePicker("date",2,0) #Select any days of the year
app.setDatePickerRange("date", 2023, endYear=2023) #Limits the range to 2023
app.addButton("Next: Select time",select_time) #Confirms and moves to time selection

##### SELECT TIME #####
app.startSubWindow("Select time") #Creates a new subwindow
app.setSize("350x150") #Sets the size of the window
app.setResizable(False) #Prevents change to size window

app.addLabel("time_pick","Select a availible timeslot:",3,0)
app.addLabelOptionBox("Time Slots: ",[],4,0)#Creates option box for availible times
app.addButton("Next: Confirmation",confirmation_screen) #Confirms and moves to confirmation
```





These windows are displayed after selecting a hairstyle. For selecting a date, I made use of AppJar built in date picker. This allows the user to choose from any date they need within that year. My client decided to limit the year to 2023 as it wouldn’t be practical to allow user to create reservations multiple years from now or in the past. To do this, I utilised date picker range command to limit the year to 2023 only.

I again decided to not go ahead of all buttons approach as it would clutter the screen too much with all the options, overwhelming the user. This approach simplifies the screen further, allowing for faster and easier navigation as users may be familiar with this design from other sites. The date picker was also more efficient compared to multiple buttons, instead of only being able to display one week of dates only, the user will be able to access all dates within the year in much smaller space.

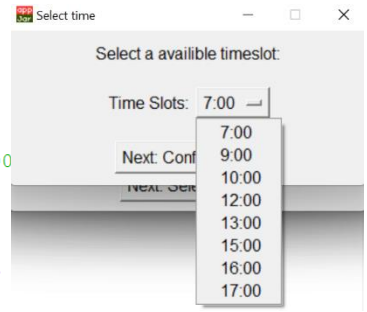
For time selector, I originally planned on alerting the user if a time slot was unavailable after they selected a time slot, my client warned me that this will only frustrate users trying to find available time slots. So instead, I only displayed available times by creating a separate function to filter out unavailable time slots. When displaying the time slots, I left the entry as blank so the function can later update the options with available time slots only.

After selecting a time slot, the next button leads to the confirmation page.

Creating the function to filter out unavailable times:

```
def get_times():

    d_pick=app.getDatePicker("date")  #Stores customers date input
    b_pick=app.getOptionBox("Barbers: ")  #Stores cutomers barber input

    cur.execute("SELECT reservation_time FROM reservation_table WHERE barber_name=? AND reservation_date=?",[b_pick,d_pick])
    #Finds reservations with barber / date input
    results=cur.fetchall()
    #Stores all the reseults in a 2D list

    bookedTimes=[] #Creating a blank 1D list for availible times
    for eachTime in results: #Passes through each result
        bookedTimes.append(eachTime[0]) #Insert all results into new 1D list

    time_slots=['7:00','8:00','9:00','10:00','11:00','12:00','13:00','14:00','15:00','16:00
    #List with all availible time slots

    availible_times=[] #Creates a new blank list for all availible times
    for index_x in time_slots: #Passes through each index in time slots
        if index_x not in bookedTimes: #Checks if searched index found within booked times
            availible_times.append(index_x) #If not, adds into new list

    app.changeOptionBox("Time Slots: ",availible_times)
    #Updates option box with availible times
```

```
def select_time():
    get_times()
    app.showSubWindow("Select time")
```



This function is responsible for filtering out all unavailable time slots based on the user's input. This function is run before displaying the selecting time slot window as it can filter out unavailable times before the user has the option to select a time slot. To filter out the correct timeslots, I search for reservations using the barber and date data the user had selected prior and stored all those results. The function then transferred the results from a 2D list to a 1D list called booked time by using a loop. I then created a list with all the timeslots the barbershop offers. Using iteration, checked though each index in time slots. Then using selection, checked if index in time slots is within booked times list. If it was not present, insert that index into a new list, available times. After the list of available times had been created, I updated the time slot option with the available time slots, so it no longer showed blank.

```
def get_times():
    d_pick=app.getDatePicker("date")
    b_pick=app.getOptionBox("Barbers: ")

    cur.execute("SELECT reservation_time FROM
    results=cur.fetchall()
    results=list(results)

    time_slots=['7:00','8:00','9:00','10:00','1

    print(time_slots)
    print(results)
    print("~~~~~~")

    availible_times=[]
    for index_x in time_slots:
        if index_x not in results:
            availible_times.append(index_x)
        else:
            pass
    print(availible_times)
```

```
['7:00', '8:00', '9:00', '10:00', '11:00', '12:00', '13:00',
16:00', '17:00']
[('8:00',), ('11:00',), ('14:00',)]
~~~~~~
['7:00', '8:00', '9:00', '10:00', '11:00', '12:00', '13:00',
16:00', '17:00']
```

The first error I came across was that I tried to compare both 1D list, time slots, and the 2D list, results. This prevented me from be able to generate a new list with only available timeslots. I printed out all the results to see if the issue was with the results but realised that all the correct values were being outputted. At the time I was unaware that SQLite fetchall command stored the results as a 2D list, and that I couldn't compare both types of lists as the 2D list had 2 values for each index, where the 1D list had 1 value for each index. Still unaware that SQLite used 2D lists, I then tried another approach using the data structure sets.

```
def get_times():
    d_pick=app.getDatePicker("date")
    b_pick=app.getOptionBox("Barbers: ")

    cur.execute("SELECT reservation_time FROM reservation_table WHERE barber_name=? AND reservation_date=?",[b_pick,d_pick])
    results=cur.fetchall()
    results=set(results)

    time_slots={"7:00","8:00","9:00","10:00","11:00","12:00","13:00","14:00","15:00","16:00","17:00"}

    availible_times=(time_slots-results)
    print (availible_times)
```

```
['13:00', '9:00', '17:00', '7:00', '8:00', '14:00', '11:00', '12:00', '10:00',
15:00', '16:00']
```

I researched online and found out that python had another data structure, sets. I learned that sets could be subtracted from another set. I first converted the results from the SQL statement to a sets data structure by using the sets constructure. With the time slots, instead of using square brackets, I used curly brackets to indicate that this was a set. Within my function, I deducted the time slot set with the results sets and outputted the values to see if it had worked. It had produced a new list of value that had removed unavailable timeslots. But then the new problem was that sets unordered items. This method wouldn't not have worked for me as the list of times had to be ordered for the option box. I considered finding out a way to reorder the items but realised that this solution would be too complex for a simple problem. I researched more about SQLite and then found out that SQLite used 2D lists. This led to my current solution to my problem.

Confirmation window:


```
##### CONFIRMATION SCREEN #####
app.startSubWindow("Confirmation") #Creates a new subwindow
app.setSize("400x250") #Sets the size of the window
app.setResizable(False) #Prevents change to size window

app.addLabel("R_details","Reservation details")
app.addLabel("final date",[]) #Add a blank label for final date
app.addLabel("final time",[]) #Add a blank label for final time
app.addLabel("final barber",[]) #Add a blank label for final barber
app.addLabel("final hairstyle",[]) #Add a blank label for final hairstyle
app.addLabel("final price",[]) #Add a blank label for final price
app.addButton("Confirm reservation",insert_reservation)
#Confirms details and runs function to insert into table
app.go()
```

This window is displayed after selecting a time. This window consists of a button that, once clicked, leads to another function to create a new reservation that creates a new record within the reservation table. It has blank labels that represent all the details the user has inputted prior and will be updated with the correct information before the user is presented with this page.

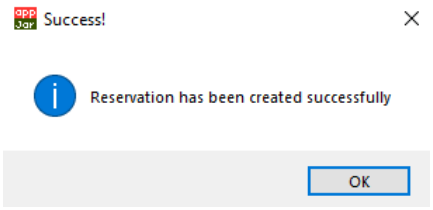
Creating the confirmation function:

```
def confirmation():
    final_date=app.getDatePicker("date") #Stores customers date input
    final_time=app.getOptionBox("Time Slots: ") #Stores customers time input
    final_barber=app.getOptionBox("Barbers: ") #Stores customers barber input
    final_hairstyle=app.getOptionBox("Hairstyles: ") #Stores customers hairstyle input
    cur.execute("SELECT price FROM hairstyle_table WHERE hairstyle=?",[final_hairstyle])
    #Searches for the price of the customers hairstyle
    final_price=cur.fetchone() #Stores the price of a variable

    price_p=[] #Creating a blank 1D list
    price_p.append(final_price[0]) #Inserts price results in 1D list

    app.setLabel("final date","Date: "+str(final_date)) #Update date label
    app.setLabel("final time","Time: "+str(final_time)) #Update time label
    app.setLabel("final barber","Barber: "+str(final_barber)) #Update barber label
    app.setLabel("final hairstyle","Hairstyle: "+str(final_hairstyle)) #Update hairstyle label
    app.setLabel("final price","Price: £"+str(price_p[0])) #Update price label
```

```
def confirmation_screen():
    confirmation()
    app.showSubWindow("Confirmation")
```

This functions outputs all the customers inputs during the reservation process. This gives the users a chance to review all their details to ensure that no mistake was made. The function is run before running the confirmation window interface as this update the labels before being displayed to the user. The function stores all the inputs the customers made. Using a SQL statement, the price is searched from the hairstyle table using the hairstyle the user had inputted prior. It then updates all labels within confirmation window code so that all the details are outputted for the customer. The confirm reservation button then runs another function to create a new record in the reservation table.

Creating function to save a reservation:




```
def insert_reservation():
    username=app.getEntry("inputU")#Stores customers username input
    final_date=app.getDatePicker("date") #Stores customers date input
    final_time=app.getOptionBox("Time Slots: ") #Stores customers time input
    final_barber=app.getOptionBox("Barbers: ") #Stores customers barber input
    final_hairstyle=app.getOptionBox("Hairstyles: ") #Stores customers hairstyle input
    cur.execute("SELECT price FROM hairstyle_table WHERE hairstyle=?",[final_hairstyle])
    #Searches for the price of the customers hairstyle
    final_price=cur.fetchone() #Stores the price of a variable

    cur.execute("INSERT INTO reservation_table (barber_name,reservation_time,reservation_date,hairstyle,username)
                VALUES (?,?,?,?,?)",[final_barber,final_time,final_date,final_hairstyle,username])
    #Creates a new record in reservation table with all relevent fields filled out
    con.commit()
    app.infoBox("Success!","Reservation has been created successfully")
    #Lets customer know that the reservation has been created
    app.showSubWindow("Menu")
    #Jumps back to main menu at the end
```

Once the customer looks over the details and presses confirm reservation, this function is run. First, it creates local variables that store all the user inputs from the reservation windows. Using a SQL statement, the price is searched from the hairstyle table using the hairstyle the user had inputted prior. Once all the details have been stores, using another SQL statement, the function inserts the details into the reservation tables with all the required fields. As I used auto increment when creating this table, this automatically creates a primary key for each record as each records primary key consists of the previous records primary key plus one. This ensures that each record is unique, also allowing me to ignore the field when inserting data. Once the reservation has been saved, the user is alerted via an info box that informs them that the process has been successful. After OK has been clicked, program jumps back to the main menu screen as they have reached the end of the reservation process.

Prototype 1: Evaluation

No	Criteria	Have I met?	Improvements for later prototypes
1	Account/login		Two factor authentication
2	Design		Currently has no priority, improve on this in prototype 3
3	Create reservations		Able to create reservation, slight changes needed
4	Database		Add an email field in user table for reminders
5	Recommend hairstyles		One of the main priorities in prototype 2
6	Edit hairstyles		Another priority for prototype 2
7	Hardware compatible		No improvements, compatible with lower end computers
8	Prevent double. booking		No improvements as it fully functional
9	Menu/home buttons		Add home button on every page / employee's windows in prototype 2
10	Cancel reservation		Another priority in prototype 2

Feedback from client:

Speaking with my client, they were pleased that the program is functional as it is, but he does still have minor improvements to the core functionalities.

One piece of feedback was to create the windows for the employees only, mainly being the calendar system for each barber reservations. This was due for prototype 1 but due to time pressures, I decided to create this in prototype 2 and finish off more essential features.

Lastly, he requested, that every window should have a home button or a button to go back to previous windows to allow customers to fix mistakes they may have made. I was also planning on completing this in prototype one but again, due to time constraints, had to push it to prototype 2. In prototype 2, I plan on adding on every window a home button that will, once clicked, return to the main menu.

Prototype 2: Creating all additional features/improve quality of life.

These are the features I intend to add for the second prototype:

-Generates a recommended hairstyle based on users' inputs

-Home button on every window to jump back to menu

-Function to view and cancel reservations

-Barber calendar

-Add a message box for customers to send to barber details, within confirmation window

-When opening a new window, program closes previous window

-Add email system, where users can be sent receipts / 2 factor authentication

-Validate the date picked to ensure it isn't in the past

-Preventing 2 accounts with same details

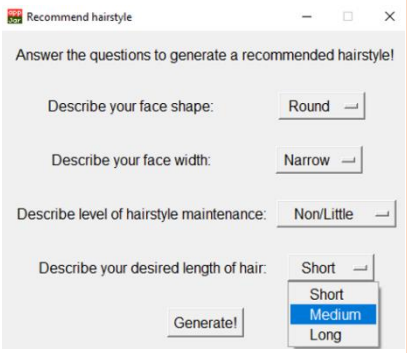
-Delete all old reservation automatically

-Barbers edit hairstyle table within program

-Prevent barber account from login into customer side of program

-Function that fetches all hairstyles

Creating recommended hairstyle window:




```
##### RECOMMENDED HAIRSTYLE #####
app.startSubWindow("Recommend hairstyle") #Creates a new subwindow
app.setSize("450x350") #Sets the size of the window
app.setResizable(False) #Prevents change to size window

app.addLabel("R_hairstyle","Answer the questions to generate a reco
#Lets users what to do
app.addLabelOptionBox("Describe your face shape:        ",["Round","Square"])
app.addLabelOptionBox("Describe your face width:        ",["Narrow","Wide"])
app.addLabelOptionBox("Describe level of hairstyle maintenance:  ",["Non/Little","Medium","I
app.addLabelOptionBox("Describe your desired length of hair:  ",["Short","Medium","Long"])
#These provide options for user to select, whilst asking questions
app.addButton("Generate!",generate_hairstyle)
#Onces clicked, runs a function to generate hairstyle
```

This window is presented to the customer if they select the not sure button that I added on the select hairstyle window. Leads them to a window with multiple option boxes that allow them to input their answers and button at the bottom to feed their answer through the algorithm to generate a recommended hairstyle.

Generating hairstyle function:

```
def generate_RH():
    head_shape=app.getOptionBox("Describe your face shape:       ") #Stores face shape input
    head_width=app.getOptionBox("Describe your face width:       ") #Stores face width input
    u_maintainence=app.getOptionBox("Describe level of hairstyle maintenance:  ") #Stores maintainance input
    hair_length=app.getOptionBox("Describe your desired length of hair:  ") #Stores hair length input

    cur.execute("SELECT hairstyle FROM hairstyle_table WHERE face_shape=? AND face_width=? AND maintainence=? AND hair_leng
                ,[head_shape,head_width,u_maintainence,hair_length])
    #Searches for hairstyles that attributes matches the customers input
    r_hairstyle=cur.fetchall() #Stores search results into 2D list

    if not r_hairstyle: #Checks if any results where found
        app.changeOptionBox("Hairstyles: ", ["Buzz Cut","French Crop","Curtains","Quiff","Slickback","Crew Cut","Man
        #Resets the option box, incase they have already altered the list
        app.errorBox("No matches","No hairstyles found, please select your own")
        #Error message, alerting no matches where found, so it resets the options


    else: #Run if results where found
        recommended_hairstyle=[] #Creats empty list
        for x in r_hairstyle: #Checks every index
            recommended_hairstyle.append(x[0]) #Takes the first value and places in list
        app.changeOptionBox("Hairstyles: ",recommended_hairstyle)
        #Updates availible option boxes with generated hairstyles
        app.infoBox("Matches","Some recommendation have been found!")
        #Alerts user that matches have been found
```

```
def reset_hairstyle():
    app.changeOptionBox("Hairstyles: ", ["Buzz Cut","French Cr
    #Resets list of options
    app.showSubWindow("Select hairstyle")
    #Open the window with original options
```

This function is run after the customer clicks the generate button after selecting their answers. It first stores all the inputs as local variables that are then used to search the hairstyle table for hairstyles with the attributes that match with the customers inputs.

In the SQL statement, I used all AND gates as my client wanted all the customers input to match the hairstyle to provide a better match, however this results in more combination of input that don't match any hairstyle and therefore a higher chance of a no hairstyles being found. I suggested to my client that instead of used AND for face shape and width, I would use OR as this would only produce a slight inaccurate result as only one of face shape and width must be true whilst maintenance and length would both still have to be true. The reason I suggested this is that it will significantly reduce the chance of no hairstyle being found. But as per my client request, he said that he would rather sacrifice more results for a very accurate result, as in the end, the customers will be more pleased with their experience and haircut at the barbershop. The program then stores all the results found.

Next, the program checks if any results were found. If no results were found, it would reset the list of hairstyles back to the original and alert the user that no results were found and that they should select their own. The reason the program resets the list is that as this questionnaire can be taken multiple times, producing multiple lists. Once no results have been found after taking the test multiple times, the list of hairstyles will need to reset to allow the customer to access to them all again. If results have been found, it will take the 2D list results and convert them into a 1D list so that they display correctly in the list of hairstyles. It will update the list of hairstyle options within selecting hairstyle window and alert the customer that matches have been made.

Originally, I planned on displaying the recommended hairstyles options in another window that is separate from the original hairstyle window. But I came across a problem where once I created a new option box, the rest of the program will have to be altered to accommodate the extra variable that can store the customer hairstyle input. This overcomplicated the program more by having to check which variable contained the customer input, so I then decided to combine the window with selecting hairstyle window. This kept the customer input in one variable and prevented change to the rest of the program.

If the customer didn't like the generated result, I added a button that resets the list of hairstyle options so they can pick what they like instead.

Closing previous windows:

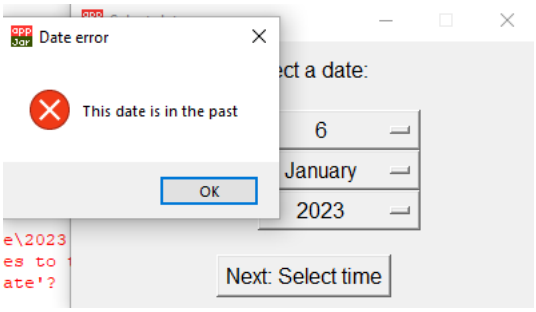
```
def recommend_hairstyle():
    app.hideAllSubWindows(useStopFunction=False)
    app.showSubWindow("Recommend hairstyle")

def generate_hairstyle():
    generate_RH()
    app.hideAllSubWindows(useStopFunction=False)
    app.showSubWindow("Select hairstyle")

def home():
    app.hideAllSubWindows(useStopFunction=False)
    app.showSubWindow("Menu")

def b_home():
    app.deleteAllGridRows("B_view")
    app.hideAllSubWindows(useStopFunction=False)
    app.showSubWindow("Barber: select date")

def barber_reservation():
    app.hideAllSubWindows(useStopFunction=False)
    barber_get_reservation()


def new_hairstyle():
    app.hideAllSubWindows(useStopFunction=False)
    app.showSubWindow("New hairstyle")
```

```
def select_barber():
    app.hideAllSubWindows(useStopFunction=False)
    app.showSubWindow("Select barber")

def select_hairstyle():
    app.hideAllSubWindows(useStopFunction=False)
    app.showSubWindow("Select hairstyle")

def select_date():
    app.hideAllSubWindows(useStopFunction=False)
    app.showSubWindow("Select date")

def select_time():
    get_times()
    app.hideAllSubWindows(useStopFunction=False)
    app.showSubWindow("Select time")

def confirmation_screen():
    confirmation()
    app.hideAllSubWindows(useStopFunction=False)
    app.showSubWindow("Confirmation")
```

With this extra line of code, once a new window is open, it closes the previous window. This is to prevent cluttering on the customer's screen with all the windows that have been opened prior. With this being introduced, I must provide a way for customers to jump back to previous windows or menu as they can no longer reopen the window they want.

Creating the function to validate the date:

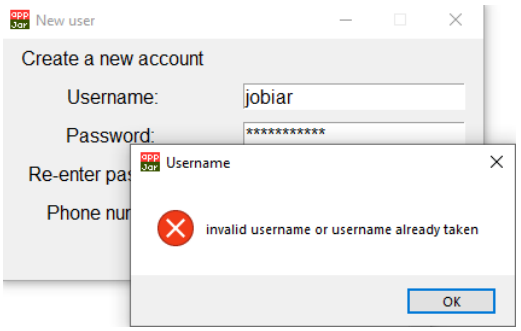
```
from datetime import date
def valid_date():
    date=app.getDatePicker("date") #Stores customer date input
    today=date.today() #Stores the current date

    if date<today: #Checks if date picked is in the past
        app.errorBox("Date error","This date is in the past")
        #Error message if in past
    else:
        select_time()
        #Continues the program
```



This function uses the python library, date, allowing me to access the current date. This function stores both date the customer picked and the date that is currently on that day. It will then compare both variable and if the date is in the past, it will provide an error message but if not, it will continue the program. This function is run after the customer clicks the select time button on the date window. My client reminded me that the program should prevent booking reservations in the past as it doesn't make sense to allow customers to create reservation in the past. As I didn't know how to get the current time, I researched online and came across a website explaining how to utilise this library.

Modifying validate function to check if duplicate account exists:

```
cur.execute("SELECT username FROM user_table WHERE username=?", [new_Uname])
#Searches for username that matchs with user input
tbl_username=cur.fetchall() #Stores results as variables
if not tbl_username: #Checks if any results where found
    username_valid=True #If no duplicates, set valid as true
else:
    username_valid=False #If duplicate, set valid as false


if len(new_Uname)>0 and len(new_Uname)<12 and username_valid==True:
    #Checks if username is between 0 to 11 characters and unique
```



I modified the validate function by also checking if an existing username exists. As the username is the primary key of the user table, I had to ensure only unique accounts were created otherwise defeat the purpose of the primary key being unique. The reason I changed it was that if a new account is made with an existing username, both can create reservations under the same name causing confusion for the barbers in the barbershop. Another reason is to prevent any other errors arising in the future with the duplicate username.

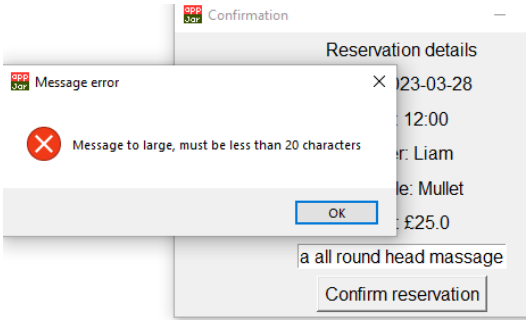
The function now also fetches all usernames present within the user table with the same username inputted by the user and stores the results as a variable. It then checks if any results have been produced. If no results have been produced, it sets valid as true but if there are results being produced, sets valid as false. If valid is true, it means that no other accounts share the username, and vice versa. The function now also checks if username valid is true as well as if it between a certain range of characters. To alert the user if any error had been made, I modified the error message to include the fact that duplicate usernames are also not allowed.

Creating the option to input messages:

```
##### CONFIRMATION SCREEN #####
app.startSubWindow("Confirmation") #Creates a new subwindow
app.setSize("400x250") #Sets the size of the window
app.setResizable(False) #Prevents change to size window

app.addLabel("R_details","Reservation details")
app.addLabel("final date",[]) #Add a blank label for final date
app.addLabel("final time",[]) #Add a blank label for final time
app.addLabel("final barber",[]) #Add a blank label for final barber
app.addLabel("final hairstyle",[]) #Add a blank label for final hairstyle
app.addLabel("final price",[]) #Add a blank label for final price
app.addEntry("message") #Entry box for messages for the barber
app.addButton("Confirm reservation",insert_reservation)
#Confirms details and runs function to insert into table
app.stopSubWindow()
```

```
if len(final_message)>20:
    app.errorBox("Message error","Message to large, must be less than 20 characters")
else:
    cur.execute("INSERT INTO reservation_table (barber_name,reservation_time,reservati
    #Creates a new record in reservation table with all relevent fields filled out
    con.commit()
    app.infoBox("Success!","Reservation has been created successfully")
    #Lets customer know that the reservation has been created
    send_reciept()
    app.showSubWindow("Menu")
    #Jumps back to main menu at the end
```

```
cute("""CREATE TABLE "reservation_table" (
 "reservationID"     INTEGER NOT NULL UNIQUE,
 "barber_name"       TEXT NOT NULL,
 "reservation_time"  REAL NOT NULL,
 "reservation_date"  DATE NOT NULL,
 "hairstyle" TEXT NOT NULL,
 "username"  TEXT NOT NULL,
 "message"   TEXT,
 PRIMARY KEY("reservationID" AUTOINCREMENT)
```



The program now also gives the option to add a message for the barber regarding their haircut. This information will be stored within the reservation table within its own field that has been created. To insert this, I had to run the function that creates the database again and delete any older version. As my client wanted this to be optional. The program does accept blank input boxes, done by leaving the field without not null command.

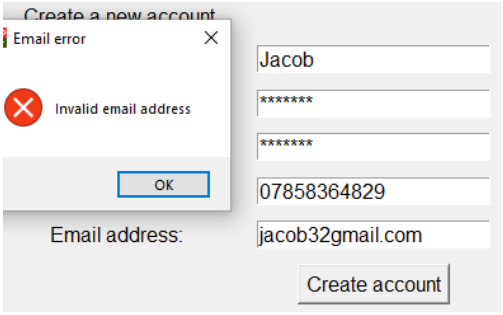
Before inserting into the table, I validate the information by restricting to 20 characters to prevent unnecessarily large database, this is done during the insert reservation function. Once the check has been made, it inserts the reservation into the reservation table.

Preparing for email functions:

```
##### CREATE NEW USER #####
app.startSubWindow("New user") #Creating new subwindow
app.setSize("400x250") #Sets the size of the window
app.setResizable(False) #Fixes the size of window

app.addLabel("new user title","Create a new account",1,0) #Title of window
app.addLabel("username1", "Username:",2,0)
app.addEntry("new_username",2,1) #Input box for new username
app.addLabel("password2","Password: ",3,0)
app.addSecretEntry("new_password",3,1) #Input box for new password
app.addLabel("repassword2","Re-enter password: ",4,0)
app.addSecretEntry("new_password2",4,1) #Input box to check if password matches
app.addLabel("phone","Phone number: ",5,0)
app.addEntry("new_phone",5,1) #Input box for phone number
app.addLabel("email","Email address: ",6,0)
app.addEntry("user_email",6,1) #Input box for email address
app.addButton("Create account",create_account,7,1) #Button leads to validating / inserting
app.stopSubWindow() #Stops the window
```

```
for x in new_email:   #Checks if any character have @
    if x=="@":   #Checks indiviual characters for @
        email_valid=True   #If @ found, set valid to True
        break   #Exits loop if capital letter found
    else:
        next  #Goes to next character
        email_valid=False   #If no @ found, set valid to False

if email_valid==True:
    app.infoBox("Correct details","All details are valid, creating account")   #Success
    valid=True   #Set valid to True, used in create_account()
else:
    app.errorBox("Email error","Invalid email address")
```

```
ecute("""CREATE TABLE "user_table" (
 "username"   TEXT NOT NULL UNIQUE,
  "password"  TEXT NOT NULL,
  "phone_number"      TEXT NOT NULL,
  "email"     TEXT NOT NULL,
  "is_employee"       INTEGER NOT NULL,
  PRIMARY KEY("username")
```



I've made a series of edits to my program to accommodate for an email field that I will later use for 2 factor authentication and sending receipts. The first edit I made was by adding another field within user table that will contain email addresses for all accounts, I've made the field not null as the program will need to verify the customer, using the email authentication, to log in to their account.

Next within the create account window, I added another input box for users to input their email addresses. Once they click create account button the program will now also insert the email into the database. Before the email is inserted into it is validated within the validate detail's function. I validate the email by checking every character for the @ character and setting a variable valid to be either true or false, this will be later used during the nested selection statements. Once the email has been validated along with all the other details, it inserts the details within the database.

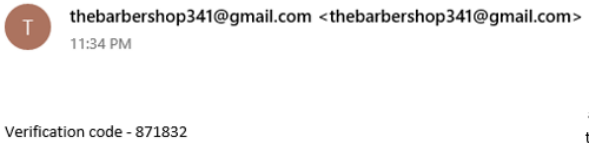
Creating home button function:

```
def home():
    app.hideAllSubWindows(useStopFunction=False)
    app.showSubWindow("Menu")
app.addIconButton("home0",home,"home", align=None) #Create the icon button
```

I've created this small function that re opens the main menu once its clicked. I made use of AppJar inbuilt icons. It defines the button along with adding the image of the home icon. This allows the home button to be easily identified on the window. I've added this button on every window after the main menu is presented.

Creating the function that sends the authentication code to customer:

```
def send_authenticate_code():
    username=app.getEntry("inputU") #Stores the customer username
    cur.execute("SELECT email FROM user_table WHERE username=?",[username])
    #Fetches email within the user table with the customers username
    tbl_email=cur.fetchone() #Stores customer email address

    email_from = 'thebarbershop341@gmail.com' #Stores the sender email
    password = 'dvxyukjbjyaaqmas' #Password that gains access to senders emails
    email_to=str(tbl_email[0]) #Sets the recipient email as customer email

    global verify_code
    #Sets the verification code as global to use in other function
    verify_code=random.randint(100000,999999)
    #Generates the authentication code using random library
    email_content=("Verification code - "+str(verify_code))
    #Email content itself that going to be sent
```

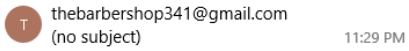


The next series of functions are all for two factor authentication.

This function generates the authentication code by using the imported random library. The function first stores all the customer username as a variable and using the username, fetches the customers email using the SQL statement. I set the email being sent as a global variable as I will need this in another function. The email will consist of the authentication code only. In order to generate a code, I used the randint command from the random library to generate a value within a specified range and convert it from an integer to a string so I can compare it to the customers input on the authentication window I will be creating next.

To send an email within python, I had to use the python libraries smtplib and ssl. This allowed to send plaintext emails to only one recipient. I first created a Gmail account for my client where within the email setting, gained access to password that allowed programs, such as python, to use the email address as the password verified that it was used by authorised user. The program will now only use this email address to send all the barbershops email, making it easier to manage for me and my client. As I didn't know how to send emails using python, I had to do online research where I came across tutorial on YouTube and other websites that gave an explanation and a template of code on how to send emails. Using this template, I had to modify it so it would fulfil my use.

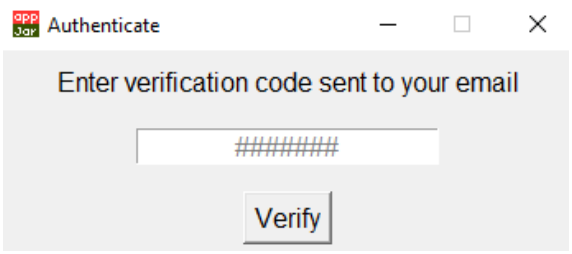
```
email_content=("Verification code: "+str(verify_code))
#Email content itself that going to be sent
```



One problem that I faced was the content of the email was not sending if I used a colon character within the string. I initially though the problem was the whole string itself but when I tested with the verification code and nothing else, it worked perfectly fine. I kept on researching other methods to add variables within string such as using f"{verify code}" and %s but all produced the same results. After trial and error, I tried different text before verify code variable and it worked. After repeating this process, I determined that the problem was within colon and since I didn't know why it caused a problem, I switched it for a dash instead.

Authentication window:

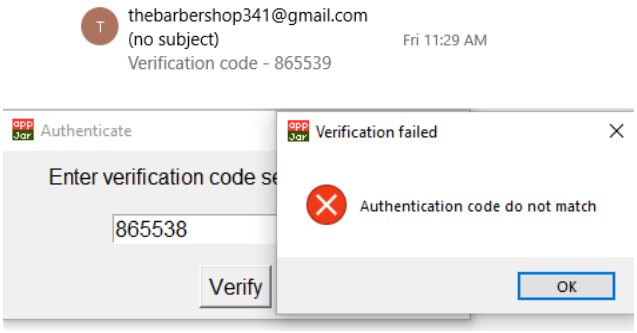
```
##### 2 FACTOR AUTHENTICATION #####
app.startSubWindow("Authenticate") #Creates new subwindow
app.setSize("350x125") #Sets size of window
app.setResizable(False) #Fixes size of window
app.addLabel("sd","Enter verification code sent to your email")
app.addEntry("code_input") #Entry box for user to input their code
app.addButton("Verify",authenticate_code)
#Once clicked run the validate function
app.setEntryDefault("code_input","#######")
app.stopSubWindow() #Stops the window
```



This window is displayed after the user has successfully inputted the details to an account. To verify their identity further, they must input the verification code that is sent to their email. Once they input the correct code, they have been successfully verified and are presented to the menu page. However, if they input the code wrong, the program rejects access to the program till the code match up. This is be kept simple by only consisting of the label, input box and button.

Creating the function to compare authentication code:

```
def authenticate_code():
    user_code=app.getEntry("code_input") #Stores user input

    if user_code!=str(verify_code): #Compare user code to actual code
        app.errorBox("Verification failed","Authentication code do not match")
        #Error message if code a message dont match
    else:
        app.hideSubWindow("Authenticate")
        app.showSubWindow("Menu")
        #If code match up, it present the menu page
```



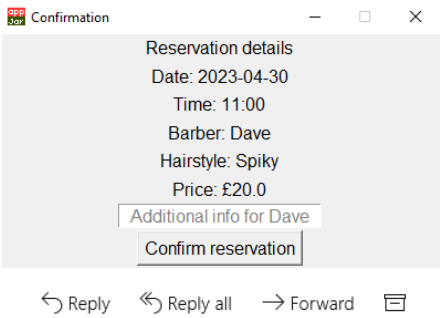
This function is run after the user clicks on the verify. The function first stores the users input as a variable and uses it to compare to the code sent to their email address. If the code doesn't match up, it will give an error message and if they do it will display the menu. When using the verify code variable, I had to convert it to a string as when the code was generated, it was stored as an integer. If I compared the user inputted code, which is a string, to the verify code, an integer, it produces an error as they are different data types.

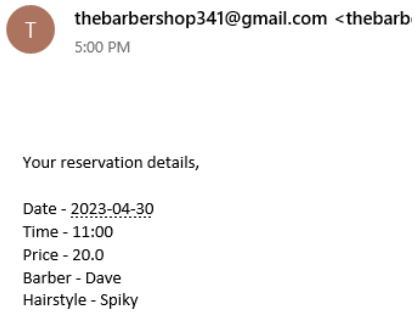
I've considered that if they input an incorrect code, the program will send the user another verification code as another layer of security, I would have done this by giving the user an error message and running the send authentication code function again that will then update the verify code variable to the new code. This would prevent the old code working

Creating the function that sends receipts to customer:

```
def send_reciept():
    username=app.getEntry("inputU") #Stores the customer username
    cur.execute("SELECT email FROM user_table WHERE username=?",[username])
    #Fetches email within the user table with the customers username
    tbl_email=cur.fetchone() #Stores customer email address

    final_date=app.getDatePicker("date") #Stores customers date input
    final_time=app.getOptionBox("Time Slots: ") #Stores customers time input
    final_barber=app.getOptionBox("Barbers: ") #Stores customers barber input
    final_hairstyle=app.getOptionBox("Hairstyles: ") #Stores customers hairstyle input
    cur.execute("SELECT price FROM hairstyle_table WHERE hairstyle=?",[final_hairstyle])
    #Searches for the price of the customers hairstyle
    final_price=cur.fetchone() #Stores the price of a variable

    email_from = 'thebarbershop341@gmail.com' #Stores the sender email
    password = 'dvxyukjbjyaaqmas' #Password that gains access to senders emails
    email_to=str(tbl_email[0]) #Sets the recipient email as customer email

    email_content=f"""
    Your reservation details,

    Date - {final_date}
    Time - {final_time}
    Price - {final_price[0]}
    Barber - {final_barber}
    Hairstyle - {final_hairstyle}"""
    #Email that will be sent to the user

    context = ssl.create_default_context()
    #Creates a secure connection with server
    with smtplib.SMTP_SSL("smtp.gmail.com", 465, context=context) as server:
        server.login(email_from, password) #Login to senders email address
        server.sendmail(email_from, email_to,email_content)
        #Sends the email with recipient details and email itself
```

```
else:
    cur.execute("INSERT INTO reservation_table (barber_name,rese
    #Creates a new record in reservation table with all relevent
    con.commit()
    app.infoBox("Success!","Reservation has been created success
    #Lets customer know that the reservation has been created
    send_reciept()
    app.hideSubWindow("Confirmation")
    app.showSubWindow("Menu")
    #Jumps back to main menu at the end
```





This function is responsible for sending the receipt after the user has confirmed their reservation. When the confirm reservation button is clicked it runs the insert reservation function that now also runs this function once it has successfully inserted the reservation into the reservation table.

This function is similar to the function that sends the authentication code, the difference is that it retrieves and stores all the data regarding the reservation within variables. When storing the customer email, I stored the index 0 of it as SQLite stores the results within a list. When trying to then send the email, it would not send as the email was in the list. The simple fix was storing the index that contains the email only, storing the email only and not the whole list. When displaying the email, I made use of the triple speech marks that allows breaks to happen within the string that prevents details being displayed all on the same line. This presents the details in a logical structure, displaying all the relevant details that were in the variables defined at the beginning.

A feature that my client wanted was a reminder system where another email was sent prior to the reservation. After researching online on how to do this, I've found methods that can do this but once the program is closed, it wouldn't work. I've seen other methods that involve other pieces of software that are much more complex and will take too much time for me to successful implement into my program within my deadline. I'm certain that I would not be able to implement this feature, however if I have enough time in prototype 3. I will attempt to implement this feature.

Converting valid variable from global to local:

```
create_account():
new_Uname=app.getEntry("new_username").strip() #Stores username input as a variable
new_Pword=app.getEntry("new_password") #Stores password input as a variable
re_enter_pword=app.getEntry("new_password2") #Stores re enter password as a variable
new_Pnumber=app.getEntry("new_phone").replace(" ", "").strip() #Stores phone number as a variable
new_email=app.getEntry("user_email")

employee=0 #Creates a new variable to set is_employee field as 0
valid=validate_details(new_Uname,new_Pword,new_Pnumber,re_enter_pword,new_email)
#Runs details through function to check validity

if valid==True: #Checks if details are valid
```

Previously in prototype 1, in the validate details function I used a global variable, valid. As it better to have as little global variables within the program, I converted the valid variable back into local and then returned valid out of the function. I didn't do this in prototype 1 as I wasn't sure on how to access the return value from a function, but I now learned that return value can be stored within a variable that call the validate function. Now allowing me to use the return value within another function. This change was only a minor one but later, but with the knowledge I used return more.

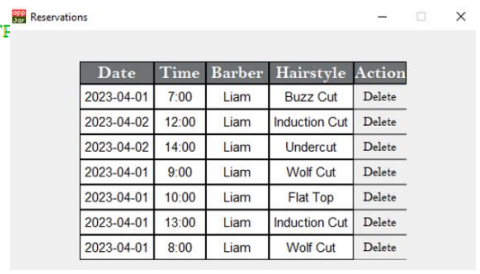
View reservation window:

```
##### VIEW RESERVATIONS #####
app.startSubWindow("Reservations") #Creates a new subwindow
app.setSize("550x450") #Sets the size of the window
app.setResizable(False) #Prevents change to size window

app.addTable("view_r",[["Date", "Time", "Barber", "Hairstyle"]], action=delete_reservation,actionButton="Delete")
#Creates tab;e that sets the headers and leaves the rows blank
app.addIconButton("home6",home,"home", align=None)
#Home button to lead to menu
app.stopSubWindow()
```

This window is displayed to the user once they click view reservation button within the main menu. I made use of AppJar tables where it will display the users' current reservations within a spreadsheet. I define the table whilst including its headers and leave the content of the table blank so I can fetch the data and insert the data later. This window also includes the home button that jumps back to the main menu. I set an action within the table that, once clicked, will run another function that will delete that reservation that the user clicked on, button also displaying the prompt, delete.

Creating the function that populates view reservation table:

```
def get_reservations():
    app.deleteAllGridRows("view_r") #Resets the table
    username=app.getEntry("inputU") #Stores username within variable
    cur.execute("SELECT reservation_date,reservation_time,barber_name,hairstyle FR
    #Fetches reservation details with the username inputted
    reservations=cur.fetchall() #Stores reservations within variable

    if not reservations:
        #Checks if varible has reservations
        app.showSubWindow("Menu") #If no reservations, output an message
        app.infoBox("No reservations present","You have no current reservations")
        #Message presented to the user
    else:
        app.addTableRows("view_r",reservations)
        #If reservations, add the list of reservations within the table
        app.showSubWindow("Reservations") #Display the reservation window
        app.hideSubWindow("Menu") #Hides the previous window, menu
```



This function is responsible for inserting all the customers reservations within the table with all the necessary information. The function starts by ensuring that the table is empty as if the user goes back to the menu and reopens the view reservation table again. It runs this function again, re-adding the same reservations within the table causing duplicates. I use an SQL statement to fetch all the reservations under the customers username and store them within the variable. If there's no reservations, it provides a message stating no reservations are under their name and display the menu once again. However, if there are reservations, it will insert the reservations into the table that I defined when creating the interface and display the view reservation window.

```
reservations=cur.fetchall() #Stores reservations within variable
app.addTableRows("view_r",reservations)
app.showSubWindow("Reservations") #Display the reservation window
```

```
self.numColumns = len(data[0])
                  ~~~~^^^
IndexError: list index out of range
```

In the original function that I created, once the program fetched all the reservations, I straight away inserted it into the table and displayed the view reservation table. This worked fine if the reservations variables contained any reservations. However, when I tested if the reservations variable had no data assigned, no reservations under that username, it would produce this error where the program tried to access an index that doesn't exist. To fix this issue, I first checked if the variable had any reservation and based on the result, either inserted the reservations or displayed an error message.

Creating the function that deletes reservations:

```
def delete_reservation(row_num):
    delete=app.getTableRow("view_r",row_num) #Stores the reservation being deleted
    result=app.questionBox("Are you sure?",f"Are you sure you want to delete the reservation at {delete[0]}?", parent="Reservations")
    #Ensures user if wants to delete reservation

    if result==True: #If they want to delete
        app.deleteGridRow("view_r",row_num) #First delete the row on the table
        cur.execute("DELETE FROM reservation_table WHERE reservation_date=? AND reservation_time=?",[delete[0],delete[1]])
        con.commit()
        #Next deletes of the reservation table
        app.infoBox("Deleted",f"Reservation on {delete[0]} has been successfuly deleted")
        #Informs user that it has been deleted
```

This function is run after the action button in the table, delete, is clicked. AppJar's action command returns the row number of the button and is passed in as a parameter that I then use to fetch the reservation that the customer wants to delete. As a precautionary measure, the program will next display a question box ,with its reservation date, to confirms if the customer wants to delete that reservation. I store the return from the question box within a variable and check if the results were either true or false. If the result of the question box was true, it will first delete the record from the table displayed to the user ,to prevent confusion why it doesn't seem to be deleted when it has deleted, and then using an SQL statement, delete the reservation from the reservation table. To ensure the correct reservations are deleted, I set the conditions as the reservation details, fetched prior. As only one reservation can have the same: reservation date, reservation time and barber. This is how the program ensures the correct reservation is deleted only. If the return to the question is false, no reservation is deleted and nothing else changes.

Preventing barber accounts from using customer side of program:

```
def valid_login ():
    user_username=app.getEntry("inputU") #stores username input as a variable
    user_password=app.getEntry("inputP") #stores password input as a variable
    employee=0

    cur.execute("SELECT password FROM user_table WHERE username=? AND is_employee=?", [user_username,employee])
    #Searches if users input is within database
    tbl_password=cur.fetchone() #Stores one result as password

    cur.execute("SELECT username FROM user_table WHERE username=? AND is_employee=?", [user_username,employee])
    tbl_username=cur.fetchone() #Stores one result as username
```

I edited the valid login function to also check if the employee field is set to false/0. The reason why I decided to make this change now is that if a barber account creates reservations, this may cause unexpected errors when using their barber username as the program uses the username inputted in a variety of different situations that may cause problem when the username of the account clashes with the barber names that are also the barber accounts username, for example Liam Smith's username is Liam, where the username also matches with the barber name fields/records in the reservation table. I would have tested this extensively to see if any problems occur, but as my time is limited, I will rather take this precautionary measure.

Creating the function that deletes all the old reservations:

```
def delete_old_reservations():
    today=date.today() #Stores the current date
    cur.execute("SELECT reservation_date FROM reservation_table")
    #Fetches all reservations currently in the table
    all_reservations=cur.fetchall()


    for each_date in all_reservations: #Checks every index
        year,month,day=each_date[0].split("-")
        #Splits date into 3. The year, month and day
        reservation_date=datetime.date(int(year),int(month),int(day))
        #Converts the string into into datetime data type
        if (reservation_date<today): #Checks if reservation is in the past
            cur.execute("DELETE FROM reservation_table WHERE reservation_date=?",[each_date[0]])
            #If in the past, delete from the table
            con.commit()
```

Before the interface function is run, this function first deletes all the reservation in the past as they are no longer needed. This also prevent the database file size from becoming unnecessarily large over time and ensures data integrity as redundant data is deleted off. The function first stores the current date using the datetime python library as this is needed later for the comparison. Using an SQL statement, fetches all reservations within the reservation table and stores within a variable. To compare the reservations, string, to the current date, datetime. They must both be in the datetime data type. The program converts the reservation date from a string to datetime by first splitting the date into the year, month, and day and changing it into an integer. The date is changed to integer as it then allows it to convert the reservation into datetime format. Now that both reservation and current date are in datetime form, the program can now check if the reservation is in the past. If in the past, using an SQL delete statement, reservations with that date in the past will be deleted off the reservation and if the reservation isn't in the past, the program will move to the next index/reservation.

```
all_reservations=cur.fetchall()
old_reservation=[]
for index_x in all_reservations:
    if index_x[0]<today:
        old_reservations.append(index_x[0])
```

```
    if index_x[0]<today:
TypeError: '<' not supported between instances of 'str' and 'datetime.date'
```

I originally didn't convert the reservations into datetime data type and instead straight away compared the reservation date against the current date. I thought this would have worked as they both had the same format, year, month, and date. But this produced the type error as I couldn't compare the string with the datetime. I initially considered splitting the variables into year, month, and date. Once the variables were split, I would use a series of nested if statements to compare each integer to check which was larger and based of that, delete old reservations. But shortly after, I learnt that it is possible to convert the reservation date into datetime and then compare. I used this approach as it is far easier than comparing each year, month, and date.

Creating the function to fetch all hairstyles:

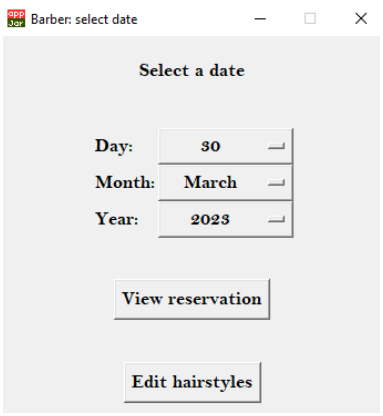
```
def fetch_hairstyle():
    cur.execute("SELECT hairstyle FROM hairstyle_table")
    con.commit()
    #Fetches all  hairstyles from hairstyles table
    hairstyle_tbl=cur.fetchall()
    #Stores hairstyles within table
    hairstyles=[] #Creates an empty list
    for each_hairstyle in hairstyle_tbl:
        #Checks through every index in reservations
        hairstyles.append(each_hairstyle[0])
        #Places hairstyles within reservation
    return hairstyles
```

```
##### SELECT HAIRSTYLE #####
app.startSubWindow("Select hairstyle") #Creates a new subwindow
app.setSize("500x350") #Sets the size of the window
app.setResizable(False) #Prevents change to size window
app.setBg("#ded8ca", override=True)
app.addImage("logo5","logo.png", compound=None)

app.addLabel("hairstyle_pick","Select a hairstyle:",)
hairstyles=fetch_hairstyle() #Runs function to get hairstyles
app.addLabelOptionBox("Hairstyles: ",hairstyles)
```

As I plan on implementing a feature that allows the employees to create/edit/delete hairstyles in the hairstyle table. In order to prepare for this feature, I can no longer hardcode the hairstyles within the option box as they need to reflect the current hairstyle table. I created a new function that fetches all the hairstyles within the table using an SQL statement. The function then stores the results within a 2D list. The function then transferred the results from a 2D list to a 1D list called hairstyles by using a counter-controlled loop, this is done as if I return the results straight from the table into the option box, it would have also included both x and y index in each option. This looked unattractive, leading me to convert it into a 1D list first causing it to look much neater as the second index doesn't exist. This function is run before any option box displaying the hairstyles is outputted ensuring the employees and customers always have the latest version of the hairstyles table to choose from.

Barber selecting a reservation date window:

```
##### BARBER SELECTS DATE #####
app.startSubWindow("Barber: select date") #Creates a new subwindow
app.setSize("350x350") #Sets the size of the window
app.setResizable(False) #Prevents change to size window
app.addLabel("qq2","Select a date")

app.addDatePicker("b_date",2,0) #Select any days of the year
app.setDatePickerRange("b_date", 2023, endYear=2025) #Limits the range to 2025
app.setDatePicker("b_date") #Set the first options as current date
app.addButton("View reservation",barber_reservation)
#Confirms and moves to view reservations
app.addButton("Edit hairstyles",edit_menu)
#Sub menu to leads to edit options
app.stopSubWindow()
```



The next series of functions and features are all related for the barber side of the program.

This window is displayed after the barber logs into their account. It outputs a date picker option box that allows them to enter a date that they want to see their reservations on. Once they click confirm, displays another window that presents all reservations for that barber on that date. The edit hairstyle button leads to another window that acts as sub menu that allows access to all the edit hairstyle features. I've decided to also make this window act as a homepage/main menu for the barber side of the program as this page does allow access to every feature available to the barber. When using the date picker command in AppJar, I set the range of years that have access as they will not need to access any further and set the first date to appear to be the current date. This allows faster access to the view reservations window, as it is most likely the barber is interested in viewing reservations on the current date, preventing the time wasting from selecting a date multiple times during their opening hours.

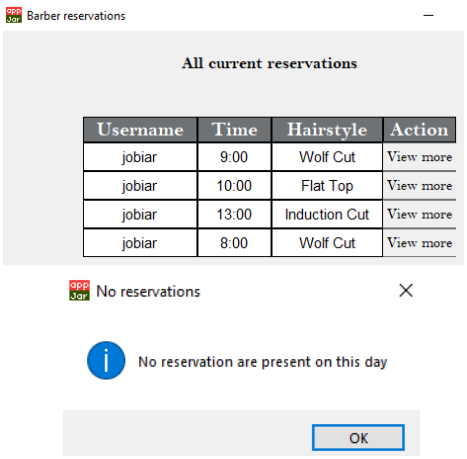
Barber view reservation window:

```
##### BARBER VIEW RESERVATION #####
app.startSubWindow("Barber reservations") #Creates a new subwindow
app.setSize("550x450") #Sets the size of the window
app.setResizable(False) #Prevents change to size window

app.addTable("B_view",[["Username","Time","Hairstyle"]], action=view_more,actionButton="View more")
#Creates table with it headers defined and records left blank
app.addIconButton("home16",b_home,"search", align=None)
#Home button
app.stopSubWindow()
```

This window displays all the reservation the barber has on the date that they selected. I originally planned on implementing a calendar system, but due restrictions such as time and AppJar GUI options. I decided to instead use AppJar's inbuilt tables as this will suffice as a suitable replacement. This approach was much easier than find an away to implement a calendar system with days of the month/weeks being outputted. This approach does not sacrifice any features and information, it provides the same functionality but just another approach. I asked my client for feedback on this approach. He said that as it provides the same functionality whist still being easy to use, he doesn't mind this change. One benefit this approach has over the calendar system, is that it will be familiar with the barbers already as previously, they also used a table to record their reservations. This interface is identical to the customer view reservation with the only difference being that the tables action button runs a function to fetch all additional information regarding the reservation.

Creating the function that populates the barbers' reservations:

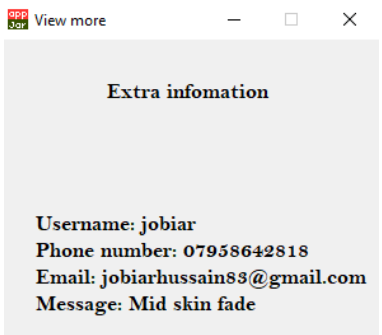
```
def barber_get_reservation():
    app.deleteAllGridRows("B_view") #Resets the table
    barber=app.getEntry("barberU") #Stores barber name within variable
    b_date=app.getDatePicker("b_date")
    cur.execute("SELECT username,reservation_time,hairstyle FROM reservation_table
                "WHERE barber_name=? and reservation_date=?",[barber,b_date])
    #Fetches reservation details with the barber name and date inputted
    b_reservations=cur.fetchall() #Stores reservations within variable

    if not b_reservations: #Checks if varible has reservations
        app.hideSubWindow("Barber reservations")
        app.showSubWindow("Barber: select date")
        app.infoBox("No reservations","No reservation are present on this day")
        #Message presented to the user
    else:
        #If reservations, add the list of reservations within the table
        app.addTableRows("B_view",b_reservations)
        app.showSubWindow("Barber reservations")
        #Displays the barber reservations page
```



This function is responsible for inserting the reservations into the table. Its firsts reset the table to prevent duplicate data when jumping back and forth to this window. It fetches and stores all the data required to find all reservations under that barber name and date selected. It assigns the reservation to a variable that is then checked if any data is present. If there are any reservations, it will update the reservation table. If there are no reservations it will present a message that alerts the barber and prevent it from inserting the data within the table as this would produce an index error.

View more information window:

```
##### BARBER VIEW MORE INFO #####
app.startSubWindow("View more")
app.setSize("300x250") #Sets the size of the window
app.setResizable(False) #Prevents change to size window
app.addLabel("label8","Extra infomation")

app.addLabel("extra info",[],1,0)
#Sets a blank label to set later
app.stopSubWindow()
```



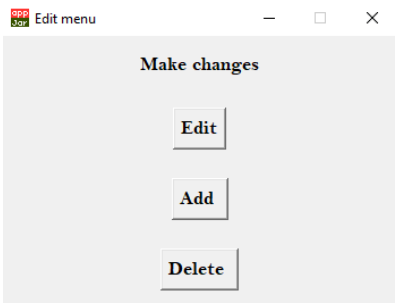
This window is displayed to the barber when they click on the view more button on the table with the reservations. This displays all information regarding the customer that specific reservation. It consists of the username and their contact information. If they did input a message for the barber, this message will be available for the barber on this window. The label that I defined within the interface function is left blank. I've left this blank as before this window is displayed, the program will run a function that will fetch all the details required and then updates the label before it is displayed.

Creating the function that retrieves all the relevant information for view more:

```
def view_more(row_numl):
    view_this=app.getTableRow("B_view",row_numl) #Assigns the record selected to a variable
    b_date=app.getDatePicker("b_date") #Assigns the dates to a variable
    barber_username=app.getEntry("barberU") #Stores username input within a variable
    cur.execute("SELECT username,phone_number,email FROM user_table WHERE username=?",[view_this[0]])
    #Fetches all relevent customer details
    user_details=cur.fetchone() #Stores the details within a variable
    cur.execute("SELECT message FROM reservation_table WHERE reservation_date=? AND reservation_time=? AND barber_name=?",[b_date,view_this[1],barber_username])
    #Fetches the message that the user may have inputted
    message=cur.fetchone() #Stores the message within a variable
    con.commit()

    app.setLabel("extra info",f"""
    Username: {user_details[0]}
    Phone number: {user_details[1]}
    Email: {user_details[2]}
    Message: {message[0]}""")
    #Sets the label with all the inforamtion stored within the variables
    app.showSubWindow("View more")
```

This function is responsible for gathering all the data that will be presented to the barber and setting the blank label with the information. The action button on the table passes into this function the row number. With that row number I retrieve the reservation linked with the row number and store the date the barber selected on the select a date window. Using an SQL statement, I gather all customer information using the username that's present within variable containing the reservation. The function then gathers any message the user may have inputted using the reservation time, date and barber. As no two reservations can share the same time/date/barber, this ensure the correct message is fetched. The function then sets the blank label with all the information retrieved prior.

Edit hairstyles sub menu window:

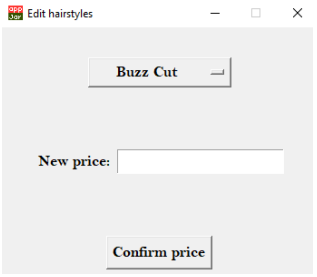



```
##### BARBER EDIT MENU #####
app.startSubWindow("Edit menu")
app.setSize("350x300") #Sets the size of the window
app.setResizable(False) #Prevents change to size window

app.addLabel("label3","Make changes")
app.addButton("Edit",edit_hairstyle)#Leads to change price
app.addButton("Add ",new_hairstyle) #Leads to create new hairstyle
app.addButton("Delete ",delete_hairstyle) #Leads to delete hairstyle
app.stopSubWindow()
```

This window is displayed after the barber clicks on the edit hairstyles button on the select date window. This window is a sub menu that displays all the option the barber has, to make certain changes to the hairstyle table. These series of function allow the hairstyle table/barbershop to remain relevant and prevent redundancy as they will be able to adapt to latest trends and prices. This window is again simple with only 3 buttons that all lead to their own functions that carry out the task.

I've considered instead of separating all the different options, I could display a table that allows the barbers to make changes within the table. This would allow them to edit all the records/cells such as the names and the hairstyle attributes. This would have been a much more intuitive approach, but the problems were: not knowing how to allow them to make changes to individual cells within table, me having to validate every input the barber made. For example, ensuring the prices where in a correct format or ensuring they matched the hairstyle attribute name completely as any mistake, such as a letter in the wrong case, would break the recommend hairstyle feature as it relies on the fact that every attribute , such as non/little, are spelled and types the same. This would have required much more development time/research, time that was running low. This is why I decided to go with the simpler approach of separating the features and avoiding the use of tables.

Edit prices option window:




```
##### BARBER EDIT HAIRSTYLES #####
app.startSubWindow("Edit hairstyles")
app.setSize("350x300") #Sets the size of the window
app.setResizable(False) #Prevents change to size window
hairstyles=fetch_hairstyle() #Retrieves all hairstyles
app.addOptionBox("Edit: ",hairstyles)
#Displays updates hairstyle options
app.addLabelEntry("New price: ") #Input new price
app.addButton("Confirm price",new_price)
#Buttons confirms the edit
app.stopSubWindow()
```

This window is displayed after the barber clicks edit button on the edit menu. This window allows them to change the prices of a hairstyle only as the other fields usually remain constant. Instead of an entry box to input the name of the hairstyle, I used an option as it prevents any validation of the name and allows the barber to view the latest version of hairstyles within the table, due to the fetch hairstyle function. There's an input box that allows the barber to set a price, then a button that allows them to a chance to check and hit confirm,

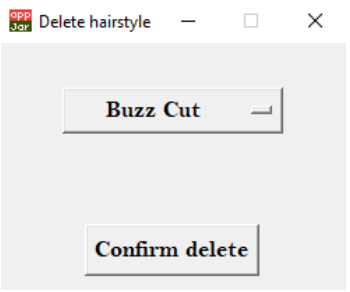
Creating function that updates hairstyle record with new prices:

```
def new_price():
    edit_hairstyle=app.getOptionBox("Edit: ") #Retrievs hairstyle name input
    new_price=app.getEntry("New price: ") #Retrievs new price input

    if new_price.replace(".", "").strip().isnumeric()==True: #Validates if input is all integers
        cur.execute("UPDATE hairstyle_table SET price=? WHERE hairstyle=?",[new_price,edit_hairstyle])
        con.commit()
        #If input valid, inserts into the table
        app.infoBox("Successfully updates","New prices have been set")
    else:
        app.errorBox("Format error","Price in incorrect format")
        #If input invalid, error message
```

This function is responsible for validating the new price input and based on validation, either updates the prices within the hairstyle table or give an error. This function is run after the barber clicks confirm price button on the edit price window. The function first retrieves all the details the barber inputted in edit price window and stores within a variable. As a price cannot have letter/most symbols, the function firsts validate the price input. It first removes any accidental whitespace using strip command and replaces the decimal point with replace command. Now that the numbers in the price are isolated, the function can now use is numeric command to check if all the characters are numbers. The reason why I checked if there are all numbers last is because if any whitespace or decimal points where present, this would give a false to the is numeric check even if all numbers where present. If the validation passed, old price is updated using a SQL update statement and if validation failed, an error message is displayed instead.

Delete hairstyle window:

```
##### BARBER DELETE HAIRSTYLE #####
app.startSubWindow("Delete hairstyle")
app.setSize("400x350") #Sets the size of the window
app.setResizable(False) #Prevents change to size window

hairstyles=fetch_hairstyle() #Retrieves all hairstyles
app.addOptionBox("Delete: ",hairstyles)
#Displays updated hairstyle options
app.addButton("Confirm delete",confirm_delete_hairstyle)
#Button confirms choice
app.stopSubWindow()
```



This window is displayed after the barber clicks the delete button on the edit hairstyle menu window. This page outputs an option box that allows barber to select a hairstyle from the updated hairstyle table and a button that runs a function to delete the hairstyle selected.

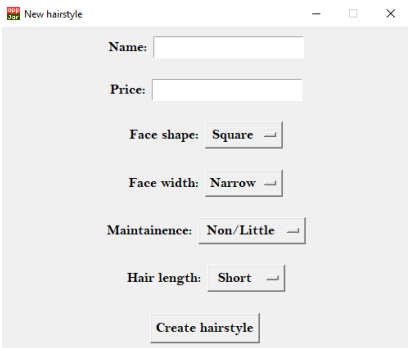
Creating the function that deletes the hairstyle:

```
def confirm_delete_hairstyle():
    delete_hairstyle=app.getOptionBox("Delete: ") #Retrives hairstyle being deleted
    result=app.questionBox("Delte hairstyle?",f"Are you sure you want to delete the hairstyle, {delete_hairstyle}?", parent="Delete hairstyle")
    #Confirms if barber wants to delete hairstyles
    if result==True: #Checks if they confirm or not
        cur.execute("DELETE FROM hairstyle_table WHERE hairstyle=?",[delete_hairstyle])
        #If confirmed, deletes hairstyle from hairstyle table
        con.commit()
        app.infoBox("Successfully deleted",f"Hairstyle, {delete_hairstyle} has been deleted")
        #Message that informs barber
```

This function is run after the barber clicks the delete button on the delete hairstyle window. It first stores the input of the hairstyle the barber wants to delete within a variable. By using AppJar's question box, it displays an option for the barber to confirm if this is hairstyle is what they want to delete. If they do confirm, the function runs an SQL delete statement that deletes the hairstyle from the hairstyle table, if they do not confirm, the hairstyle isn't deleted.

Create hairstyle window:

```
##### BARBER NEW HAIRSTYLE #####
app.startSubWindow("New hairstyle")
app.setSize("500x400") #Sets the size of the window
app.setResizable(False) #Prevents change to size window

app.addLabelEntry("Name: ") #Input box for name
app.addLabelEntry("Price: ") #Input box for price
app.addLabelOptionBox("Face shape: ",["Square","Round"])
app.addLabelOptionBox("Face width: ",["Narrow","Wide"])
app.addLabelOptionBox("Maintainence: ",["Non/Little","Medium","Lot"])
app.addLabelOptionBox("Hair length: ",["Short","Medium","Long"])
#Input options for hairstyle attributes
app.addButton("Create hairstyle",insert_hairstyle)
#Button that creates hairstyle
app.stopSubWindow()
```



This window is displayed after the barber clicks the add button on the edit menu window. This window allows barber to input the details for a new hairstyle that can be inserted into the table. It consists of entry boxes for the name and price, where the price is going to validated. Consists of option boxes for the hairstyle characteristics. Utilised option boxes as this prevents validation of each hairstyle characteristic if I used entry boxes. A confirm button that runs a function that validates the details and then inserts the new hairstyle into the hairstyle table.

Creating the function that inserts the new hairstyle:

```
def insert_hairstyle():
    name=app.getEntry("Name: ") #Stores name input in varible
    price=app.getEntry("Price: ").strip() #Stoes price input in variable
    shape=app.getOptionBox("Face shape: ") #Stores face shape input
    width=app.getOptionBox("Face width: ") #Stores face width input
    maintainence=app.getOptionBox("Maintainence: ") #Stores hair maintainence input
    length=app.getOptionBox("Hair length: ") #Stores length input as variable

    if price.replace(".", "").isnumeric()==True: #Checks if price valid input
        cur.execute("INSERT INTO hairstyle_table (hairstyle,price,face_shape,face_width,maintainence,hair_length)VALUES
        con.commit()
        #If valid, inserts into the table
        app.infoBox("Successfully updated","New hairstyle has been added")
        #Message box, informing barber
    else:
        #If invalid, error message
        app.errorBox("Incorrect format 2","Price is in incorrect format")
```

This function is responsible for inserting the new hairstyle within the hairstyle input and is run after they click the create hairstyle button in the create hairstyle window. The function first retrieves all the inputs the barber made within the create hairstyle window and stores them within variables. Next it runs the same validation on the price that I've done in new price function. If the price is valid, using an SQL insert statement, the details are inserted into the hairstyle table within a new record. If the price is an invalid input, it produces an error message that prevents the inserting of the new hairstyle.

Creating the barber side home button:

```
def b_home():
    app.deleteAllGridRows("B_view") #Resets tables
    app.hideAllSubWindows(useStopFunction=False)
    #Hides all subwindows
    app.showSubWindow("Barber: select date")
    #Back to select date window
app.addIconButton("home15",b_home,"search", align=None)
#Home button to select date
```

As the select date window acts as the main menu for the barber side of the program. I added an icon button on every window past the select date window that once clicked, runs the function. This allows the barber to go back and have access to all barber features again. The function first resets the barber reservation table to prevent duplicate rows and then closes all sub windows, after displays the select date window again.

Prototype 2: Evaluation

No	Criteria	Have I met?	Improvements for later prototypes
1	Account/login		No improvements needed as it fully functional
2	Design		Focus on this in prototype 3
3	Create reservations		No improvements needed as its fully functional
4	Database		No improvements needed as its fully functional
5	Recommend hairstyles		Add more hairstyles
6	Edit hairstyles		Allow edits to hairstyle attributes
7	Hardware compatible		No improvements, compatible with lower end computers
8	Prevent double booking		No improvements as it fully functional
9	Menu/home buttons		No improvements needed as its fully functional
10	Cancel reservation		No improvements needed as its fully functional

Feedback from client:

Speaking with my client, he was very pleased that most of the functionality that he requested was implemented successfully. However, there were some improvements he would have like to be made.

One piece of feedback was that when my client was testing out the recommended hairstyle feature, he said he would like to reduce the number of unsuccessful searches whilst also maintaining the accuracy. To improve on this, I suggested to Liam that he should search for more hairstyles that I can later input into the hairstyles table. This then should reduce the number of null outputs as there are now more possibilities.

He also stated that the design of the program was not ready, then I reminded him that this was the focus for the next prototype.

Another piece of feedback was that when view reservations on the table, my client wanted them to be presented in chronological order. As this may require a lot more time to develop, I said that I will try to implement this feature but may not be able to within the time frame.

The last minor that my client requested was when editing a hairstyle, instead of only being able to edit the price, he wanted to also be able to changes its characteristics as well. He said that they should be able to edit them as trends mostly dictate what hairstyle suit what type of people head.

Prototype 3: Design/final touches.

This prototype is going to be relatively short compared to the previous prototypes as majority of the features have already been implemented. These are the features I intend to add for the third and final prototype:

-Set background colour

-Addition of barbershop logo

-User interface icon overhaul

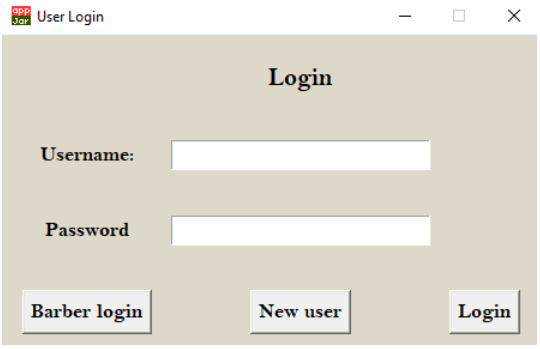
-Setting font

-Adding more hairstyles into hairstyle table

-Modifying edit price to also include edit hairstyle attributes

Setting the background colour:

```
##### LOGIN PAGE #####

app.setSize("450x250")#Sets size of window
app.setResizable(False) #Prevents window size change

app.addLabel("Login title","Login",1,1)
app.setBg("#ded8ca", override=True)
#Sets the backround colour
```



Now that the programs functionality and features are finished, now the focus has shifted on the design. I first changed the background colour from AppJar's default grey colour to a more earthy cream colour. As my client requested this tone as it matched the interior design of his barber shop, he wanted his program to also match. I used AppJar's command setBg, to set the background colour for all sub windows within the program, further providing a uniform experience. I originally wanted to do a two-tone background like in my final design, but as there wasn't a suitable method, I decided it was best to keep the background as 1 colour. To have a two-tone colour scheme, I attempted to set a background image that was the same within the final design. This however came with many problems, the main problem being that widget within the interface were not cut out corrected, causing the widgets own background colour sticking out and not matching with the image behind. The other problem I faced with this was that every image had to the exact same size as the window, this caused the image to not fit properly within the windows causing distortions. These factors lead to the image approach as the background being unattractive.

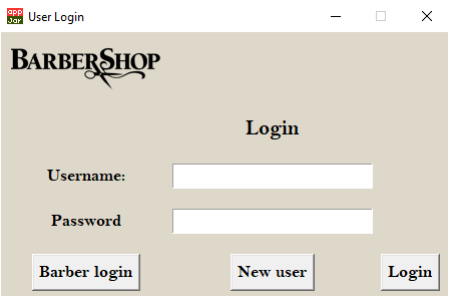
Setting the user interface font:

```
##### LOGIN PAGE #####

app.setSize("450x250")#Sets size of window
app.setResizable(False) #Prevents window size change
app.addImage("logoll","logo.png", compound=None)
app.addLabel("Login title","Login",1,1)
app.setFont(size=13, family="Bell MT", weight="bold")
#Sets the program font/size/weight
app.setBg("#ded8ca", override=True)
#Sets the backround colour
app.getLabelWidget("Login title").config(font=("Bell MT", "16", "bold"))
#Configurates this widget only
```

At the beginning of the function that creates the interface, I defined the program font that sets the font for the whole program, using AppJar's command set font. Using this command, I set the font to be the same as my final design as my client preferred this font over others. To make the text easier to read without increasing the font size too much, I set the font to be also in bold throughout the whole program. This further helps the program to become more accessible as it is now easier to see smaller text. I used AppJar's configurate command to change only some labels within my program. I further enlarged headers and more important information throughout the program to draw more attention first to the most important parts of the program, further increasing readability.

Setting the barbershop logo:

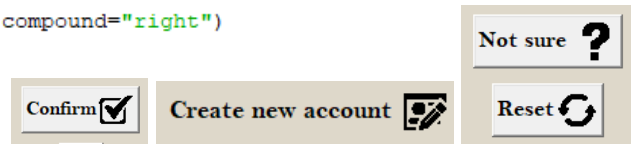
```
##### LOGIN PAGE #####

app.setSize("450x250")#Sets size of window
app.setResizable(False) #Prevents window size change
app.addImage("logoll","logo.png", compound=None)
#Adds the barbershop logo
```



As per my client's request, I added the barbershop logo on every window within the program. This helps each page instantly recognizable as my client's barbershop. Once my client sent the transparent image of their logo, I took the image within photoshop and matched the background colour of the image to the programs, by using the same hex code. Once the image was created, I first saved it as a GIF file and added it within the program. The problem I faced with this was that this caused the image to be in low resolution, to fix this, I used the PNG file type, and this added a much higher resolution image to the program. As all the windows are similar in size, this allowed me to use the same size image of the logo without the logo being either too small or too large.

User interface icon overhaul:

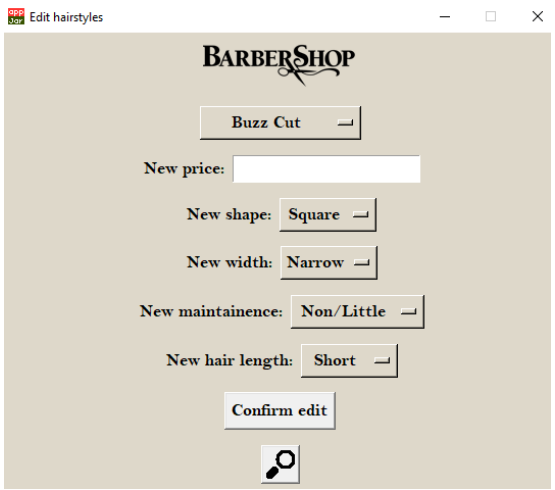
```
app.addIconButton("Create reservation",select_barber,"book-alt-2", align="top")
#New Reservation button
app.addIconButton("View reservations",get_reservations,"view", align="top")
#View reservation button
app.addIcon("Barber login ","cut", compound="right")
```



Throughout the whole program, I implemented icons and icons button on most windows. I made use of AppJar's icon command that allows labels to include icons, as well as buttons. To source these icons, within the AppJar files, there's a folder that contain all icons native to AppJar. For each icon I added, I must first find suitable icons and with the file name, add the name of the file within the lines of code. The reasons that I added many icons throughout the program was to further increase accessibility. This allows users to easily navigate the program without the need of reading text, this especially will help my clients customers, whose first language isn't English, to easily navigate the program as the icons visually describe the purpose. During evaluation, all icons within the program will be presented.

Modifying edit hairstyle function:

```
##### BARBER EDIT HAIRSTYLES #####
app.startSubWindow("Edit hairstyles")
app.setSize("550x450") #Sets the size of the window
app.setResizable(False) #Prevents change to size window
app.setBg("#ded8ca", override=True)
app.addImage("logo18","logo.png", compound=None)
hairstyles=fetch_hairstyle() #Retrieves all hairstyles
app.addOptionBox("Edit: ",hairstyles)
#Displays updated hairstyle options
app.addLabelEntry("New price: ") #Input new price
app.addLabelOptionBox("New shape: ",["Square","Round"])
app.addLabelOptionBox("New width: ",["Narrow","Wide"])
app.addLabelOptionBox("New maintainence: ",["Non/Little","Medium","Lot"])
app.addLabelOptionBox("New hair length: ",["Short","Medium","Long"])
#Input options for hairstyle attributes
app.addButton("Confirm edit",edit_details)
#Buttons confirms the edit
def edit_details():
    edit_hairstyle=app.getOptionBox("Edit: ") #Retrievs hairstyle name input
    new_price=app.getEntry("New price: ").strip() #Retrievs new price input
    shape=app.getOptionBox("New shape: ") #Stores face shape input
    width=app.getOptionBox("New width: ") #Stores face width input
    maintainence=app.getOptionBox("New maintainence: ") #Stores hair maintainence input
    length=app.getOptionBox("New hair length: ") #Stores length input as variable

    if new_price.replace(".", "").isnumeric()==True: #Validates if input is all integers
        cur.execute("UPDATE hairstyle_table SET price=?,face_shape=?,face_width=?,maintainence=?,hair_length=? WHERE hairstyle=?",
        con.commit()
        #If input valid, inserts into the table
        app.infoBox("Successfully updates","New prices have been set")
    else:
        app.errorBox("Format error","Price in incorrect format")
        #If input invalid, error message
```



From the prototype two feedback, my client wanted to allow the barbers to also make changes to the attributes of each hairstyle instead of only being able to edit the price. This required minor adjustment, I had to also include in the window option boxes now that contained the options for each attribute. Once they edited everything and click confirm edit, the program will run the edit detail's function that will now also retrieve all the option box inputs and store them within their own variables. Now when updating the hairstyle record, the function now updates the attributes along with the price.

Increasing number of hairstyle options:

From the feedback from prototype two, my client wanted to reduce the number of unsuccessful matches with the generate hairstyle feature. I asked my cline to send a new list of unique hairstyles along with their prices and other attributes to increase the number combinations possible.

With this new list, I created new hairstyles within the program using all the details the client provided. Now that there are over 50 unique hairstyles within the table, it is now unlikely that an unsuccessful result will come from the recommended hairstyle feature. After testing the feature, I rarely came across a null result.

35	37	Undercut ...	35.0	Square	Wide	Medium	Long
36	38	Flat top	20.0	Round	Narrow	Lot	Long
37	39	Flow	27.0	Square	Wide	Medium	Medium
38	40	High and ...	12.0	Square	Wide	Non/Little	Medium
39	41	Curtain ...	25.0	Square	Wide	Lot	Medium
40	42	Scissor crop	30.0	Round	Narrow	Lot	Medium
41	43	Top knot	20.0	Round	Narrow	Non/Little	Medium
42	44	Burst	15.0	Round	Narrow	Lot	Medium
43	45	Mop top	20.0	Round	Wide	Non/Little	Long
44	46	Elephant ...	30.0	Round	Wide	Medium	Long
45	47	Brush up	17.0	Square	Narrow	Medium	Medium
46	48	Tapered ...	40.0	Round	Narrow	Medium	Medium
47	49	Longer buz...	40.0	Round	Narrow	Non/Little	Short
48	50	Swept ...	23.0	Round	Narrow	Non/Little	Short
49	51	Irregular top	25.0	Round	Wide	Medium	Long
50	52	Stir fired top	30.0	Round	Wide	Lot	Short
51	53	Angular ...	25.0	Round	Narrow	Lot	Short

Prototype 3: Evaluation

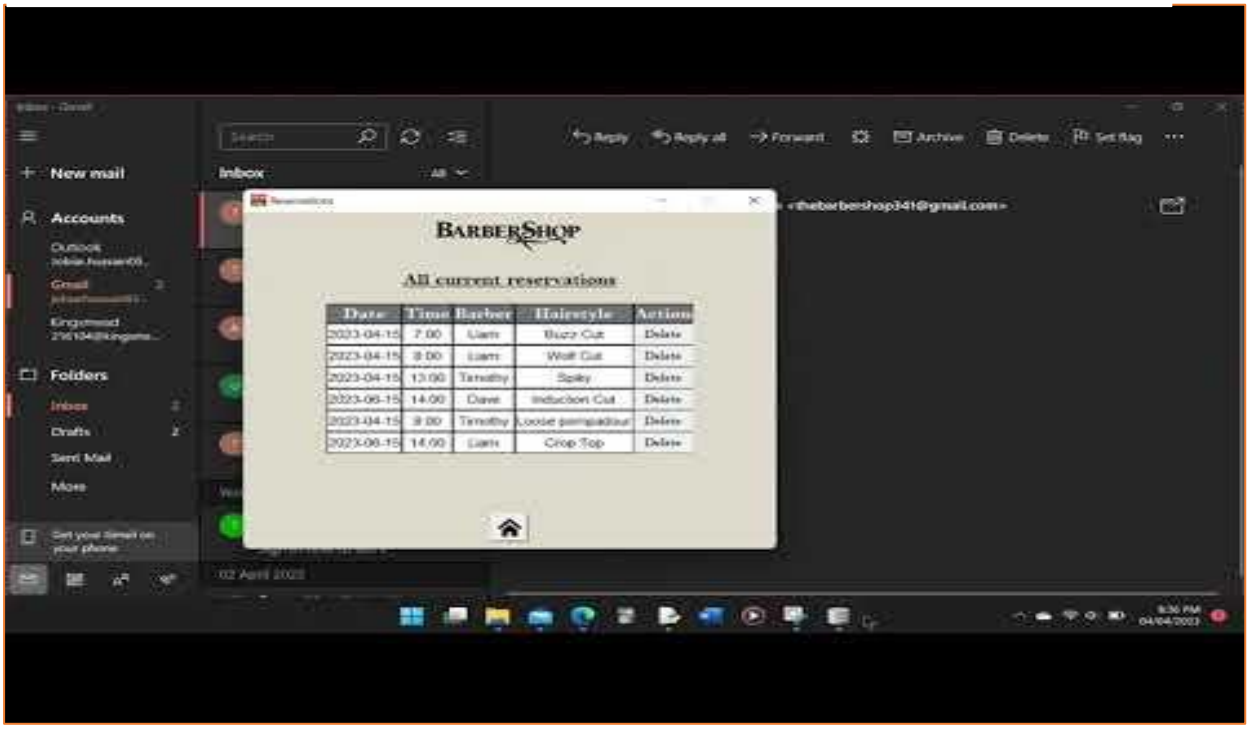
No	Criteria	Have I met?	Improvements for later prototypes
1	Account/login		No improvements needed as its fully functional
2	Design		No improvements needed
3	Create reservations		No improvements needed as its fully functional
4	Database		No improvements needed as its fully functional
5	Recommend hairstyles		No improvements needed as its fully functional
6	Edit hairstyles		No improvements needed as its fully functional
7	Hardware compatible		No improvements, compatible with lower end computers
8	Prevent double booking		No improvements needed as its fully functional
9	Menu/home buttons		No improvements needed as its fully functional
10	Cancel reservation		No improvements needed as its fully functional

Feedback from client:

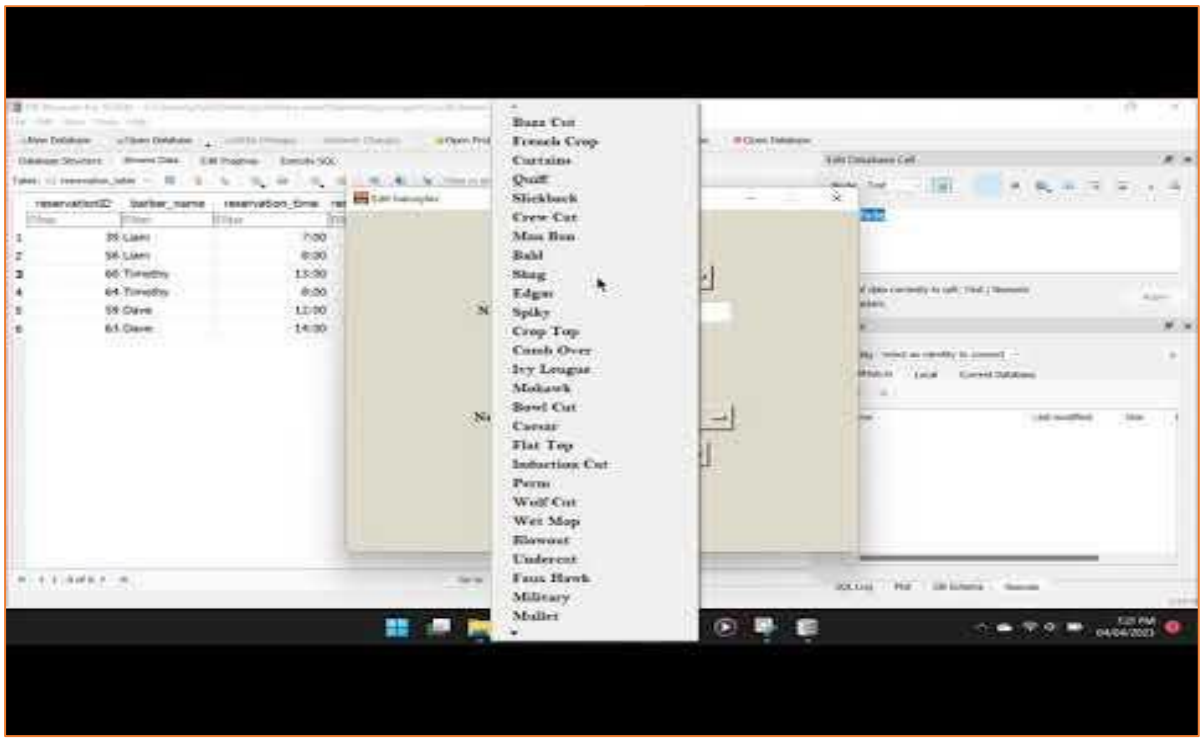
Speaking with my client, he was very pleased with the end result. Compared to the initial success criteria, I have met all initial requirements. As I received no further feedback from my client, I will move onto evaluate the program as a whole.

Evaluation:

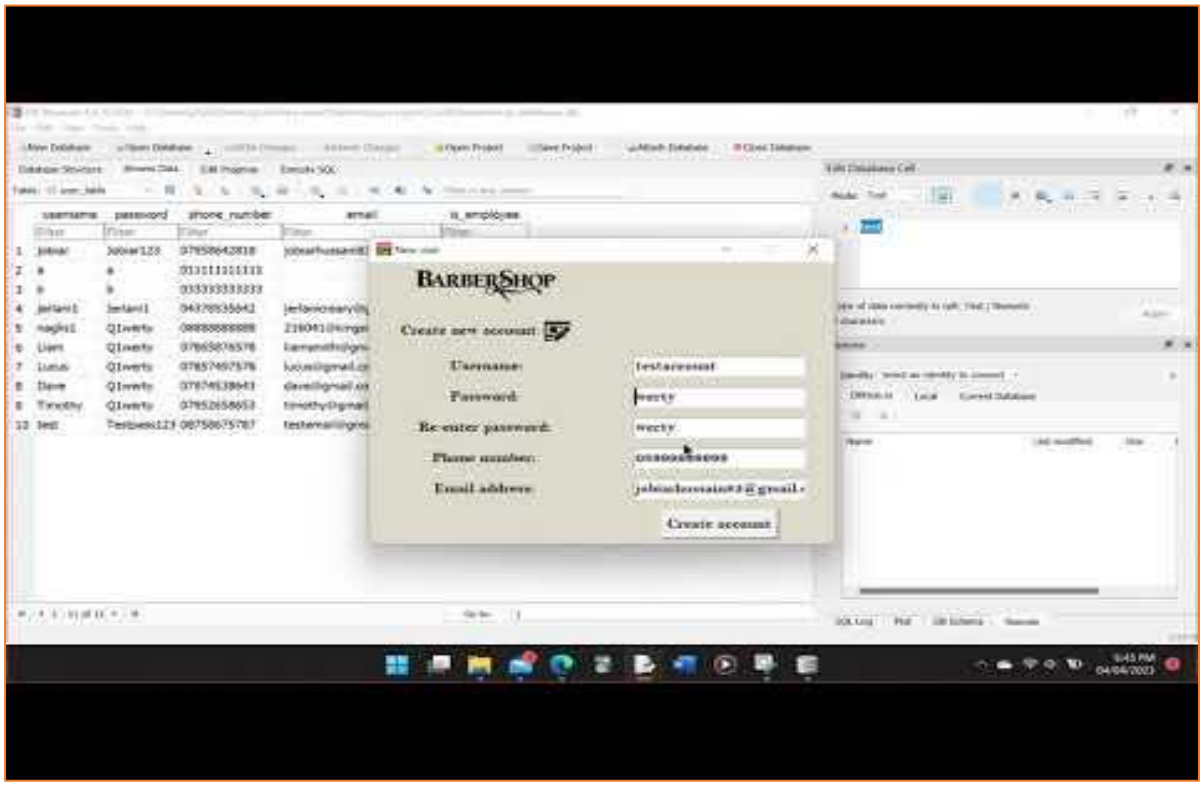
Video 1: Customer side testing



Video 2: Barber side testing



Video 3: Create account testing.



Test	What is the test	Test data/Time stamp	Outcome	
Fit design language	Test against documents and interior design of the barbershop to see if there is a coherent design language. Compare the colour shapes and feel	**-See throughout all videos**	The design came out as expected. But some design choices my client wanted, two-tone colour, were not implemented. The client also wanted rounded boxes as well as boxes around the area of interactions but again, was not implemented	
Run on variety of hardware	I will run my program first on my clients' laptops within the barbershop to see if the employees can access their calendars. To test this further I will run it on older, less computationally powerful hardware to test the lower end to ensure it's compatible with a variety of devices		I tested this on my clients' computers in the barbershop and the program worked with no issues. I've also tested the program on a lower-end laptop and again came across no issue, this test confirmed that this program is compatible with a variety of machines.	
Have high-quality images for icons	As I use the program, I will inspect every image/icon to ensure they are at a high quality	**-See throughout all videos**	Me and my client have inspected the image quality of the logos and icons and are happy with their resolution, they can see clearly at different angles and distances	
Maintain data integrity	Check the data within hairstyle table match with barbershops service With correct pricing	**-Video 2 (3:00+)**	I've compared the list of services with my client's list and came across no errors, each service and its details are all listed correctly	
Enforce user access levels	Test the login system by trying to login into both employee/customers logins with invalid details like "j@("(ss"	**-Video 1 (0:25-2:25)** -"j@("(ss" -jobiar -jobia -No input -Liam (barber account) -Wrong authentication code	I attempted to login into the customer side of the program with the barber account details and vice versa, the program successfully prevented access. I also attempted to log in with invalid details and without fail, prevented access to all invalid inputs and no inputs as well.	
Usernames validate	Test if a new account is created with usernames, not within 0 to 15 characters. Test also with no input.	**-Video 3 (0:00-2:40)** -jobiar (already exists) -jobiar1234566 -testaccount	I attempted to an account with the same username that already exists, but the program successfully prevented the account from being created. Attempting to create an account with empty username input was rejected with data also outside its expected range of characters.	
Passwords validate	Test if the program allows access with passwords without capital letters and numbers like "jobiar". Test invalid passwords by both creating a new user and login in. Test also with no input	**-Video 1 (0:25-2:25)** -No input -JOBIAR123 -Jobiar123 -Q1werty (barber account)	I attempted to log in with the correct username and a variety of wrong passwords. These all prevented access to the system. I also attempted to access the customer side with a barber account password, and this also was rejected by the program.	
Re-enter password match	Test if a new account is created if I enter a password that doesn't match the password made before. Such as "Q1werty" and "Q1wert"	**-Video 3 (0:00-2:40)** -Q1werty -Q1wert -werty -Qwerty -Q2werty	If the password was valid, with every invalid input I put in for the re-entered password. The program successfully rejected the input as the passwords didn't match	
Test the login/create a new user button functions properly	I will test this by clicking the button and seeing if it directs the user to the correct page or not	**-Video 1 (1:35-2:25)**	After clicking the button, the program successfully displayed the correct screen	
Create a new user successfully	I will create a variety of users to see whether the program creates an	**-Video 3 (2:35+)**	I inputted the same details of an already existing account, if I tried to create the account	

	account with the correct information. Will also try to create identical accounts with the same details to see if the program accepts it	-jobiar (username) -Q1werty (password) -testaccount (username)	the program rejected the username input as it isn't unique. I created a new account with a new username and within the database, successfully stored the correct details
Menu page that allows user access different parts	I will test if the buttons on the menu page led to the correct part of the function.	**-Video 1 (2:24-2:28)** **-Video 1 (4:30-4:35)**	I clicked on the buttons, and it successfully displayed the correct windows
Reservation successfully save on the database	I will create multiple reservations whilst also trying different options to ensure everything functions correctly	**-Video 1 (2:25-5:00)** **-Video 1 (5:00-6:10)**	I created multiple reservations, and they all store correctly within the database and are displayed within the view reservations windows
Displays all relevant information in employees' calendar	I will create multiple test reservations and through employee login, check if it correctly displays reservations for that employee only	**-Video 2 (0:45-1:05)** **-Video 2 (2:00-2:25)**	I created multiple reservations with different barbers and then logged into the barber accounts. I tested if the reservations with that barber only were outputted, this worked as intended
Does the program prevent double booking	Test if I can create a reservation on the same day with the same barber at the same time. Such as creating 2 reservations with Liam on the 15th of June at 14:00	**-Video 1 (5:00-5:45)** -15th June at 14:00 -15th June at 15:00	I created a booking for that date and another booking with the same details. I attempted to book at the same time, but the program prevents the same timeslot being picked, but not for other barbers
View reservation	I will create reservations with the same account and then see if the view reservation page shows reservations that have been created	**-Video 1 (4:10-5:00)**	I created multiple reservations under one account and when viewing them, only the reservations under the username's name are shown successfully.
Test if the reservation can be cancelled	I will create a variety of reservations from the same account and delete some reservations, check the table itself and see if the picked reservations have been deleted only	**-Video 1 (6:15-6:50)**	I attempted to delete some reservations and after confirming, check the database to see that it successfully deleted the correct reservations and within the view reservation window
The program displays the correct dates of the current week during creating reservations	See if the program displays all the days that should be available	**-Video 1 (3:20-3:38)** **-Video 2 (0:45-0:50)**	I tested the option box to confirm that it does display all available dates possible
Recommend hairstyle produces logically correct results	I will run the function multiple times and check if the responses make sense and are logically correct	**-Video 1 (2:35-3:20)**	I ran the feature multiple times to see that it gave answers that made sense based on the inputs I made. However, some combinations of inputs are still producing a null result.
The home button leads to the menu page	Click the button on all pages to see if it directs me to the home page	**-Video 1 (6:50-7:18)** **-Throughout all of video 2**	I clicked the home button on every window, and it does go back to the menu window every time
The confirmation screen displays all the correct inputs from the user	I will create reservations and see if the confirmation displays the relevant information regarding the reservation	**-Video 1 (2:25-4:12)** **-Video 1 (5:25-5:45)**	Before creating multiple reservations, I checked if the confirm reservation window displayed the correct information. With every reservation made, the correct details were outputted

Update/ reschedule reservations	Will create multiple reservations on the same account and edit some reservations with different dates, then check the database to see if the correct reservation were affected only		I don't have an edit reservation feature. This was unsuccessful.	
Allow employees into the employee calendar whilst denying users in	Will attempt to log in to the employee login with the user's login details to if they work, test if the invalid account "jobiar" can log in and password "Jobiar123"	**-Video 2 (0:00-0:20)** **-Video 1 (1:05-1:15)** -jobiar (username) -Jobiar123 (password)	I attempted to log in to the barber side of the program with a customer account and every time, the program rejects my access	
Method to scroll through lists	Will attempt to scroll/flip through a list so everything can be seen on a single page	**-Video 1 (2:25-2:37)**	The scroll bar on the option boxes successfully allowed access to all available options	
Updating hairstyle table	Attempt to update a price within the hairstyle table. I will change the buzzcut price from £15 to £20.5, and insert a new hairstyle with its attributes	**-Video 2 (2:57+)** -20ss -£20 -20,00 -20.50	I modified hairstyle prices and when confirming the changes, it successfully changed the details within the database. This also successfully updated the table when changing hairstyles attributes	

Comments:

From my testing, I can conclude that the system is majorly robust and is very well validated. The test that I failed was the test if I could edit reservations that were already made but my program lacks this feature. I could have improved further by making the information/error boxes clearer and more understandable. Helping communicate to the user what the problem was. Apart from that, the testing was a success.

White box testing

In this test, I will be testing majority of variables that is defined/used within each main function. The point of this test to see if the variables are storing the correct information. This test will consist of me using print statements to be able to visually see each variable.

Validate login details:

```
def valid_login ():
    user_username=app.getEntry("inputU") #stores username input as a variable
    user_password=app.getEntry("inputP") #stores password input as a variable
    employee=0

    cur.execute("SELECT password FROM user_table WHERE username=? AND is_emplo
    #Searches if users input is within database
    tbl_password=cur.fetchone() #Stores one result as password
    print(tbl_password)
    print(user_username)
    print(user_password)

    cur.execute("SELECT username FROM user_table WHERE username=? AND is_emplo
    tbl_username=cur.fetchone() #Stores one result as username
    print(user_username)
```

```
('Jobiar123',)
jobiar
Jobiar123
jobiar
```

All correct data was stored, matched against input.

Create account:

```
def create_account():
    new_Uname=app.getEntry("new_username").strip() #Stores username inp
    new_Pword=app.getEntry("new_password") #Stores password input as a
    re_enter_pword=app.getEntry("new_password2") #Stores re enter passw
    new_Pnumber=app.getEntry("new_phone").replace(" ", "").strip() #Stc
    new_email=app.getEntry("user_email")

    print(new_Uname,new_Pword,re_enter_pword,new_email,new_email)

    employee=0 #Creates a new variable to set is_employee field as 0
    valid=validate_details(new_Uname,new_Pword,new_Pnumber,re_enter_pwc
    #Runs details through function to check validity
    print(valid)
```

```
james
James124
James124
087856379657
james@gmail.com
False
```

All correct data was stored, matched against input. Phone number was incorrect, lead to false valid.

Filter unavailable times:

```
def get_times():

    d_pick=app.getDatePicker("date")  #Stores customers date input
    b_pick=app.getOptionBox("Barbers: ")  #Stores cutomers barber input

    cur.execute("SELECT reservation_time FROM reservation_table WHERE barber_
    #Finds reservations with barber / date input
    results=cur.fetchall()
    #Stores all the results in a 2D list
    print(d_pick)
    print(b_pick)
    print(results)

    bookedTimes=[] #Creates a blank 1D list
    for eachTime in results: #Passes through each reult
        bookedTimes.append(eachTime[0])#Inserts all results into new 1D list
    print(bookedTimes)

    time_slots=['7:00','8:00','9:00','10:00','11:00','12:00','13:00','14:00',
    #Lists with all availible time slots

    availible_times=[] #Create a blank 1D list
    for index_x in time_slots: #Passes through each index in time slots
        if index_x not in bookedTimes: #Checks if searched index found within
            availible_times.append(index_x) #If not, add into new list
        else:
            pass
    app.changeOptionBox("Time Slots: ",availible_times)
    #Updates option box with availible times
```

```
2023-04-15
Liam
[('8:00',)]
['8:00']
```

All correct data was stored, matched against input. Successfully retrieved all unavailable times.

Confirmation:

```
def confirmation():
    final_date=app.getDatePicker("date") #Stores customers date input
    final_time=app.getOptionBox("Time Slots: ") #Stores customers time input
    final_barber=app.getOptionBox("Barbers: ") #Stores customers barber input
    final_hairstyle=app.getOptionBox("Hairstyles: ") #Stores customers hairstyle input
    cur.execute("SELECT price FROM hairstyle_table WHERE hairstyle=?",[final_hairstyle])
    #Searches for the price of the customers hairstyle
    final_price=cur.fetchone() #Stores the price of a variable

    print(final_date)
    print(final_time)
    print(final_barber)
    print(final_hairstyle)
    print(final_price)
```

```
2023-04-22
16:00
Dave
Buzz Cut
(20.5,)
```

All correct data was stored, matched against input. Fetched the correct price from the table.

Inserting reservation:

```
def insert_reservation():
    username=app.getEntry("inputU")#Stores customers username input
    final_date=app.getDatePicker("date") #Stores customers date input
    final_time=app.getOptionBox("Time Slots: ") #Stores customers time input
    final_barber=app.getOptionBox("Barbers: ") #Stores customers barber input
    final_hairstyle=app.getOptionBox("Hairstyles: ") #Stores customers hairstyle input
    cur.execute("SELECT price FROM hairstyle_table WHERE hairstyle=?",[final_hairstyle])
    #Searches for the price of the customers hairstyle
    final_price=cur.fetchone() #Stores the price of a variable
    final_message=app.getEntry("message")

    print(username)
    print(final_date)
    print(final_time)
    print(final_barber)
    print(final_hairstyle)
    print(final_price)
    print(final_message)
```

```
jobiar
2023-06-24
14:00
Timothy
High and tight
(12.0,)
Taper fade on sides
```

All correct data was stored, matched against input. Fetched the correct price from the table.

Recommended hairstyle:

```
def generate_RH():
    head_shape=app.getOptionBox("Describe your face shape:        ") #Stor
    head_width=app.getOptionBox("Describe your face width:        ") #Stor
    u_maintainence=app.getOptionBox("Describe level of hairstyle maintenar
    hair_length=app.getOptionBox("Describe your desired length of hair:  '

    cur.execute("SELECT hairstyle FROM hairstyle_table WHERE face_shape=?
    #Searches for hairstyles that attributes matches the customers input
    r_hairstyle=cur.fetchall() #Stores search results into 2D list

    print(head_shape)
    print(head_width)
    print(u_maintainence)
    print(hair_length)
    print(r_hairstyles)
```

```
Round
Narrow
Non/Little
Short
[('Longer buzz cut',), ('Swept undercut',)]
```

All correct data was stored, matched against input. Fetched all hairstyles with same input attributes from the table.

Validate date:

```
def valid_date():
    date=app.getDatePicker("date") #Stores customer date input
    today=date.today() #Stores the current date

    print(date)
    print(today)
```

```
2023-04-02
2023-04-05
```

All correct data was stored, matched against input. Fetched the correct current date by using date library.

Retrieving all user reservations:

```
def get_reservations():
    app.deleteAllGridRows("view_r") #Resets the table
    username=app.getEntry("inputU") #Stores username within variable
    cur.execute("SELECT reservation_date,reservation_time,barber_name,
    #Fetches reservation details with the username inputted
    reservations=cur.fetchall() #Stores reservations within variable

    print(username)
    print(reservations)
```

```
jobiar
[('2023-04-15', '8:00', 'Liam', 'Wolf Cut'), ('2023-04-15', '13:00', 'Timothy',
'Spiky'), ('2023-04-15', '8:00', 'Timothy', 'Loose pompadour'), ('2023-06-15', '
15:00', 'Lucus', 'Wet Mop'), ('2023-06-24', '14:00', 'Timothy', 'High and tight'
)]
```

All correct data was stored, matched against input. Retrieved all the correct reservations under that username.

Compare verification code:

```
def authenticate_code():
    user_code=app.getEntry("code_input") #Stores user input
    print(user_code)
    print(verify_code)

    if user_code!=str(verify_code): #Compare user code to actual code
```

```
170543
170543
```

All correct data was stored, matched against input. Retrieved the correct verification code generated.

Sending verification code:

```
def send_authenticate_code():
    username=app.getEntry("inputU") #Stores the customer username
    cur.execute("SELECT email FROM user_table WHERE username=?",[username])
    #Fetches email within the user table with the customers username
    tbl_email=cur.fetchone() #Stores customer email address

    email_from = 'thebarbershop341@gmail.com' #Stores the sender email
    password = 'dvxyukjbjyaaqmas' #Password that gains access to senders emails
    email_to=str(tbl_email[0]) #Sets the recipient email as customer email

    global verify_code
    #Sets the verification code as global to use in other function
    verify_code=random.randint(100000,999999)
    #Generates the authentication code using random library
    email_content=("Verification code - "+str(verify_code))
    #Email content itself that going to be sent
    print(username)
    print(tbl_email)
    print(verify_code)
    print(email_content)
```

```
jobiar
('jobiarhussain83@gmail.com',)
349800
Verification code - 349800
```

All correct data was stored, matched against input. Retrieved the correct verification code and the email address from the table.

Sending receipts:

```
def send_reciept():
    username=app.getEntry("inputU") #Stores the customer username
    cur.execute("SELECT email FROM user_table WHERE username=?",[username])
    #Fetches email within the user table with the customers username
    tbl_email=cur.fetchone() #Stores customer email address
    final_date=app.getDatePicker("date") #Stores customers date input
    final_time=app.getOptionBox("Time Slots: ") #Stores customers time input
    final_barber=app.getOptionBox("Barbers: ") #Stores customers barber input
    final_hairstyle=app.getOptionBox("Hairstyles: ") #Stores customers hairstyle input
    cur.execute("SELECT price FROM hairstyle_table WHERE hairstyle=?",[final_hairstyle])
    #Searches for the price of the customers hairstyle
    final_price=cur.fetchone() #Stores the price of a variable
    email_from = 'thebarbershop341@gmail.com' #Stores the sender email
    password = 'dvxyukjbjyaaqmas' #Password that gains access to senders emails
    email_to=str(tbl_email[0]) #Sets the recipient email as customer email
    email_content=f"""
    Your reservation details,

    Date - {final_date}
    Time - {final_time}
    Price - {final_price[0]}
    Barber - {final_barber}
    Hairstyle - {final_hairstyle}""" #Email that will be sent to the user
    print(username)
    print(tbl_email)
    print(email_content)
```

```
jobiar
('jobiarhussain83@gmail.com',)

    Your reservation details,

    Date - 2023-05-23
    Time - 13:00
    Price - 15.0
    Barber - Dave
    Hairstyle - Military
```

All correct data was stored, matched against input. Retrieved the correct email address from the table.

Retrieving barber reservations:

```
def barber_get_reservation():
    app.deleteAllGridRows("B_view") #Resets the table
    barber=app.getEntry("barberU") #Stores barber name within variable
    b_date=app.getDatePicker("b_date")
    cur.execute("SELECT username,reservation_time,hairstyle FROM reservat
    #Fetches reservation details with the barber name and date inputted
    b_reservations=cur.fetchall() #Stores reservations within variable

    print(barber)
    print(b_date)
    print(b_reservations)
```

```
Liam
2023-04-15
[('jobiar', '8:00', 'Wolf Cut')]
```

All correct data was stored, matched against input. Retrieved the correct reservations from the table.

Deleting reservations:

```
def delete_reservation(row_num):
    delete=app.getTableRow("view_r",row_num) #Stores the reservation being deleted
    result=app.questionBox("Are you sure?",f"Are you sure you want to delete the re
    #Ensures user if wants to delete reservation

    print(delete)
    print(result)
```

```
['2023-04-15', '13:00', 'Timothy', 'Spiky']
True
['2023-06-24', '14:00', 'Timothy', 'High and tight']
False
```

All correct data was stored, matched against input. Retrieved the correct reservations from the table. Correct answer from question box was stored.

Deleting old reservations:

```
def delete_old_reservations():
    today=date.today() #Stores the current date
    cur.execute("SELECT reservation_date FROM reservation_table"
    #Fetches all reservations currently in the table
    all_reservations=cur.fetchall()

    print(today)
    print(all_reservations)
```

```
2023-04-06
[('2023-04-15',), ('2023-04-29',), ('2023-04-15',), ('2023-06-15',), ('2023-06-2
4',), ('2023-05-23',)]
```

All correct data was stored. Retrieved the correct current date using the date library. Also retrieved all reservations within the table.

Retrieving the view more information:

```
def view_more(row_num1):
    view_this=app.getTableRow("B_view",row_num1) #Assigns the record selected to a variable
    b_date=app.getDatePicker("b_date") #Assigns the dates to a variable
    barber_username=app.getEntry("barberU") #Stores username input within a variable
    cur.execute("SELECT username,phone_number,email FROM user_table WHERE username=?",[view_t
    #Fetches all relevent customer details
    user_details=cur.fetchone() #Stores the details within a variable
    cur.execute("SELECT message FROM reservation_table WHERE reservation_date=? AND reservati
    #Fetches the message that the user may have inputted
    message=cur.fetchone() #Stores the message within a variable
    con.commit()

    print(view_this)
    print(b_date)
    print(barber_username)
    print(user_details)
    print(message)
```

```
['jobiar', '14:00', 'High and tight']
2023-06-24
Timothy
('jobiar', '07958642818', 'jobiarhussain83@gmail.com')
('Taper fade on sides',)
```

All correct data was stored, matched the input. Retrieved the correct details from the table.

Editing hairstyle details:

```
def edit_details():
    edit_hairstyle=app.getOptionBox("Edit: ") #Retrievs hairstyle name input
    new_price=app.getEntry("New price: ").strip() #Retrievs new price input
    shape=app.getOptionBox("New shape: ") #Stores face shape input
    width=app.getOptionBox("New width: ") #Stores face width input
    maintainence=app.getOptionBox("New maintainence: ") #Stores hair maintainence input
    length=app.getOptionBox("New hair length: ") #Stores length input as variable

    print(edit_hairstyle)
    print(new_price)
    print(shape)
    print(width)
    print(maintainence)
    print(length)
```

```
Pompadour
27
Square
Wide
Lot
Medium
```

All correct data was stored, matched the input.

Retrieving latest version of hairstyles:

```
def fetch_hairstyle():
    cur.execute("SELECT hairstyle FROM hairstyle_table")
    con.commit()
    #Fetches all  hairstyles from hairstyles table
    hairstyle_tbl=cur.fetchall()

    print(hairstyle_tbl)
```

```
def select_hairstyle():
    hairstyles=fetch_hairstyle() #Runs function to get hairstyles
    app.changeOptionBox("Hairstyles: ",hairstyles) #Updates list of hairstyles
    app.hideAllSubWindows(useStopFunction=False)
    app.showSubWindow("Select hairstyle")
```

```
[('Buzz Cut',), ('French Crop',), ('Curtains',), ('Quiff',), ('Slickback',), ('Crew Cut',), ('Man Bun',), ('Bald',), ('Shag',), ('Edgar',), ('Spiky',), ('Crop Top',), ('Comb Over',), ('Ivy Leugue',), ('Mohawk',), ('Bowl Cut',), ('Caesar',), ('Flat Top',), ('Induction Cut',), ('Perm',), ('Wolf Cut',), ('Wet Mop',), ('Bl owout',), ('Undercut',), ('Faux Hawk',), ('Military',), ('Mullet',), ('Pompadour',), ('Man Bob',), ('Blunt cut fringe',), ('Short textured ',), ('Loose pompadour',), ('Spiky mordern undercut',), ('Long fringe',), ('Undercut viking chic',), ('Flat top',), ('Flow',), ('High and tight',), ('Curtain fringe',), ('Scissor crop',), ('Top knot',), ('Burst',), ('Mop top',), ('Elephant trunk',), ('Brush up',), ('Tapered curls',), ('Longer buzz cut',), ('Swept undercut',), ('Irregular top',), ('Stir fired top',), ('Angular fringe',)]
```

Successfully retrieved the latest version of the hairstyle table. I realised that when the create interface function is run, the list of hairstyles where outputted 3 times. This was a mistake as before the window with the option box full of hairstyles are displayed, the function runs again to provide an updated list. This outputted 3 times due to the program running this function every time I create an option box that outputted the hairstyle list. To fix this, I define the option box and left it blank. This prevents the function being run unnecessary at the beginning as it normally runs before displaying the window.

```
##### SELECT HAIRSTYLE #####
app.addLabel("hairstyle_pick","Select a hairstyle:",)
app.addLabelOptionBox("Hairstyles: ",[])
```

Deleting hairstyle:

```
def confirm_delete_hairstyle():
    delete_hairstyle=app.getOptionBox("Delete: ") #Retrives hairstyle being deleted
    result=app.questionBox("Delte hairstyle?",f"Are you sure you want to delete the
    #Confirms if barber wants to delete hairstyles

    print(delete_hairstyle)
    print(result)
```

```
Stir fired top
False
```

All correct data was stored, matched the input. Stored the correct answer to question box.

Comments:

As I've now tested all the variables within the program, all variables work as intended. After checking the output of the variable and comparing the output to the data within the table or the input I put in, this confirmed to me that all the variables had the expected data within them. With this thorough test, I can conclude this test was a success.

Black box testing

Within all 3 videos above, I've also tested each feature and function within the program. From this testing I can conclude that all feature that I have implemented work as intended, they successful output the correct data whilst also making the expected changes to the database. I've done a series of task to show how each feature works.

-Logging in: **video 1 (0:25-2:25)**

-Creating a reservation / recommended hairstyle: **video 1 (2:25-4:25)**

-Email verification code / receipt: **video 1 (1:45-2:25 / 4:10-4:30)**

-Viewing deleting reservations: **video 1 (4:30-4:40 / 6:15-6:45**)

-Home button: **video 1 (6:50+)**

-Creating an account: **video 3**

-Barber logging in: **video 2 (00:00-0:45)**

-Barber view reservations: **video 2(0:45-3:00)**

-Barber editing hairstyle: **video 2 (3:00-4:05 / 4:40-5:15)**

-Barber creating hairstyle: **video 2 (4:05-4:40)**

-Barber deleting hairstyle: **video 2 (5:20:5:55)**

I can conclude that every feature and button work as intended. All inputs that need to be validated are validated successfully and the database successfully reflects all changes that have been made.

Beta / User testing:

Another test that was carried out was user testing. I handed over my program to my clients and random customers waiting to get a haircut. This was a perfect opportunity to see the usability and if the product was user centred. I gave each user a task and without any support, asked them to carry this task out. While they carried out the task, I kept record of the progress and at the end asked for their feedback and comments on the program.

Tests/tasks:	Tester	Outcome	How successful/comments
Create an account	Customer 1	They were able to successfully create a valid account. They did however get error messages after inputting his phone number wrong, he understood that his phone number input was wrong and was quick to realise he missed a digit.	This was a success as they said they could easily create an account as the create account button was clearly labelled. He stated that the phone number error message was helpful as without it, he would have inputted the wrong number. The customer also states that after they create an account, the login page should be presented after instead of restarting the program to access the login page. This feature was missed while development, I will ensure this idea is implemented next.
Login with new account	Customer 1	After inputting the details, he just created. He was able to pass on to 2 factor authentication. After checking his inbox, he was able to login into his new account	This was another success as the customer was able to login to his account that he just created.
Book a reservation on the 27th of May 2023 at 3pm	Customer 1	Once they logged in, I asked him to book a reservation at a specific time and date, they were able to book a reservation at this time quite easily	This was successful as they booked a reservation with ease. They came across no problem as they said that everything was laid out in a logical way/order. They said this was familiar to other booking systems. They also stated that the icons helped him navigate the program easily and also gave the program a more friendly and approachable look.
Book a reservation where they don't know what haircut they want	Customer 2	I asked another customer to book a reservation but asked them to not know what haircut they wanted. At first, they were confused on what I meant, but after reading the page. They realised that they should click the button that led to recommended hairstyle feature. After inputting their details, they did get a null result but after changing some preferences, they were able to get a result.	This was somewhat successful, as I did give a major hint on what to do. This will be difficult to test as I will need to find a tester who is already unsure on what hairstyle they want. As telling them they should be unsure, they were able to figure out what needed to be done. Also, the tester did get a null result to begin with, this is not ideal but, in the end, was able to get a result that he was happy with. He stated that the page was cluttered with too many buttons. This was understandable as there are many more buttons on that page compared to the other pages.
View the receipt they received	Customer 2	After they booked a reservation, I asked them to view the receipt they had received. This was easy enough for the customer as everything after booking was done automatically, they were able to open their inbox and view the reservation email.	This was a success as they were able to easily view their reservation receipt on their phone. He stated that this was very useful as this will help remind him when their reservation was if he ever forgot. This also prevented him needing to login back into the program and view the reservation details there.
Find the reservation created within the program	Customer 2	After they created the reservation, they were presented with the menu window. I asked them to find the reservation details within the program. Very quickly, they were able find their reservation.	This was a success as he was able to spot that on the menu, he could view his reservation. This was because there was a button with a view icon. Once he clicked, easily found his reservation details within the table.
Delete the reservation he just made	Customer 3	At the beginning of the test, I asked a customer to also delete the reservation after he created it. Once he created it, he was on the menu page. After searching for a delete option, he realised that within the table, there was a delete	This was somewhat successful as the customer wasn't able to quickly identify where to delete reservation. He stated that it would have been easier if the view button on the menu also said delete. After listening to this feedback, I've decided to change the label to state view/delete reservations.

		option. Once clicked he was able to delete the reservation.	
Book a reservation and tell the barber specifics regarding the haircut	Customer 4	I asked the customer to book a reservation and also told him to specify that he wanted a taper fade along his haircut. As he arrived on the select hairstyle window, he was confused about my task as he didn't see any option to select a hairstyle and the type of fade. I then had to tell him that it was on the confirmation page. After this, he was able to input the fade option.	This was unsuccessful as I had to tell the customer where to input the details for the barber. He stated that it should have been with the selecting hairstyle page as this made the most logical sense. After getting this feedback, I will make the change where the input box will be moved to select hairstyle page.
To view all reservations on the current date	Client, Dave	I asked one of my clients, Dave, to find all the reservations currently this date. After successfully login in with the details I provided, he was able to select the correct date and view all the reservations under his name	This was successful as Dave was able to easily find the reservations that I asked for without any help required. This task took little time for him to complete as he said that process was in a chronological way.
Contact any customer and tell them their reservation has to be rescheduled	Dave	Once he viewed all the reservations, I asked him to contact a customer by finding their contact details. He quickly released that he needed to click the view more button on the table where he successfully found the contact details for customer that booked that reservation.	This was successful as Dave was able to find the contact details for the reservation he selected. He stated that the button to view more was in a good location as it made sense for the details to be under that specific reservation that he selected.
Edit the price of haircut, French crop, to £25	Client, Lucus	I asked another barber, Lucus, to login with his account and edit a hairstyle. Without any assistance, he was able to edit the hairstyle price within a short timeframe.	This was a success as Lucus was easily able to navigate the program to edit the hairstyles. He stated that the feature was located in a great spot that was easy to navigate quickly. He also said the interface for editing the hairstyle was laid out in a way that was easy to understand/make modification.
Create a new hairstyle	Lucus	Now that he knew how to edit a hairstyle, Lucus successfully was able to create a new hairstyle quickly as he was already familiar with the edit hairstyle feature.	This was another success as Lucus could quickly navigate and create a new hairstyle. As the create hairstyle window was similar to the edit hairstyle window. He stated that since the pages were similar, he found it very easy to create a new hairstyle. The similarities of the windows are a benefit as this produces a uniform experience, allowing the users to be familiar with all pages even if they have only used one.
Delete the hairstyle just made	Lucus	I asked Lucus to delete the hairstyle he just had created. He was able to successfully navigate to delete hairstyle window and delete the hairstyle. This was completed in a short period of time.	This was a success as Lucus was able to quickly delete the hairstyle that he just created. He liked the fact the all the features related to the hairstyle tables was within a simple menu and that the delete window was simple. The only issue he had was that window size is quite large considering only little bit of things are happening on the page. I've decided to not make changes to the size of the window as I want all the edit table windows sizes to continue to match.

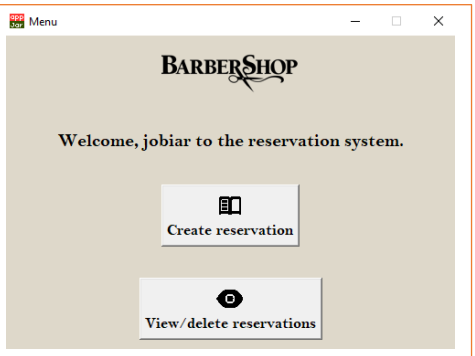
Comments:

This test was another success where I also gained valuable feedback that will allow my program to become even more user friendly. The program at its current iteration has been proven to be user centred as both barbers and customers can easily navigate and carry out most tasks with no prior information and additional support. The changes that I next make will be based of the feedback received. These changes being that the view reservation button on the menu window should also state that it can delete, move the message box to select hairstyle window as it makes sense to have the additional information about the haircut to be also in the select hairstyle window instead of the confirmation page and after creating an account, present the user with the login window

Changes made based on feedback:

Modifying view reservation button:

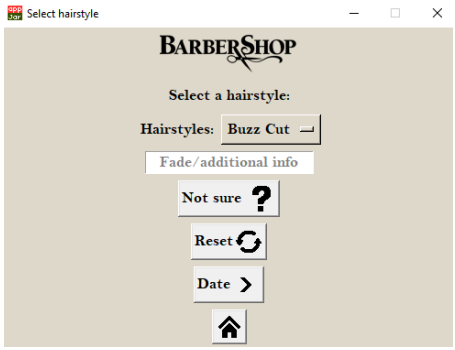
```
##### MAIN MENU #####
app.startSubWindow("Menu") #Create a new subwindow
app.setSize("500x350") #Sets size of window
app.setResizable(False) #Fixes size of window
app.setBg("#ded8ca", override=True)
app.addImage("logo12","logo.png", compound=None)

app.addLabel("Welcome",[]) #Title of window
app.getLabelWidget("Welcome").config(font=("Bell MT", "15", "bold"))

app.addIconButton("Create reservation",select_barber,"book-alt-2", align="top")
#New Reservation button
app.addIconButton("View/delete reservations",get_reservations,"view", align="top")
```



As one of my testers had trouble finding the feature to delete reservations, he suggested that I should also state that the view button deletes. Based on this feedback, I changed the label of the button to now display, view/delete, now allowing customers to quickly identify where they are able to delete reservations as well. The only issue with this was that the icon doesn't also convey the idea of delete but as I cannot add 2 icons to a button. As this is only a minor problem, I've decided to leave the icon as view only as it still functions as view.

Moving message box:

```
##### SELECT HAIRSTYLE #####
app.startSubWindow("Select hairstyle") #Creates a new subwindow
app.setSize("500x350") #Sets the size of the window
app.setResizable(False) #Prevents change to size window
app.setBg("#ded8ca", override=True)
app.addImage("logo5","logo.png", compound=None)

app.addLabel("hairstyle_pick","Select a hairstyle:",)
app.addLabelOptionBox("Hairstyles: ",[])
app.addEntry("message") #Entry box for messages for the barber
app.setEntryDefault("message", "Fade/additional info")
```



One of my testers had trouble finding the option to input additional information when on the selecting hairstyle screen. As it was on the confirmation screen, he was confused on where to carry out the task that I gave. Based on this experience, I decided to move the message box to the select hairstyle window as it makes sense to input information regarding the haircut, on the haircut selecting window than the confirmation screen. Another change I made was in the input box itself. I also changed it to say fade as well to further prevent confusion on where to input information regarding the fade. The only issue I have with this change is that it further clutters this window compared to others, as it has many options. But as all the features on this page are needed, I've decided to make no further changes.

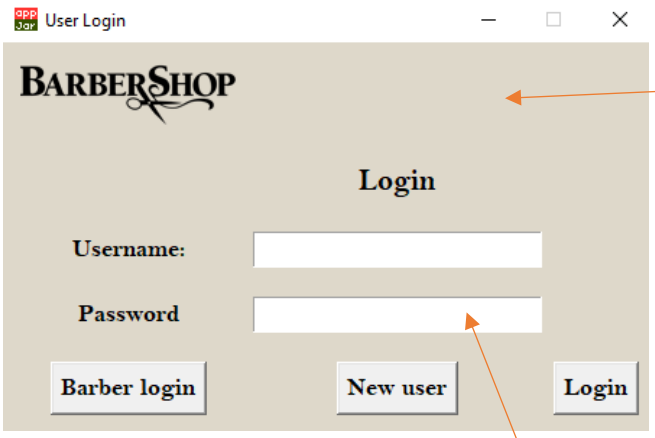
Display login after create account:

```
def create_account():
    new_Uname=app.getEntry("new_username").strip() #St
    new_Pword=app.getEntry("new_password") #Stores pas
    re_enter_pword=app.getEntry("new_password2") #Stor
    new_Pnumber=app.getEntry("new_phone").replace(" ",
    new_email=app.getEntry("user_email")

    employee=0 #Creates a new variable to set is_emplo
    valid=validate_details(new_Uname,new_Pword,new_Pnu
    #Runs details through function to check validity

    if valid==True: #Checks if details are valid
        cur.execute("INSERT INTO user_table VALUES (?,
        con.commit()
        app.hideAllSubWindows(useStopFunction=False)
        app.show("User Login")
```

One of my testers suggested that I should display the login page after successfully creating account. As this prevents having to restart the program to access the login page. This was an easy fix, if all the details were valid, the function inserts the details into the table and now also hides all previous sub windows and then presents the login page once again.

Evaluation of every major window:

Login window:



This crème background on every window was chosen as it matched the same colour used within the interior of the barbershop. I planned on adding a two-tone colour scheme that would have provided contrast, but I couldn't implement this. If I used a background image, this would have looked unprofessional as the borders of the widgets would not have matched.

3 buttons options that either; login, create an account or login as a barber. These are presented at the bottom of the page. These buttons all match with same font and background colour, providing a more professional look and experience. The text colour is in black as this contrasts with the light-coloured background that allows them to be easily visible.

2 entry boxes that have been clearly labelled that allows the customer to know what needs to be inputted. These are centred in the screen as they are the main function of this window, login in. The boxes are also perfectly aligned with each other, improving the symmetry and aesthetics. They were originally planned to be rounded boxes but as this isn't possible in AppJar, couldn't be implemented. This however still provides a clean professional. The background of the entry box is lighter compared to the background, allows it to be easily differentiated from the light background. Also providing more contrast when they input black text within the entry box. Each input is compared to the database as only authorised users can enter.

Why was this window necessary?

This screen is essential as it acts the gateway to enter the program. This is needed to allow users to input their login details so they can be identified by the system and securely logged in. Without this window, no record of who made the reservation can be made, leading to more confusion. This window also allows users to create an account if they do not have an account to login to the system and allow the barbers to login to their side of the program.

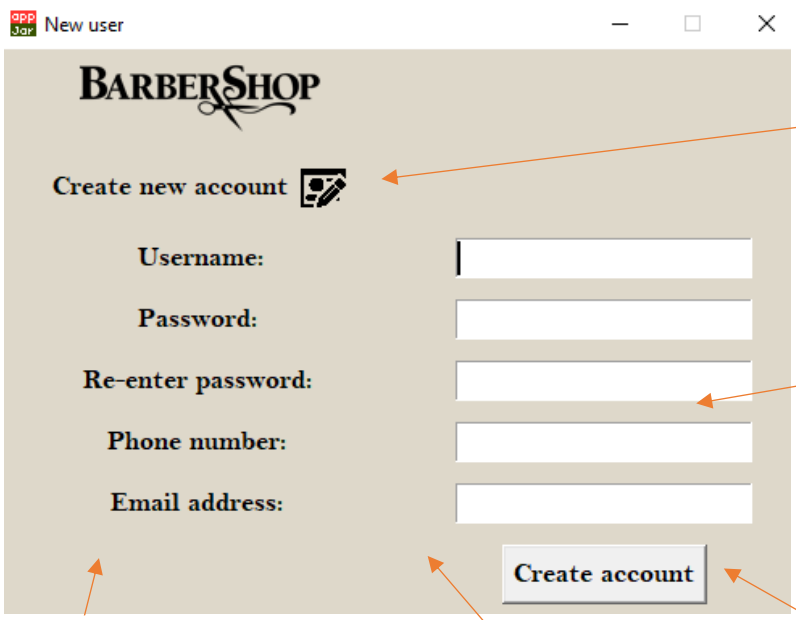
How did I ensure robustness and security?

Once the details are inputted into the entry boxes and user clicks on the login button. The system compares the details inputted and checks weather if any details match the input. If the details do not match, the system gives an error message and prevents access to the system. However, if the details do match, as another layer of security. Two factor authentication has been implemented as another way to confirm their identity. If the random generated verification codes sent to the account holders' email do not match with the inputted code, the system will reject the user even if the username and password inputs are correct. But if the authentication code matches, the system can confirm that the user accessing the account is the account holder. It also successfully prevents barber account login in via customer login, further preventing any issues down the line.

Final evaluation:

With this window, me and my client are extremely happy with this page as the window itself looks attractive and professional. Even though the two-tone design couldn't be implemented, I am still pleased that this window came out this way. This window matches with the colour scheme of the program whilst also being easy to read and understand. Along with its looks, it provides great security by ensuring that users are fully verifies before giving them access to a user's account. It also successfully allows the option to create a new account whilst also allowing an option for the employees to login. This page prevents many security threats/risks so overall believe this window was a success.

Create an account window:



Icon label that describes the purpose of this page. Icons that also visually describes what the purpose is, further improving accessibility of the window. Label also aligns perfectly with the logo and the labels for each entry box, making this window appear neater and more professional. Black text against a light background makes the text easier to read.

Entry boxes that allow the user to input each piece of detail that is required. All boxes are clearly labelled to prevent confusion on what is required. All boxes are perfectly aligned with each other, making the window look tidier. White coloured entry box contrasts with the blacked coloured user inputs, making the text easier to read.

Create account button, allows the user to confirm their action and create an account if all details are valid. Button position in the corner as the button will fall into the user line of sight after reading over their account details.

If the details are not valid, an error box will be presented to the user, explaining exactly where the issue is. The error box will persist on the screen till the user addresses it and understands the message.

Same crème background colour that matches with the theme of the barbershop and program.

Why was this window necessary?

This window is required for new users to be able to access the program. As the program only allows verified accounts access, any new user must be able to create an account in order to access the rest of the system. An account system is required as this stores all relevant information regarding that customer. It stores their contact details that allow the barber to contact their customers if any update or new have come about their reservations. An example of this may be that the barber isn't in on that day, causing the customer to reschedule in order to get their favourite barber. Other account details are required such as their username as this allows the barbers to know which customers they should be expecting, preventing any mistakes about which customer is next.

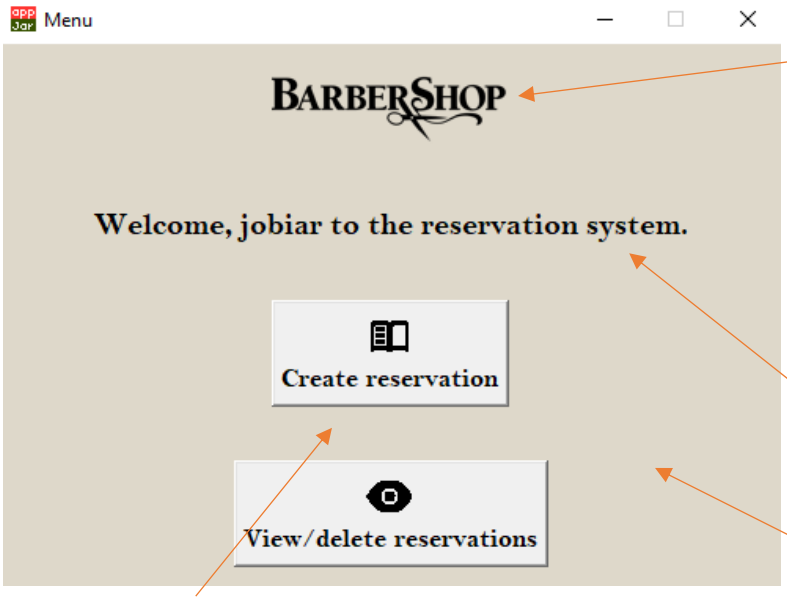
How did I ensure robustness and security?

This window validates every input the user makes to ensure only valid data is stored within the user table. The username is validated by ensuring that the number of characters is between 0 and 12 characters and if that username is unique, this prevents problems where 2 accounts share the same username. To ensure a secure password is created. The program only accepts passwords that are between 6 and 16 characters, have a capital letter and number within their password. To ensure that user input is what they intended, they must re-enter their password. If the passwords match, this ensures that the input is free from any input errors. If they do not match, the program will give a specific error message that notifies that the passwords do not match and prevents account creating until they match. The program validates the phone number by ensuring that they are all integers and that the number of characters are 11 digits, as UK numbers are all 11 digits. Lastly validates the email address by ensuring it contains an @ symbol.

Final evaluation:

Once again, I am happy with this window. The aesthetics of this window came out the way me and client wanted. This window successfully matches the same theme that is constant throughout the whole program. Every input that the user makes are all thoroughly validated and robust when switching from screens. Overall, this window was a success as it allows users to create validated accounts securely.

Main menu:



At the top of every window, proudly displayed the barbershops logo. This makes every window more representative of the barbershop. Allows every window to be instantly recognisable. Blacked out version of the logo contrasts well against the background, further making the logo stand out.

This label personally greets the customer and generally helps the user feel more welcomed to the experience. This helps match the same hospitality provided by the barbers. Black coloured text against a lighter background helps readability.

Same crème background colour that matches with the theme of the barbershop and program. Further providing the same uniform experience throughout the program.

These buttons provide the main functionality to the customer side of the program. They allow the customer to start the process of booking a reservation and view/delete their current reservations. Icon button to also describe the function visually whilst also drawing attention to the buttons. The icons enhance the aesthetics whilst increasing readability/accessibility. Used book icon as this mimics the idea of how the barbers used to record reservations in a book. Used the eye icons as this conveys the idea of looking at something like reservations.

Why was this window necessary?

This window is vital as this provides the options to access all functionality of the customer side of the program. Once the user is successfully logged in and validated, this window allows the user to carry out the main function, create a reservation. Once they have created a reservation, they will be able to view their reservations from this window that will be displayed again, after they have created their reservation. As this window allows access to everything the customer can access, the home button that will be present on every window after the menu will jump back this menu, once again allowing them to continue down another path through the program. To implement the options, I made use of buttons that, once clicked, branch to all other areas within this side of the program.

How did I ensure robustness and security?

When customers click the buttons on this page, the menu will be automatically hidden and then display the window that leads after clicking that button. The users can move back to the main menu by using the home button on each window. The user can do this freely as closing and displaying sub windows do not cause widgets to be created or deleted. This allows jumping from page to page, without any error. As this page exclusively uses buttons, this prevents any inputs from the customer being needing to be validated and prevents the risk of any malicious inputs being made into input boxes.

Final evaluation:

This window was a success as it successfully allows users to access all parts of the program. The aesthetics of the window matches mine and my client's expectation. It matches the final design quite well by having a similar colour scheme, used the same icons and displayed the barbershops logo in a similar layout. This window successfully allows the user to jump to new pages with little delays with no errors. The home button on every window successfully displays this menu window whilst hiding the windows that the customer is currently on. Overall, a complete success.

Selecting details/hairstyle:

At the top of this window, displayed is the barbershops logo. This makes every window more representative of the barbershop. Allows every window to be instantly recognisable. Blacked out version of the logo contrasts well against the background, further making the logo stand out.

A drop-down box and an entry box that allows the users to select a any hairstyle and additional options. Options are centred in the middle as this helps draw the users' eyes down the list of functions provided by the page. Entry box allows the user to input any details regarding the haircut such as any additional services they want, such as a beard trim as well. Black text against a light background for easy readability.

Home button at the bottom of every page that allows the user to go back access other branches of the program. Button at the bottom of the screen moves the button out of the way as it isn't related to the other functionalities provided by the other buttons. Icon button with no words describing its function as home icons is well known as go back to the beginning/menu icon.

A list of buttons all centred in the middle that allow access to all options. Allows them to access the generated hairstyle feature with another button to reset the list of hairstyles in case they don't like the generated list of haircuts. Next buttons that confirms their selection and moves onto the next window. Used ? as this conveys the idea of confusion. Used reset circle as this conveys the idea of going back to where you started. Arrow icons portrays the idea of moving forward. All buttons have icons that help visually describe the feature, increasing accessibility and readability.

Why were these windows necessary?

This window and other similar windows after that allow the customer to select their options, they are necessary as this provides the main functionality of the program. It provides them with information such as what hairstyles are available for them to select, what barber they want and the available time/date they can choose from. Gives them an option to select a reservation that works for them and what type of haircut they want, once they input their details, all saved within the reservation table. This window also allows access to the recommended hairstyle feature that allows user to give the program a chance to find hairstyle suits best for the customer based on what input they provide. Without these windows, users will not be able to create any reservations.

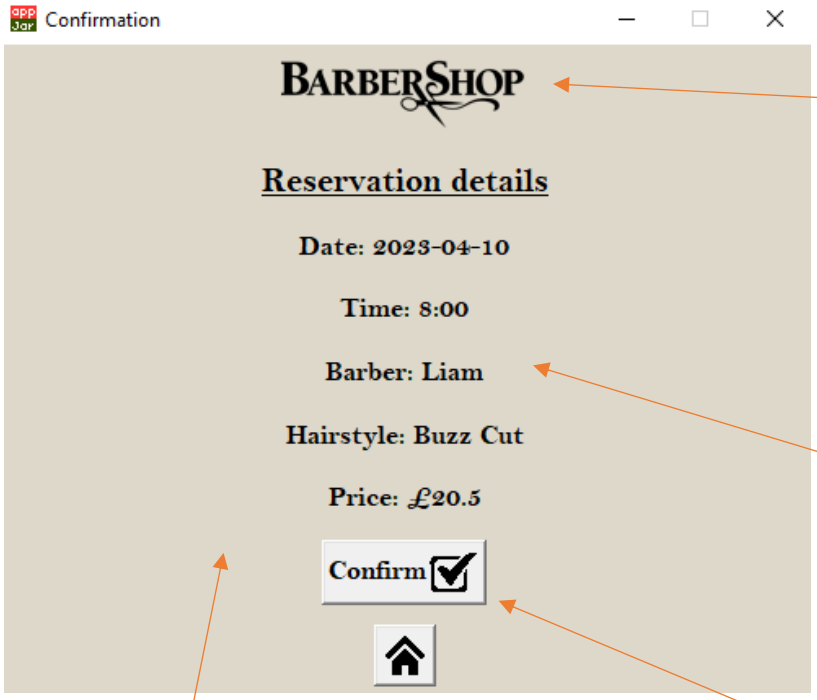
How did I ensure robustness and security?

When the user is inputting their additional information, the program validates their input before it stores the input within the database. It ensures that the input doesn't exceed 20 characters, may reduce the chance of malicious input (SQL injection) being accepted by the program. Further reducing the risk of security threats. When selecting a timeslot, it ensures that only available timeslots, times where individual barbers are free, can be selected. Even if a customer creates a reservation and straight after attempts to create another reservation at the same time/barber. The database is re checked and it re updates the available timeslots, further increasing robustness. Another measure to increase robustness was preventing customers from being able to book reservations in the past. All inputs, except one, don't use entry boxes. This further prevents validation being required and helps reduces security vulnerabilities being exploited, such as SQL injection.

Final evaluation:

These windows are a success as they allow customer to input all valid reservation details and create a reservation. It then successfully updates the table and robustly prevents any other reservation double booking. All windows share the same aesthetics and lay out in a logical way. Me and my client were please on the outcome and design of these windows as they successfully fulfil their function and most of the requirements. The only issue I have is that select hairstyle window has become a slight bit cluttered, but apart from that, these windows were a success.

Confirmation screen:



At the top of this window, displayed is the barbershops logo. This makes every window more representative of the barbershop. Allows every window to be instantly recognisable. Blacked out version of the logo contrasts well against the background, further making the logo stand out.

Labels that display all details that the user inputs. The title of the window is in larger font that draws attention to the text, read more easily as well. Labels displayed down the middle of the window, displaying all details in a logical order ranging from date/time and then haircut details. Black text against light background can easily be read.

Once the reservation has been confirmed, email with all the reservation details will be automatically sent out to the customer. Help remind the customers the reservation details if they happen to forget.

Icon button that allows the user to confirm the reservation, once confirmed. Will update the table and send out a receipt to their email. Icon helps visually describe the function of confirming whilst drawing attention to button. Further increases readability and accessibility.

Why was this window necessary?

This window provides a chance for the customer to look over their reservation details and ensure that they haven't made any input errors. This helps prevents customer being frustrated when creating a reservation that they are not happy with and overall helps provide a better experience for the customer. This window isn't an essential one as this doesn't prevent the user from creating a reservation. This window was purely implemented for the benefit for the customers.

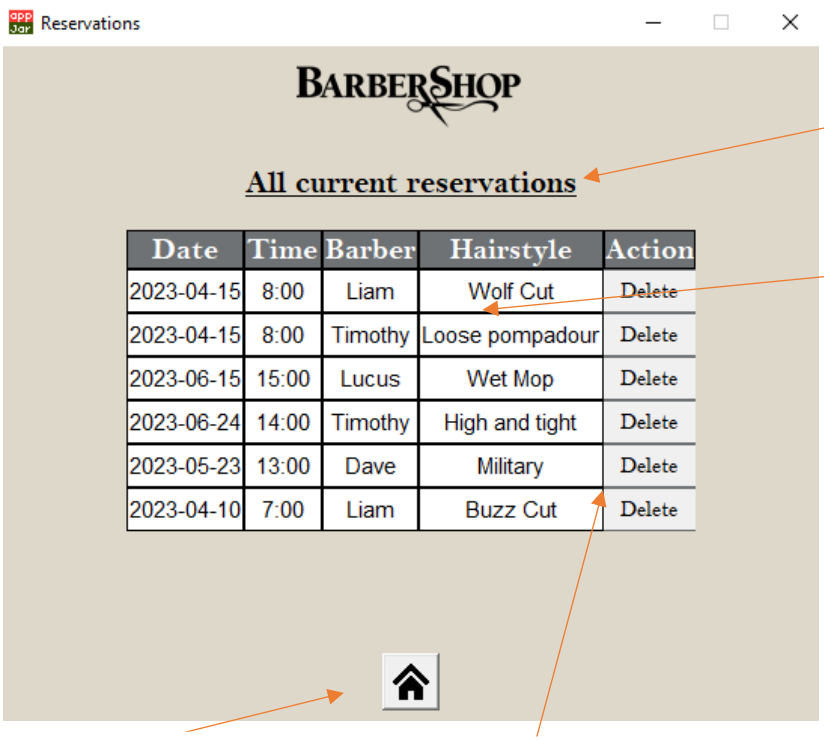
How did I ensure robustness and security?

As there are no inputs boxes, this prevents the need of validation and prevents any security risks involved with input boxes where users can input anything. This window ensures robustness as if the user decides to go back to the menu and change certain details. Once they come back to the confirmation window, all the labels will be updated with the latest inputs made by the user. If the prices have been updated by the barbers, when displaying the confirmation screen. The program will retrieve the latest version of the hairstyle prices from the hairstyle table and ensure that all correct prices have been presented to the user, this prevents confusion for the customer for when it comes to the reservation day, the barber notifying that the prices have been changed. Every time a customer creates a reservation, the program robustly displays the correct details without any errors.

Final evaluation:

This window overall was a success. It fulfils it job by allowing customers to check if the details they have inputted the correct details. The design of the confirmation page came out similar enough with the layout and colours to the final design. Even though some features, such as the upload image feature were scrapped. This still fulfils my client's requirements. It also successfully sends the receipt of all the confirmation details once the customer confirms the reservation. With all of this taken into consideration, I would say this window came out as a success.

View reservations:



Label centred at the top of the window explains the function of the window to the user. It being underlined and centred help draw attention to the label quickly as it can explain the use of the window. Black text against a light background increases readability.

The table, being the main focus of the window, is centred in the middle. Helping draw more attention to the most important part. It contains all current reservations under that username as other reservations are not important for a user to see. It displays all important information regarding the reservation, including the time, date, barber, and hairstyle. This information can be used to remind customers of the reservation details if they have forgotten. All other information, such as the message, have been abstracted away as this isn't essential information a customer needs in order to attend the reservation and it may make the table too large and unattractive on the window. Hairstyle/barber field is included as this may aid a user's decision to delete a

Home button on the bottom of the window allows users to jump back to the menu to go down other routes throughout the program. Icon used only as the home button is well understood as the menu/back button. Placed at the bottom as this moves out of the way, away from the main focus, the table.

The delete button allows customers to delete their reservations, a delete button is placed at the end of each reservation, allowing users to delete specific reservations.

Why was this window necessary?

This window displays all current reservations and provides an option to delete any reservations customers want. This is an important window as without it, if a customer cannot attend the reservation and do not notify the barbershop. The barbershop will be unaware, and a barber may lose an hour of their time with due to no customers available. This will cause frustration for the barbers and worse, a loss for the business. This window also provides another way for customer to be reminded of their reservation details, preventing them from coming in on the wrong time/date. This window also prevents the hassle for the customer, normally they would have to call up the barbershop in order to be reminded of the details or to delete a reservation, but with this window, can all be done easily within the program.

How did I ensure robustness and security?

As only verified users are able to login to an account, this prevents unauthorized users from being able to delete a customer's reservation without their knowledge. To ensure robustness, the program without any error, displays all correct reservation under that username specifically. If a customer deletes a reservation, the program ensures that specific reservation is deleted only, and once its deleted of the database, will also delete the reservation off the table the customer can see. When users create a reservation and come to view the reservation straight after, the program re checks the database and updates the list of reservations within the table, whilst prevents any overwriting or copies being created. This ensures the table adapts to any changes made to the reservation table under that username.

Final evaluation:

This window is a success as it successfully displays all reservations and also allows users to delete any reservation of their liking. This window is looks quite different from the final design, me and my client were still pleased with the result as the table approach, still looks professional and tidy. Another success is that there are no limits to how many reservations are presented, even if there are too many reservations, the program provides functionality to scroll through the list of reservations. Overall, a successful window.

Barber selecting view date:

Label that describes the purpose of the window, allows barbers to know which date they want to view their reservations on. An icon further helps to describe the windows function as the magnifying glass is commonly known/understood as search. Top centred label draws attention to it after the users looks over the date picker, icon further draws attention to the label after looking over the date picker. Black text against light background, again, help fit the theme of the program whilst making it easier to read.

The date picker is centred in the middle as it is the main focus of the window, it being large and centred in the middle focuses the users' eyes first on this date picker as it is the main focus. It allows barbers to select any date to see what reservation they have on the day that they picked. Each month has the correct number of days preventing the user from selecting a date that doesn't exist.

If the barber happens to select a date that contains no reservation. A pop up will be displayed to the barber and notifies that no reservations are presents on this day. I used an error box as this pops up in the middle of the screen, drawing focus to the error message. This box also prevents the barber from accessing anything else until they have addressed the error message they have just received; this helps endure they have read the message and understand the problem.

2 buttons that lead to different part of the barber side of the program, one allows them to continue and view reservations that have been selected and the other allows barbers to edit the hairstyle table. Edit hairstyle button is located at the bottom as it isn't related to any other feature on this window, this helps move it out the user's way but still allowing access to it, if needed. The view reservation button has an icon that describes the action of moving forward as this is what the button does. It moves on to the next part of viewing reservations.

Why was this window necessary?

This window is needed as it allows the barbers to select a specific date, they would like to view their reservations on. This windows doubles as the main menu for the barber side of the program as, once the barber securely logs in, it gives access to all the features that is accessible to the barber. This allows them to view reservations and also give the option to edit hairstyles. Without this window, barbers will not be able to access any other features the barber side has to offer. I've implemented this by adding a date picker that allows them to select a date and 2 buttons that can branch to all other areas within the program. As this window allows access to all features within the program, all pages after all have button that allows the barber to jump back this page and access all branches of the barber side of the program again.

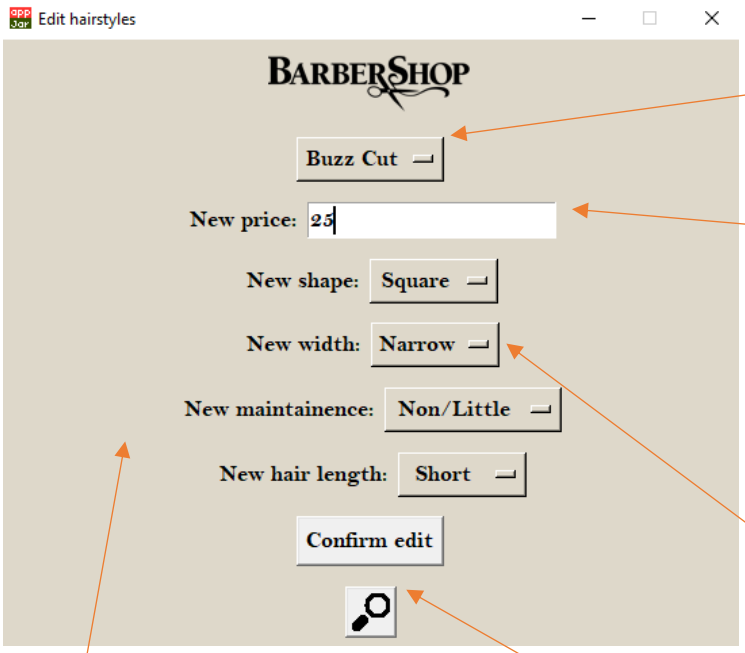
How did I ensure robustness and security?

When the barber selects a date with no reservations, instead of the program trying to insert an empty list into the table and causing many errors, without any errors, it successfully prevents that section of code being run but instead produces an error message. The program prevents input of unavailable dates, such as 30th of February. When customers click the buttons on this page, this window will be hidden and then display the window that leads after clicking that button. The users can move back to this window by using the search button on each window after this one. The user can do this freely as closing and displaying sub windows do not cause widgets to be created or deleted. This allows jumping from page to page, without any error. As this page exclusively uses buttons, this prevents any inputs from the customer being needing to be validated and prevents the risk of any malicious inputs being made into input boxes.

Final evaluation:

This window is a success as it functions as intended. It successfully allows barbers to select a date and view reservations on that date only, the edit button allows barbers access to all edit features within the program. I am pleased with the aesthetics of the program as it matches the theme of the program.

Editing hairstyle table:



Option box centred at the top allows the barber to select any hairstyle within the update version of the hairstyle table. At the top as this in same logical order as the fields within the hairstyle table. Starting from hairstyle name, prices, and attributes

Input box that allows the barber to insert any price they want to change the hairstyle 2. Used as entry box instead of an option box as the option box will only allow a limited number of price options, entry box allows barber to select any price they want. Black text against white background increases readability.

Series of option box centred down the middle of the window as all the options are the main focus of the window. Allows the barber to insert the new attributes for the hairstyle. Used an option box as there are limited options available. Label that explains each option, preventing confusion about what each option represents.

If the price format input is incorrect, a persistent error box will appear alerting the user of the problem. The box will not allow any other inputs into the program until the barber address the message and closes it. Helps prevent the issue occurring again.

2 buttons that allows the barber to confirm the edit or to back the barber side menu, select date. The confirm button is located right after all the hairstyle input options as this is the last input they will need to click in order to edit the hairstyle. The search icon button is at the very bottom as this is unrelated to the rest of the features on this page, therefore doesn't need to be at the centre of attention. Used magnifying icon as this matches the same icon on the select date page and also represents the idea of searching.

Why was this window necessary?

These updating window (edit/delete/create) are necessary as allowing the barbers to modify the hairstyle table helps prevent the hairstyle table and therefore the program from becoming redundant. If this feature wasn't implemented, they wouldn't be able to edit the table unless they were able to access the table. If they could have access, this will prevent them from being on top of the latest trends in hairstyles and prices. Ultimately leading the barbershop to become irrelevant/unpopular and losing out in potential business.

How did I ensure robustness and security?

I validated the new prices before they are inserted into the table. The only entry box input that is validated is the price, it must only consist of numbers and an optional decimal point in order for the program it insert the price into the database. This ensure robustness as this prevents errors/confusion occurring for customer when the price is in an incorrect format. This also reduces the chance of security risks as the number of inputs accepted by the database is very limited. I've limited the number of entry boxes to the maximum of 2 as this reduces the need of validation and therefore reduce the risk of security risks that are common in entry boxes. The name box that allows the barber to create a new hairstyle name isn't validated, this may increase the chance of security risks but as only barbers have access to this window, I feel confident that no immediate changes need to be made. The program ensures robustness by also keeping the list of hairstyles available for editing always updated to the latest version. This is important as if a barber creates a new hairstyle and realises there's a mistake. When fixing the new hairstyle, the list of options must be updated to include the new hairstyle that just been created. The program, without any errors, ensures the correct hairstyle is either inserted/deleted/modified every time.

Final evaluation:

These windows were mostly a success. Even though the program successfully edits/create/deletes hairstyles with the database. After evaluating, I have found an issue. Editing hairstyle doesn't display the current details, this would have been helpful as this would give perspective and would have helped barbers make better changes to the hairstyle. Apart from the that, me and my client were pleased with the aesthetics of the program as it continues to match the theme.

Evaluation success criteria:

No.	Criteria name	Have I met the criteria?	What are the client opinions?	Met?
1	Functional account system/user login	Yes, this criteria has been met. The program successfully ensures only valid account access the program whilst also having 2 factor authentication to verify the user further.	My client was very happy with the accounts system and that is successfully allows valid users in. He was happy that I was able to implement two factor authentication and that the program prevents users from login in via barber login, vice versa.	
2	Follow the design language of the barber shop	Yes, this criteria has been met. The colour scheme has matched the aesthetics of the barbershop interior. The same colour, font and logo do make the transition from the program to the barbershop. Even though the final design hasn't been matched completely with the two-tone colour and rounded boxes, this actual design was a suitable replacement	My client overall finds the aesthetics of the program a great match with the barbershop. He enjoys the fact that the colour scheme/font of the program is both visually appealing and that it matches with the theme of the barbershop. Even though he still prefers the final design, he is still happy with the end result.	
3	Functional system to make a reservation	Yes, this criteria has been met. The customer will be able to book a reservation with any available barber at any available date/time. They have access to all hairstyle the barbershop offers and are sent a receipt of all the reservation details afterwards.	My client was very happy with this as it fulfils all of the important requirements he set. He said that when he tested it, that it was easy to navigate and was easily able to create a reservation.	
4	Database containing all accounts, reservations, and services	Yes, the criteria has been met. The program can successfully create a database with all 3 tables within it. They are able to stores all the reservations, account information and all hairstyles within the table	My client was pleased that the database was fully functional and that the program was able to make changes to the table with no errors.	
5	Process to find a recommended hairstyle	Yes, the criteria has been met. The program successfully provides an option to allow the users to input their preferences and details in order to get a recommended generated result.	My client was again happy with this result as it functions as intended. He was especially happy with the accuracy of the result and believes this will help attract more customers to his barbershop. Even though it is possible to get null results, my client was still happy on the outcome due to high accuracy.	
6	Employees have the ability to update the services database	Yes, this criteria has been met. The barbers are successfully able to make any changes to the hairstyle table within the program without having to manually access the table or by writing code. The feature allows them to create a new hairstyle, delete a hairstyle or edit a current hairstyle.	My clients were very happy with this feature as they now can rest assure that this program can remain relevant it now gives access to the hairstyle table only to barbers. When they tested this feature, they said they found it very easy to navigate and make changes as it was laid out in a logical ways that allowed them to make changes intuitively.	
7	Able to run on employee's hardware and majority of customers	Yes, this criteria has been met. When testing the program within the barbershop computer. The program ran with no issue, it was able to run quickly with no delay or stuttering. When testing	My client was happy as the program can run quickly on their hardware with no issue.	

	hardware as well	on a range of different hardware, I came across the same results.		
8	Program able to prevent double bookings in the same timeslot	Yes, this criteria has been met. The program successfully prevents 2 reservation being made with the same barber on the same time and date. Instead of giving an error message, it just removes the option to select timeslots that are unavailable	My client was happy was this. As this was very important, this feature was to be expected. He's happy that it can prevent frustration in his co workers and customers.	
9	A dashboard /menu for both customers and barbers	Yes, this criteria has been met. The program successfully has a menu for both customer and barber side of the program. It allows access to all branches available to the user without causing any errors whatsoever.	My client was pleased with the result as he found the customer side menu aesthetically pleasing and welcoming to the user. He is happy that one window alone allows the customers to access all sides easily.	
10	Ability to cancel their reservation	Yes, this criteria has been met. The customer can cancel any reservation that isn't in the past. The are able to do this with a simple delete button that first ensures that the reservations selected is the reservation they want to delete.	My client was happy that customers are able to easily view and delete the reservations easily. He was equally happy with the table look compared to the calendar system as it still provides the same functionality in a simpler way. From retrospect, he understands that the calendar system may have been too complex for something so simple.	

Project successes:

Window/function	What was successful?	Why was this a success?	Further improvememts?
All windows	All windows look aesthetic and match the same theme	I ensured to keep the same look and feel throughout the whole program. Kept the same colours, logo positions, used icons. Made sure widgets aligned properly on the screen and consistant window/text sizes.	I could figure out a way to add rounded widgets and a larger rounder box around the area of focus like I have done in my final design.
Login	It prevents access to invalid users and barbers	The ensure that the details match the details within the database and ensure that only customer accounts can login.	I could have had a reset password using the email system if a user forgorts their password.
Create account	Allows users to create an account and login to the system whilst prevent invalid account details	The program ensures only valid details are allowed to create an account. This includes unique usernames and secure passwords. Once everything was valid, it successfully inserted the new account into the table.	Validate the email address/phone number before creating the account. I can send a verification code to user to verify their contact details
Two factor authenticion / sending reciepts	Successfully sends a random verification code to email and ensures both input and code match before allowing access to the system. Sends reciept to user with reservation details	Verifies users securley to enure the account holder has access to their account. Reasearch was able to teach me how to send emails via python. At first I thought this was difficult but after breaking the code down and understanding what each line does, i found it much easier to implemnt into the system.	I could use the email function to create a reminder system, allow users to rest password. Also include barbershop logo within the email itself.
View/delete reservations	Being able to output all reservations under that username and allowing	Easily allows users to view reservation and delete reservations due to appjars table that made it much easier to implement this feature. After doing learning more on AppJar's website,	I could allow the customer to make edits to current reservation, allow them to edit

	customers to delelt individual ones	was able to implement this feature and this knowledge also came	the hairstyle or message they wrote.
Generate recommended hairstyle	Produces accurate recommnedation based on user inputs	This is an success as this allows customers to try something new, this may help find hairstyles that customers may love and would normally never had considered before. The program is able to produce highly accurate results pureley due to the expertise of my client, who provided all haristyle and their attributes.	I could increase the number of questions asked to gain even more accurate results. Increase the number of haristyles within the table to reduce the number of null results. Display photos of the hairtyle generated, to help give a better idea of the hairstyle would look.
All create reservation windows	Able to create reservations quickley and easiliy	This is a success as users are able to book reservations, without much effort and time. Saves the hassle users normally have to face in order to book a simple reservation. At the end, successfully saves all the correct details within the table	Allow them to also select additional services from option box, such as cut and wash and beard cut. Have different prices for children/seniors
Prevent double booking	Program automatically removes the option to select unavailible timeslots	This is a success as this prevents confusion and fustration is both the barber and customers if 2 customers come for a haircut at the same time. Instead of giving an errror message if the time is already booked, the program is one step ahead and removes the option enitirely.	I cannot think of any improvements to features but can try to make the code more efficient.
Delete old reservations	Program autmatically deletes all reservations that are in the past before the interface function is even run.	This prevents the reservation becoming too large in the future. In the next 5 t0 10 years later, if the old reservations were still present, this could slow down query time and the database file can become much larger. As this data is no longer deleted, it made sense to delete off old reservation. By researching online, I learnt how to use date time library that allowed me to create this function	I cannot think of any improvemnts to features but can try to make the code more efficent
Modify hairstyles table	Allows the barbers to make any changes to the hairstyle table	This helps prevent the system from beoming redundent as barbers can keep up with the latest trends and prices. Allows them to expand their services without the need of a programmer to manually insert the data into the database.	Dsiplay current hairstyle details that allow the barber to make better modifications to the hairstyle record. Prevent them from under/over estimating the value of a hairstyle as they can now compare to the old prices.

Project failures/missed requirements:

Window/function	What went wrong?	Why did this go wrong?	How could I further improve this?
Upload image	Customers cannot upload an image of the hairstyle they want	I originally planned on implemented this feature but when researching on how to upload images, it become too complex and would have required more time to have implemented. I would also have to find a way to store the uploaded images so the barbers would be able to see	Find a loaction to be able to store the uploaded images and create the code that allows to upload images. The user would then be able to upload images on the confirmation page, like I intended in the final design

Reminder system	Shortly before the reservation, program cant send an email reminding the customer that theres a reservation	Whilst researching online, I learnt about a way to send emails at a given time using the datetime library in python. The reason this wasn't implemented was that this would have required the program to be constanly running in order for this to happen. This wasn't practical so I decided to scrap the feature all together.	Find a method that would allow to automate the function that would send the email even if the program was closed.
Late fee	If a customer happen to not attend a reservation, their next reservation did not add a late fee to the final price	I originally planed on implementing this feature. As deadline was coming closer and closer, I focused on other areas and ultimatly me and my client forgot until the end of prototype 3.	Set up a button the barber side that would cause the account that missed the reservation, to add a late fee for the next future reservation made. This would be a one time fee and was planned to deter customers from knowingly skipping reservations.
Rounded boxes	All widgets have sqaured off boxes	In the final design, I planned on rounding off the widget boxes to give a more mordern look and also add rounded boxes around the areas of focus. But as AppJar is very limite, I couldn't find a way to implement this.	Using another programming language or software that creates interfaces that allow more control over the program being created. Such as tkinter.
Ordered reservations	When displaying reservations in the table, the reservations are not in order of time	When I created the tables that allowed to display reservation within the interface, I noticed they were out of oder. I wanted to be able to order them but due to time pressure, I had to skip this feature	Before inserting the data into the view reservation tables, I would use a sorting algorithm that would sort the reservations in acending order of time and date. This would have been acheivible if I had more time.
Example photos	When a customer selects a hairstyle, the program doesn't display an example image	I originally planned on giving an example photo of the hairstyle as it is likely the customer doesn't know what each hairstyle looks like. This wasn't implemented for the same reason that I didn't implement upload images, finding a suitable area to store the images. From researching online seemed more complex and would have required much more time.	Once the user selects a hairstyle a popup would display the image of the hairstyle.
Calendar	When viewing a reservation, the customer would be able to see their reservations in a calender style layout	This was goiing to be difficult to implement from scratch as appjar didn't have commands to create calenders. So I decided to the next best option, tables.	In the future, I could attempt to implement the calneder layout but as my client is happy with the current table, I most likely wont make this change.
Edit account	When editing a hairtyle, barber can view its current details	When evaluating the sytem I reliased that the barber editing the hairstyles would benefit the from being able to view its current details as they will be able to compare and make more accurate changes.	Retrives the details of the hairstyle and display the details within a label on the edit hairstyle window.
Cluttered hairstyles window	When the user is on the window to select a hairtyle, they are face with many options that can become overwhelming	As there are many features related to hairstyles, on the select hairstyles page there are many buttons and inputs the user can make. This clutters the screnn much more compared the the other windows.	I could distribute the buttons on other pages to reduce cluttering or combine the features into a single button

Future development:

In the future, with more time and experience on hand I would like to implement more feature.

Encryption:

-I would like to use a hash table to store the hashed passwords and other details. The hashed details would be created by passing the details into, for example, a mid-square hashing function. This would prevent the risk of customer accounts details being leaked if the database were to be hacked.

Payment:

-I would also like to implement a payment system where customers will be able to pay for the haircut during the create the reservation process.

Reminder:

-I would also like to create a reminder system that will be able to send reminders to the users without having to keep the program running.

Online:

-Another improvement would be to move this program online so it can be easily accessible to all users with an internet connection. This would make the program much more accessible for customers.

Edit reservations:

-An improvement would be to allow customers to change certain details regarding their reservation. This could allow them changing the time/date/hairstyle picked

Reset password:

-Another feature will allow customers to reset their account password if they happen to forget. If they get the password input wrong for a certain number of times, send a recovery email to allow access again

Photos:

-Allow customers to upload photos of the hairstyle they want. Be able to see example photos of hairstyles they have picked

Late fee:

-Automatically give a fee if a customer knowingly misses a reservations. If they happen to cancel the reservation beforehand, then no late fee is added to their next reservation they book.

Log out:

-Add a button within the menu window that allows the customers to sign out of the system.

Not in?

-Allow barbers to set a date they are not going to be in. This prevents customers from booking reservation on the date specified for that barber only

Holidays:

-Prevent customers from being able book reservation for national holidays such as bank holiday Monday's and Sundays.

Specific prices:

-Based on the age of the customer, customer will have different prices based on what category they are in. Program checks if they are children, adults, or seniors. Based on the outcome, different prices will be set.

Limitations:

Throughout this project I came across many limitation. The two largest limitations being time and lack of experience/knowledge. Due to the deadline my client had set me, this prevented me from being able to implement features that I could implement with my current skill and experience, features such as the late lee and editing reservation details. But there were also many features that purely due to the lack of skill and experience, I couldn't implement. This would have included the reminder system and other high-level features like encryption.

Python itself didn't restrict me but AppJar, used to build my graphical user interface, did provide many restrictions. AppJar does restrict me from being able to more complex interface as it is just designed to be used as a beginner's way to build an interface. If I had more experience and skills, it would have been better if I used more complex interface builders as this would have allowed to create an interface similar, if not better, than my ones in my final design. But as I am still unexperienced, I still believe using AppJar was the best move. A minor limitation was the adjustment/learning period with AppJar. As I never used AppJar prior, for the first couple weeks development was slower that anticipated but after the period, was able to keep up better with the requirements.

Another limitation was that the development was unpredictable. Throughout development, I noticed that I had to create more variables and functions that I hadn't anticipated in design. This led to me spending even more time planning and creating more functions and variables. Due to this timeless, the took time away from other features, forcing me to either scrap the feature or miss a deadline.

Throughout the project I was able to create a lot of the code by myself, but for the harder functions and features I had to research online to find tutorials that helped me learn how to successfully create the functions/features. These tutorials taught me how to use certain libraries and commands. With this knowledge, I was able to implement the features.

I believe my time management was good most of the time, there were some occasions where I was going to miss the prototype deadline, this forced me to cut out on some features or leave them for later prototype, but in the end, the program was completed within the deadline my client gave with majority of the features implemented.

There were features such as the email system that was beyond the expectation for a A-level student, but the reason on why I spent so much time learning and trying to get it to work was that this feature was high on the list of requirements my clients made. As this was important for my client, I decided to spend a lot more time on this as I knew that this would benefit the system in many ways. This allowed me to do two factor authentication and receipts, but if I had more time, I would have also been able to do password reset with a recovery email.

References:

These are the sites and tutorial I had to use in order to create my code. This taught me how to use certain commands and features and with this information, adapted and implemented the code into my system.

-Email functions: https://itecnote.com/tecnote/python-use-a-variable-inside-html-email/

-SQL statements: https://www.w3schools.com/sql/

-AppJar: http://appjar.info/

-Datetime library: https://www.w3schools.com/python/python_datetime.asp

-Is digit: https://www.w3schools.com/python/ref_string_isdigit.asp

-Is numeric: https://www.w3schools.com/sql/func_sqlserver_isnumeric.asp

-Random library: https://www.w3schools.com/python/ref_random_randint.asp

-Sets data structure: https://www.programiz.com/python-programming/methods/set/difference

-Returning from functions: https://realpython.com/python-return-statement/

Project maintenance:

The final prototype has already accounted for many possible maintenance needed:

-Automatically delete old reservations that prevent the file size becoming too large with unneeded old reservations

-Users are able to already delete reservations if they cannot attend

-Users are already able to create account without developers needing to manually insert the data within database

-Barbers are able to update the hairstyle table so they do not have to manually access the table

-Option box with hairstyles already displays the latest versions of the hairstyle table

-All the code is already commented allowing other developers to understand my work and be able to modify reducing the risk of breaking other parts of the code

-The code is already documented with flowcharts and pseudocode allowing analysis for future requirements can be implemented correctly without me

Maintenance/upgrades that should be carried out in the future:

-Create a feature that allows users to delete their account, or automatically delete after a set period of time. An email can be sent notifying users to be active again if they do not want their account deleted. Prevents the problem where the database file size becomes too large with inactive user account

-Once 2023 is over, change the date option box range to the next year to allow customers to book reservations without them getting the error where the date picked is in the past

-Once 2023 is over, change the date option box range to the next year on the barber select date to allow them to view reservations their customers are making

-Create a feature that allows users to edit their account details. Users may change their contact details and therefore must allow users to update their details in order to have relevant/up-to-date data within the user table

-Implement a feature that allows customers to recover their account if they happen to get the password input wrong a certain number of time. Can be built of my existing email function that can send a recovery email to confirm their identity, then allowing them to update their password. Prevents users having to contact the barbershop in order to change their password

-Implement a feature that allows barbers to also change their passwords without needing to have the developers to access the database and updating manually

-Create a feature that allows the customer to edit their reservations details, such as hairstyle picked.

-Prevent customers from creating reservations on holidays and on days where the barbershop is closed. This will be needed as if a customer does book a reservation on a holiday, they will arrive at a closed barbershop causing frustration in customers

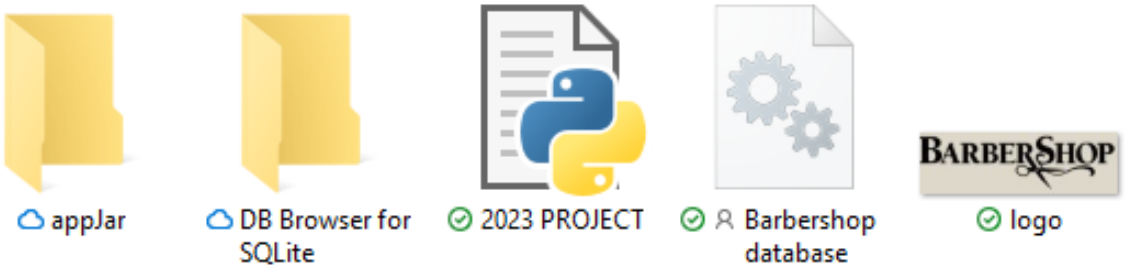
-Create a feature that allows new barbers to create accounts, new employees may join and will need an account to view their reservations

-Implement a feature where instead of the barbers' names being hardcoded into the system, first retrieve all barber accounts and update the option box with all current barbers. This will allow new barbers to be selected and old barbers that left, to be removed. This code should be similar to my fetch hairstyle's function where it updates the list of hairstyles with the latest version of the hairstyle table.

Conclusion:

The measures I had taken to create a system that my client would be happy with to replace their present paper booking system was examined once the project was finished. I have gained a lot of knowledge from this project to help me improve my abilities as a programmer and a problem solver. During the course of the development process, I acquired new skills such as the ability to create a database that can successfully save all the data a user enters into the system and the ability to add new libraries and validate user inputs such as dates entered. I've now able to send emails through python and have been successful at implementing 2 factor authentication and been able to create a graphical user interface, that looks and is fully functional. Overall, I think my project was successful since I was able to design a system that complied with all the specifications.

Folder with files used/needed:



appJar	DB Browser for SQLite	2023 PROJECT	Barbershop database	logo
Folder required to import AppJar, downloaded from AppJar's site (required)	Folder that contain SQLite 3 and DB browser that are needed to view and access the tables	Python file with the finished code (required)	Database file that contains user, hairstyle, reservation table (required)	PNG file of the barbershop logo. Used to insert the logo within the program (required)

Full copy of my code:

```
import sqlite3, smtplib, ssl, random,datetime
from datetime import date

#SQL library to communicate with database
#Email library required to send emails
#Random library for authentication code
#Date library for getting current date

con=sqlite3.connect("Barbershop database.db")
#this creates the database file
```

```
cur=con.cursor()

def create_database():
    cur.execute("""CREATE TABLE "user_table" (
            "username"          TEXT NOT NULL UNIQUE,
            "password"          TEXT NOT NULL,
            "phone_number"              TEXT NOT NULL,
            "email"    TEXT NOT NULL,
            "is_employee"     INTEGER NOT NULL,
            PRIMARY KEY("username")
    )""")
    #creates the user_table with all its fields

    cur.execute("""CREATE TABLE "hairstyle_table" (
            "hair_ID"   INTEGER NOT NULL UNIQUE,
            "hairstyle"           TEXT NOT NULL,
            "price"     REAL NOT NULL,
            "face_shape"       TEXT NOT NULL,
            "face_width"       TEXT NOT NULL,
            "maintainence"    TEXT NOT NULL,
            "hair_length"      TEXT NOT NULL,
            PRIMARY KEY("hair_ID" AUTOINCREMENT)
    )""")
    #creates the hairstyle_table with all its field

    cur.execute("""CREATE TABLE "reservation_table" (
            "reservationID"    INTEGER NOT NULL UNIQUE,
            "barber_name"    TEXT NOT NULL,
            "reservation_time"          REAL NOT NULL,
            "reservation_date"          DATE NOT NULL,
```

```
        "hairstyle"          TEXT NOT NULL,
        "username"           TEXT NOT NULL,
        "message"   TEXT,
        PRIMARY KEY("reservationID" AUTOINCREMENT)
        FOREIGN KEY ("username") REFERENCES user_table,
        FOREIGN KEY ("hairstyle") REFERENCES hairstyle_table,

    )""")
    #creates the reservation_table with all its field

    con.commit()

    file=open("Hairstyles spreadsheet.csv","r") #opens clients files with all the prices / services
    for line in file:  #ensures every record is inserted
        line=line.strip() #removes any whitespace
        hair_ID,hairstyle,price,face_shape,face_width,maintainence,hair_length=line.split(",")
        cur.execute("INSERT INTO hairstyle_table VALUES
(?,?,?,?,?,?,?)",[hair_ID,hairstyle,price,face_shape,face_width,maintainence,hair_length])
        #puts the data from the spreadsheet into the database
    con.commit()

    file=open("Employee detail.csv","r") #opens clients files with their account details
    for line in file:  #ensures every record is inserted
        line=line.strip() #removes any whitespace
        username,password,phone_number,email,is_employee=line.split(",")
        cur.execute("INSERT INTO user_table VALUES
(?,?,?,?,?)",[username,password,phone_number,email,is_employee])
        #puts the data from the spreadsheet into the database
    con.commit()
    print("Everything is fine")
    #gives me feedback telling me its fine
```

```
from appJar import gui

app=gui("User Login")

##################### Main Code ####################

def tempbutton(*arug):
    pass
tempbutton()



def valid_login ():
    user_username=app.getEntry("inputU") #stores username input as a variable
    user_password=app.getEntry("inputP") #stores password input as a variable
    employee=0

    cur.execute("SELECT password FROM user_table WHERE username=? AND is_employee=?",
[user_username,employee])
    #Searches if users input is within database
    tbl_password=cur.fetchone() #Stores one result as password

    cur.execute("SELECT username FROM user_table WHERE username=? AND is_employee=?",
[user_username,employee])
    tbl_username=cur.fetchone() #Stores one result as username

    if tbl_username==None: #Checks if username exists
        app.errorBox("Invalid customer username","Username not found")
    else:
        if tbl_password[0]!=user_password: #Compares if password in table match input
            app.errorBox("Invalid customer password","Incorrect password")
        else:
            app.hide("User Login") #Hides login page
```

```
        send_authenticate_code() #Send code
        app.showSubWindow("Authenticate") #Displays input code page
        app.setLabel("Welcome",f"Welcome, {user_username} to the reservation system.") #Updates the label

def barber_validate():
    app.hide("User Login") #Hides login page
    barber_username=app.getEntry("barberU") #stores username input as a variable
    barber_password=app.getEntry("barberP") #stores password input as a variable
    employee=1 #Variable used to check if details are for a barber account

    cur.execute("SELECT password FROM user_table WHERE username=? AND is_employee=?",[barber_username,employee])
    #Searches for passwords that only are from barbers accounts
    tbl_password=cur.fetchone() #Stores a single password value

    cur.execute("SELECT username FROM user_table WHERE username=? AND is_employee=?",[barber_username,employee])
    #Sarches for usernames that only from barbers accounts
    tbl_username=cur.fetchone() #Stores a single username value

    if tbl_username==None: #Checks if any usernames are found
        app.errorBox("Invalid username","Username not found")
    else:
        if tbl_password[0]!=barber_password: #Checks if tables passwords matches barbers input
            app.errorBox("Invalid password","Incorrect password")
        else:
            app.hideAllSubWindows(useStopFunction=False)
            app.hide("User Login")
            app.showSubWindow("Barber: select date") #Success, onto next window
```

```
def create_account():
    new_Uname=app.getEntry("new_username").strip() #Stores username input as a variable
    new_Pword=app.getEntry("new_password") #Stores password input as a variable
    re_enter_pword=app.getEntry("new_password2") #Stores re enter password as a variable
    new_Pnumber=app.getEntry("new_phone").replace(" ", "").strip() #Stores phone number as a
variable
    new_email=app.getEntry("user_email")

    employee=0 #Creates a new variable to set is_employee field as 0
    valid=validate_details(new_Uname,new_Pword,new_Pnumber,re_enter_pword,new_email)
    #Runs details through function to check validity

    if valid==True: #Checks if details are valid
        cur.execute("INSERT INTO user_table VALUES
(?,?,?,?,?)",[new_Uname,new_Pword,new_Pnumber,new_email,employee,])
        con.commit()
        app.hideAllSubWindows(useStopFunction=False) #Closes all subwindows
        app.show("User Login") #Hides login page

        #Inserts details from user into user table
    else:
        pass #Nothing happens till the user fixes details



def validate_details(new_Uname,new_Pword,new_Pnumber,re_enter_pword,new_email):
    app.hide("User Login")
    #Parameters are taken from the create_account()
    valid=False   #Creates the valid variables and automatically set to false

    for x in new_email:   #Checks if any character have @
        if x=="@":   #Checks indiviual characters for @
```

```
        email_valid=True   #If @ found, set valid to True
        break   #Exits loop if capital letter found
    else:
        next  #Goes to next character
        email_valid=False   #If no @ found, set valid to False

for x in new_Pword:   #Checks if any character are a capital letter
    if x.isupper()==True:   #Checks indiviual characters for capital letters
        capital_valid=True   #If capital letter found, set valid to True
        break   #Exits loop if capital letter found
    else:
        next  #Goes to next character
        capital_valid=False   #If no capital found, set valid to False

for x in new_Pword:    #Checks if any character are an integer
    if x.isdigit()==True:   #Checks indiviual characters for integers
        digit_valid=True   #If integers found, set valid to True
        break    #Exits loop if integers letter found
    else:
        next   #Goes to next character
        digit_valid=False   #If no integers found, set valid to False

cur.execute("SELECT username FROM user_table WHERE username=?", [new_Uname])
#Searches for username that matchs with user input
tbl_username=cur.fetchall() #Stores results as variables
if not tbl_username: #Checks if any results where found
    username_valid=True #If no duplicates, set valid as true
else:
    username_valid=False #If duplicate, set valid as false
```

```
    if len(new_Uname)>0 and len(new_Uname)<12 and username_valid==True: #Checks if username is between 0 to 11 characters and unique
        if len(new_Pword)>6 and len(new_Pword)<16 and digit_valid==True and capital_valid==True and re_enter_pword==new_Pword:
        #Checks if password has a capital and interger and if between 7 to 15 characters and if both entered password match
            if len(new_Pnumber)!=11:   #Checks if phone number has 11 characters
                app.errorBox("Phone number error","Invalid phone number")   #If not 11 characters, error message


            else:
                if len(new_email)>0 and email_valid==True:
                    app.infoBox("Correct details","All details are valid, creating account") #Success
                    valid=True   #Set valid to True, used in create_account()
                else:
                    app.errorBox("Email error","Invalid email address") #If emails incorrect format, error message


        else:
            if re_enter_pword!=new_Pword:   #Checks if re enter password matches password
                app.errorBox("Re enter password error","Passwords do not match")   #If no matches, error message


            else:
                app.errorBox("Password error","Invalid password")   #If password doesnt follow requirments, error message



    else:
        app.errorBox("Username","invalid username or username already taken")   #If username doesnt follow requirements, error message
    return valid #Used in create account function
```

```
def get_times():

    d_pick=app.getDatePicker("date")  #Stores customers date input
    b_pick=app.getOptionBox("Barbers: ")  #Stores cutomers barber input

    cur.execute("SELECT reservation_time FROM reservation_table WHERE barber_name=? AND
reservation_date=?",[b_pick,d_pick])
    #Finds reservations with barber / date input
    results=cur.fetchall()
    #Stores all the results in a 2D list

    bookedTimes=[] #Creates a blank 1D list
    for eachTime in results: #Passes through each reult
        bookedTimes.append(eachTime[0])#Inserts all results into new 1D list

    time_slots=['7:00','8:00','9:00','10:00','11:00','12:00','13:00','14:00','15:00','16:00','17:00']
    #Lists with all availible time slots

    availible_times=[] #Create a blank 1D list
    for index_x in time_slots: #Passes through each index in time slots
        if index_x not in bookedTimes: #Checks if searched index found within booked times
            availible_times.append(index_x) #If not, add into new list
        else:
            pass
    app.changeOptionBox("Time Slots: ",availible_times)
    #Updates option box with availible times

def confirmation():
    final_date=app.getDatePicker("date") #Stores customers date input
    final_time=app.getOptionBox("Time Slots: ") #Stores customers time input
    final_barber=app.getOptionBox("Barbers: ") #Stores customers barber input
```

```
    final_hairstyle=app.getOptionBox("Hairstyles: ") #Stores customers hairstyle input
    cur.execute("SELECT price FROM hairstyle_table WHERE hairstyle=?",[final_hairstyle])
    #Searches for the price of the customers hairstyle
    final_price=cur.fetchone() #Stores the price of a variable

    app.setLabel("final date","Date: "+str(final_date)) #Update date label
    app.setLabel("final time","Time: "+str(final_time)) #Update time label
    app.setLabel("final barber","Barber: "+str(final_barber)) #Update barber label
    app.setLabel("final hairstyle","Hairstyle: "+str(final_hairstyle)) #Update hairstyle label
    app.setLabel("final price","Price: £"+str(final_price[0])) #Update price label

def insert_reservation():
    username=app.getEntry("inputU")#Stores customers username input
    final_date=app.getDatePicker("date") #Stores customers date input
    final_time=app.getOptionBox("Time Slots: ") #Stores customers time input
    final_barber=app.getOptionBox("Barbers: ") #Stores customers barber input
    final_hairstyle=app.getOptionBox("Hairstyles: ") #Stores customers hairstyle input
    cur.execute("SELECT price FROM hairstyle_table WHERE hairstyle=?",[final_hairstyle])
    #Searches for the price of the customers hairstyle
    final_price=cur.fetchone() #Stores the price of a variable
    final_message=app.getEntry("message")

    cur.execute("INSERT INTO reservation_table
(barber_name,reservation_time,reservation_date,hairstyle,username,message)VALUES
(?,?,?,?,?,?)",[final_barber,final_time,final_date,final_hairstyle,username,final_message])
    #Creates a new record in reservation table with all relevent fields filled out
    con.commit()
    app.infoBox("Success!","Reservation has been created successfully, a receipt has been sent to
your email")
    #Lets customer know that the reservation has been created
    send_reciept()
    app.hideSubWindow("Confirmation")
```

```
    app.showSubWindow("Menu")
    #Jumps back to main menu at the end


def generate_RH():
    head_shape=app.getOptionBox("Describe your face shape:       ") #Stores face shape input
    head_width=app.getOptionBox("Describe your face width:       ") #Stores face width input
    u_maintainence=app.getOptionBox("Describe level of hairstyle maintenance:  ") #Stores
maintainance input
    hair_length=app.getOptionBox("Describe your desired length of hair:  ") #Stores hair length input

    cur.execute("SELECT hairstyle FROM hairstyle_table WHERE face_shape=? AND face_width=? AND
maintainence=? AND hair_length=?",[head_shape,head_width,u_maintainence,hair_length])
    #Searches for hairstyles that attributes matches the customers input
    r_hairstyle=cur.fetchall() #Stores search results into 2D list

    if not r_hairstyle: #Checks if any results where found
        hairstyles=fetch_hairstyle()
        app.changeOptionBox("Hairstyles: ",hairstyles)
        #Resets the option box, incase they have already altered the list
        app.errorBox("No matches","No hairstyles found, please select your own")
        #Error message, alerting no matches where found, so it resets the options

    else: #Run if results where found
        recommended_hairstyle=[] #Creats empty list
        for x in r_hairstyle: #Checks every index
            recommended_hairstyle.append(x[0]) #Takes the first value and places in list
        app.changeOptionBox("Hairstyles: ",recommended_hairstyle)

        #Updates availible option boxes with generated hairstyles
        app.infoBox("Matches","Some recommendation have been found!")
        #Alerts user that matches have been found
```

```
def valid_date():
    date=app.getDatePicker("date") #Stores customer date input
    today=date.today() #Stores the current date

    if date<today: #Checks if date picked is in the past
        app.errorBox("Date error","This date is in the past")
        #Error message if in past
    else:
        select_time()
        #Continues the program

def get_reservations():
    app.deleteAllGridRows("view_r") #Resets the table
    username=app.getEntry("inputU") #Stores username within variable
    cur.execute("SELECT reservation_date,reservation_time,barber_name,hairstyle FROM
reservation_table WHERE username=?",[username])
    #Fetches reservation details with the username inputted
    reservations=cur.fetchall() #Stores reservations within variable

    if not reservations:
        #Checks if varible has reservations
        app.showSubWindow("Menu") #If no reservations, output an message
        app.infoBox("No reservations present","You have no current reservations")
        #Message presented to the user
    else:
        app.addTableRows("view_r",reservations)
        #If reservations, add the list of reservations within the table
        app.showSubWindow("Reservations") #Display the reservation window
        app.hideSubWindow("Menu") #Hides the previous window, menu
```

```
def authenticate_code():
    user_code=app.getEntry("code_input") #Stores user input

    if user_code!=str(verify_code): #Compare user code to actual code
        app.errorBox("Verification failed","Authentication code do not match")
        #Error message if code a message dont match
    else:
        app.hideSubWindow("Authenticate")
        app.showSubWindow("Menu")
        #If code match up, it present the menu page


def send_authenticate_code():
    username=app.getEntry("inputU") #Stores the customer username
    cur.execute("SELECT email FROM user_table WHERE username=?",[username])
    #Fetches email within the user table with the customers username
    tbl_email=cur.fetchone() #Stores customer email address

    email_from = 'thebarbershop341@gmail.com' #Stores the sender email
    password = 'dvxyukjbjyaaqmas' #Password that gains access to senders emails
    email_to=str(tbl_email[0]) #Sets the recipient email as customer email

    global verify_code
    #Sets the verification code as global to use in other function
    verify_code=random.randint(100000,999999)
    #Generates the authentication code using random library
    email_content=("Verification code - "+str(verify_code))
    #Email content itself that going to be sent

    context = ssl.create_default_context()
    #Creates a secure connection with server
```

```
    with smtplib.SMTP_SSL("smtp.gmail.com", 465, context=context) as server:
        server.login(email_from, password) #Login to senders email address
        server.sendmail(email_from, email_to,email_content)
        #Sends the email with recipient details and email itself

def send_reciept():
    username=app.getEntry("inputU") #Stores the customer username
    cur.execute("SELECT email FROM user_table WHERE username=?",[username])
    #Fetches email within the user table with the customers username
    tbl_email=cur.fetchone() #Stores customer email address

    final_date=app.getDatePicker("date") #Stores customers date input
    final_time=app.getOptionBox("Time Slots: ") #Stores customers time input
    final_barber=app.getOptionBox("Barbers: ") #Stores customers barber input
    final_hairstyle=app.getOptionBox("Hairstyles: ") #Stores customers hairstyle input
    cur.execute("SELECT price FROM hairstyle_table WHERE hairstyle=?",[final_hairstyle])
    #Searches for the price of the customers hairstyle
    final_price=cur.fetchone() #Stores the price of a variable

    email_from = 'thebarbershop341@gmail.com' #Stores the sender email
    password = 'dvxyukjbjyaaqmas' #Password that gains access to senders emails
    email_to=str(tbl_email[0]) #Sets the recipient email as customer email

    email_content=f"""
    Your reservation details,

    Date - {final_date}
    Time - {final_time}
    Price - {final_price[0]}
    Barber - {final_barber}
    Hairstyle - {final_hairstyle}"""
```

```
    #Email that will be sent to the user

    context = ssl.create_default_context()
    #Creates a secure connection with server
    with smtplib.SMTP_SSL("smtp.gmail.com", 465, context=context) as server:
        server.login(email_from, password) #Login to senders email address
        server.sendmail(email_from, email_to,email_content)
        #Sends the email with recipient details and email itself

def barber_get_reservation():
    app.deleteAllGridRows("B_view") #Resets the table
    barber=app.getEntry("barberU") #Stores barber name within variable
    b_date=app.getDatePicker("b_date")
    cur.execute("SELECT username,reservation_time,hairstyle FROM reservation_table WHERE
barber_name=? and reservation_date=?",[barber,b_date])
    #Fetches reservation details with the barber name and date inputted
    b_reservations=cur.fetchall() #Stores reservations within variable

    if not b_reservations: #Checks if varible has reservations
        app.hideSubWindow("Barber reservations") #Closes this window
        app.showSubWindow("Barber: select date") #Shoes select date window
        app.infoBox("No reservations","No reservation are present on this day")
        #Message presented to the user
    else:
        #If reservations, add the list of reservations within the table
        app.addTableRows("B_view",b_reservations)
        app.showSubWindow("Barber reservations")
        #Displays the barber reservations page

def delete_reservation(row_num):
    delete=app.getTableRow("view_r",row_num) #Stores the reservation being deleted
```

```
    result=app.questionBox("Are you sure?",f"Are you sure you want to delete the reservation at {delete[0]}?", parent="Reservations")
    #Ensures user if wants to delete reservation

    if result==True: #If they want to delete
        app.deleteGridRow("view_r",row_num) #First delete the row on the table
        cur.execute("DELETE FROM reservation_table WHERE reservation_date=? AND reservation_time=?",[delete[0],delete[1]])
        con.commit()
        #Next deletes of the reservation table
        app.infoBox("Deleted",f"Reservation on {delete[0]} has been successfuly deleted")
        #Informs user that it has been deleted

def delete_old_reservations():
    today=date.today() #Stores the current date
    cur.execute("SELECT reservation_date FROM reservation_table")
    #Fetches all reservations currently in the table
    all_reservations=cur.fetchall()

    for each_date in all_reservations: #Checks every index
        year,month,day=each_date[0].split("-")
        #Splits date into 3. The year, month and day
        reservation_date=datetime.date(int(year),int(month),int(day))
        #Converts the string into into datetime data type
        if (reservation_date<today): #Checks if reservation is in the past
            cur.execute("DELETE FROM reservation_table WHERE reservation_date=?",[each_date[0]])
            #If in the past, delete from the table
            con.commit()

def view_more(row_num1):
    view_this=app.getTableRow("B_view",row_num1) #Assigns the record selected to a variable
    b_date=app.getDatePicker("b_date") #Assigns the dates to a variable
```

```
    barber_username=app.getEntry("barberU") #Stores username input within a variable
    cur.execute("SELECT username,phone_number,email FROM user_table WHERE
username=?",[view_this[0]])
    #Fetches all relevent customer details
    user_details=cur.fetchone() #Stores the details within a variable
    cur.execute("SELECT message FROM reservation_table WHERE reservation_date=? AND
reservation_time=? AND barber_name=?",[b_date,view_this[1],barber_username])
    #Fetches the message that the user may have inputted
    message=cur.fetchone() #Stores the message within a variable
    con.commit()

    app.setLabel("extra info",f"""
    Username: {user_details[0]}
    Phone number: {user_details[1]}
    Email: {user_details[2]}
    Message: {message[0]}""")
    #Sets the label with all the inforamtion stored within the variables
    app.showSubWindow("View more")

def edit_details():
    edit_hairstyle=app.getOptionBox("Edit: ") #Retrievs hairstyle name input
    new_price=app.getEntry("New price: ").strip() #Retrievs new price input
    shape=app.getOptionBox("New shape: ") #Stores face shape input
    width=app.getOptionBox("New width: ") #Stores face width input
    maintainence=app.getOptionBox("New maintainence: ") #Stores hair maintainence input
    length=app.getOptionBox("New hair length: ") #Stores length input as variable

    if new_price.replace(".", "").isnumeric()==True: #Validates if input is all integers
        cur.execute("UPDATE hairstyle_table SET
price=?,face_shape=?,face_width=?,maintainence=?,hair_length=? WHERE
hairstyle=?",[new_price,shape,width,maintainence,length,edit_hairstyle])
        con.commit()
```

```
        #If input valid, inserts into the table
        app.infoBox("Successfully updates","New changes have been set")
    else:
        app.errorBox("Format error","Price in incorrect format")
        #If input invalid, error message

def fetch_hairstyle():
    cur.execute("SELECT hairstyle FROM hairstyle_table")
    con.commit()
    #Fetches all  hairstyles from hairstyles table
    hairstyle_tbl=cur.fetchall()

    #Stores hairstyles within table
    hairstyles=[] #Creates an empty list
    for each_hairstyle in hairstyle_tbl:
        #Checks through every index in reservations
        hairstyles.append(each_hairstyle[0])
        #Places hairstyles within reservation
    return hairstyles

def insert_hairstyle():
    name=app.getEntry("Name: ") #Stores name input in varible
    price=app.getEntry("Price: ").strip() #Stoes price input in variable
    shape=app.getOptionBox("Face shape: ") #Stores face shape input
    width=app.getOptionBox("Face width: ") #Stores face width input
    maintainence=app.getOptionBox("Maintainence: ") #Stores hair maintainence input
    length=app.getOptionBox("Hair length: ") #Stores length input as variable

    if price.replace(".", "").isnumeric()==True: #Checks if price valid input
        cur.execute("INSERT INTO hairstyle_table
(hairstyle,price,face_shape,face_width,maintainence,hair_length)VALUES
(?,?,?,?,?,?)",[name,price,shape,width,maintainence,length])
```

```
        con.commit()
        #If valid, inserts into the table
        app.infoBox("Successfully updated","New hairstyle has been added")
        #Message box, informing barber
    else:
        #If invalid, error message
        app.errorBox("Incorrect format 2","Price is in incorrect format")

def confirm_delete_hairstyle():
    delete_hairstyle=app.getOptionBox("Delete: ") #Retrives hairstyle being deleted
    result=app.questionBox("Delte hairstyle?",f"Are you sure you want to delete the hairstyle, {delete_hairstyle}?", parent="Delete hairstyle")
    #Confirms if barber wants to delete hairstyles

    if result==True: #Checks if they confirm or not
        cur.execute("DELETE FROM hairstyle_table WHERE hairstyle=?",[delete_hairstyle])
        #If confirmed, deletes hairstyle from hairstyle table
        con.commit()
        app.infoBox("Successfully deleted",f"Hairstyle, {delete_hairstyle} has been deleted")
        #Message that informs barber


def home():
    app.hideAllSubWindows(useStopFunction=False) #Closes all subwindows
    app.showSubwindow("Menu")
    #Go backs to menu

def edit_hairstyle():
    hairstyles=fetch_hairstyle() #Retrieves all hairstyles
    app.changeOptionBox("Edit: ",hairstyles) #Updates the option box
```

```
    app.hideAllSubWindows(useStopFunction=False) #Closes all subwindows
    app.showSubWindow("Edit hairstyles")

def new_user():
    app.showSubWindow("New user")

def barber_login():
    app.showSubWindow("Barber login")

def reset_hairstyle():
    hairstyles=fetch_hairstyle()
    app.changeOptionBox("Hairstyles: ",hairstyles)



def select_barber():
    app.hideAllSubWindows(useStopFunction=False) #Closes all subwindows
    app.showSubWindow("Select barber")

def select_hairstyle():
    hairstyles=fetch_hairstyle() #Runs function to get hairstyles
    app.changeOptionBox("Hairstyles: ",hairstyles) #Updates list of hairstyles
    app.hideAllSubWindows(useStopFunction=False) #Closes all subwindows
    app.showSubWindow("Select hairstyle")

def select_date():
    final_message=app.getEntry("message") #Stores message
    if len(final_message)>20: #Checks if message is valid
        app.errorBox("Message error","Message to large, must be less than 20 characters")
        #If not, error message
    else:
        app.hideAllSubWindows(useStopFunction=False) #Closes all subwindows
```

```
        app.showSubWindow("Select date")
        #If valid, next window

def select_time():
    get_times() #Retreives all available time/updates option box
    app.hideAllSubWindows(useStopFunction=False) #Closes all subwindows
    app.showSubWindow("Select time")

def confirmation_screen():
    confirmation()
    app.hideAllSubWindows(useStopFunction=False) #Closes all subwindows
    app.showSubWindow("Confirmation")

def recommend_hairstyle():
    app.hideAllSubWindows(useStopFunction=False) #Closes all subwindows
    app.showSubWindow("Recommend hairstyle")

def generate_hairstyle():
    generate_RH() #Generates recommend hairstyle/update options
    app.hideAllSubWindows(useStopFunction=False) #Closes all subwindows
    app.showSubWindow("Select hairstyle")

def home():
    app.hideAllSubWindows(useStopFunction=False) #Closes all subwindows
    app.showSubWindow("Menu")

def b_home():
    app.deleteAllGridRows("B_view") #Resets tables
    app.hideAllSubWindows(useStopFunction=False) #Closes all subwindows
    #Hides all subwindows
    app.showSubWindow("Barber: select date")
```

```
    #Back to select date window

def barber_reservation():
    app.hideAllSubWindows(useStopFunction=False) #Closes all subwindows
    barber_get_reservation() #Retrievs all reservatons



def new_hairstyle():
    app.hideAllSubWindows(useStopFunction=False) #Closes all subwindows
    app.showSubWindow("New hairstyle")

def edit_menu():
    app.hideAllSubWindows(useStopFunction=False) #Closes all subwindows
    app.showSubWindow("Edit menu")

def delete_hairstyle():
    hairstyles=fetch_hairstyle() #Retrieves all hairstyles
    app.changeOptionBox("Delete: ",hairstyles) #Set option box
    app.hideAllSubWindows(useStopFunction=False) #Closes all subwindows
    app.showSubWindow("Delete hairstyle")

def interface():
    ##### LOGIN PAGE #####
    app.setSize("450x250")#Sets size of window
    app.setResizable(False) #Prevents window size change
    app.addImage("logo11","logo.png", compound=None)
    #Adds the barbershop logo
    app.addLabel("Login title","Login",1,1)
    app.setFont(size=13, family="Bell MT", weight="bold")
    #Sets the program font/size/weight
    app.setBg("#ded8ca", override=True)
```

```
#Sets the backround colour
app.getLabelWidget("Login title").config(font=("Bell MT", "16", "bold"))
#Configurates this widget only

app.addLabel("user", "Username:",2,0)
app.addEntry("inputU",2,1) #Input box for username
app.addLabel("password","Password",3,0)
app.addSecretEntry("inputP",3,1) #Input box for password

app.addButton("Login",valid_login,4,3) #Login button
app.addButton("Barber login",barber_login,4,0) #Barber login
app.addButton("New user",new_user,4,1) #Create user button

##### 2 FACTOR AUTHENTICATION #####
app.startSubWindow("Authenticate") #Creates new subwindow
app.setSize("450x250") #Sets size of window
app.setResizable(False) #Fixes size of window
app.setBg("#ded8ca", override=True)
app.addImage("logo10","logo.png", compound=None)

app.addIcon("Enter verification code sent to your email ","padlock-closed", compound="right")
app.addSecretEntry("code_input") #Entry box for user to input their code
app.addButton("Verify",authenticate_code)
#Once clicked run the validate function
app.setEntryDefault("code_input","#######")
app.stopSubWindow() #Stops the window


##### MAIN MENU #####
app.startSubWindow("Menu") #Create a new subwindow
app.setSize("500x350") #Sets size of window
```

```
app.setResizable(False) #Fixes size of window
app.setBg("#ded8ca", override=True)
app.addImage("logo12","logo.png", compound=None)

app.addLabel("Welcome",[]) #Title of window
app.getLabelWidget("Welcome").config(font=("Bell MT", "15", "bold"))

app.addIconButton("Create reservation",select_barber,"book-alt-2", align="top")
#New Reservation button
app.addIconButton("View/delete reservations",get_reservations,"view", align="top")
#View reservation button
app.stopSubWindow() #Stops the window

##### CREATE NEW USER #####
app.startSubWindow("New user") #Creating new subwindow
app.setSize("500x350") #Sets the size of the window
app.setResizable(False) #Fixes the size of window
app.setBg("#ded8ca", override=True)
app.addImage("logo2","logo.png", compound=None)

app.addIcon("Create new account ","register", compound="right") #Title of window
app.addLabel("username1", "Username:",2,0)
app.addEntry("new_username",2,1) #Input box for new username
app.addLabel("password2","Password: ",3,0)
app.addSecretEntry("new_password",3,1) #Input box for new password
app.addLabel("repassword2","Re-enter password: ",4,0)
app.addSecretEntry("new_password2",4,1) #Input box to check if password matches
app.addLabel("phone","Phone number: ",5,0)
app.addEntry("new_phone",5,1) #Input box for phone number
app.addLabel("email","Email address: ",6,0)
app.addEntry("user_email",6,1) #Input box for email address
```

```
app.addButton("Create account",create_account,7,1) #Button leads to validating / inserting
app.stopSubWindow() #Stops the window


##### BARBER LOGIN #####
app.startSubWindow("Barber login") #Creates a new subwindow
app.setSize("400x300") #Sets the size of the window
app.setResizable(True) #Prevents change to size window
app.setBg("#ded8ca", override=True)
app.addImage("logo3","logo.png", compound=None)

app.addIcon("Barber login ","cut", compound="right")
app.addLabel("B_username", "Username: ",2,0)
app.addEntry("barberU",2,1) #Input box for username
app.addLabel("B_password","Password: ",3,0)
app.addSecretEntry("barberP",3,1) #Input box for password

app.addButton("Login to calendar",barber_validate,4,1)
#Button leads validate function

app.stopSubWindow()

##### SELECT BARBER #####
app.startSubWindow("Select barber") #Creates a new subwindow
app.setSize("500x350") #Sets the size of the window
app.setResizable(False) #Prevents change to size window
app.setBg("#ded8ca", override=False)
app.addImage("logo4","logo.png", compound=None)

app.addIcon("Select a barber ","cut", compound="right")
app.addLabelOptionBox("Barbers: ", [" Liam ", " Lucus ", " Dave "," Timothy "])
```

```
#Displays all barber options in option box
app.addIconButton("Hairstyle",select_hairstyle,"arrow-8-right", align="right")
#Confirms selection and moves to hairstyle subwindow
app.addIconButton("home0",home,"home", align=None) #Create the icon
app.stopSubWindow()

##### SELECT HAIRSTYLE #####
app.startSubWindow("Select hairstyle") #Creates a new subwindow
app.setSize("500x350") #Sets the size of the window
app.setResizable(False) #Prevents change to size window
app.setBg("#ded8ca", override=True)
app.addImage("logo5","logo.png", compound=None)
app.addLabel("hairstyle_pick","Select a hairstyle:",)
app.addLabelOptionBox("Hairstyles: ",[])
#Displays all hairstyle options in option box
app.addEntry("message") #Entry box for messages for the barber
app.setEntryDefault("message", "Fade/additional info")
app.addIconButton("Not sure ",recommend_hairstyle,"help", align="right")
app.addIconButton("Reset",reset_hairstyle,"md-reload", align="right")
app.addIconButton("Date",select_date,"arrow-8-right", align="right")
#Confirm hairstyle and moves to date selection
app.addIconButton("home1",home,"home", align=None)
app.stopSubWindow()

##### SELECT DATE #####
app.startSubWindow("Select date") #Creates a new subwindow
app.setSize("500x350") #Sets the size of the window
app.setResizable(False) #Prevents change to size window
app.setBg("#ded8ca", override=True)
app.addImage("logo6","logo.png", compound=None)
```

```
    app.addIcon("Select a date ","calendar-alt-1", compound="right")
    app.addDatePicker("date",2,0) #Select any days of the year
    app.setDatePickerRange("date", 2023, endYear=2023) #Limits the range to 2023
    app.setDatePicker("date")
    app.addIconButton("Time",valid_date,"arrow-8-right", align="right") #Confirms and moves validate date
    app.addIconButton("home2",home,"home", align=None)
    app.stopSubWindow()

    ##### SELECT TIME #####
    app.startSubWindow("Select time") #Creates a new subwindow
    app.setSize("500x350") #Sets the size of the window
    app.setResizable(False) #Prevents change to size window
    app.setBg("#ded8ca", override=True)
    app.addImage("logo7","logo.png", compound=None)

    app.addIcon("Select an available timeslot ","time", compound="right")
    app.addLabelOptionBox("Time Slots: ",[],4,0)#Creates option box for availible times
    app.addIconButton("Confirmation",confirmation_screen,"arrow-8-right", align="right") #Confirms and moves to confirmation
    app.addIconButton("home3",home,"home", align=None)
    app.stopSubWindow()

    ##### CONFIRMATION SCREEN #####
    app.startSubWindow("Confirmation") #Creates a new subwindow
    app.setSize("500x400") #Sets the size of the window
    app.setResizable(False) #Prevents change to size window
    app.setBg("#ded8ca", override=True)
    app.addImage("logo8","logo.png", compound=None)

    app.addLabel("R_details","Reservation details")
    app.getLabelWidget("R_details").config(font=("Bell MT", "16", "bold","underline"))
```

```
    app.addLabel("final date",[]) #Add a blank label for final date
    app.addLabel("final time",[]) #Add a blank label for final time
    app.addLabel("final barber",[]) #Add a blank label for final barber
    app.addLabel("final hairstyle",[]) #Add a blank label for final hairstyle
    app.addLabel("final price",[]) #Add a blank label for final price
    app.addIconButton("Confirm",insert_reservation,"checkbox", align="right")
    #Confirms details and runs function to insert into table
    app.addIconButton("home4",home,"home", align=None)
    app.stopSubWindow()

    ##### RECOMMENDED HAIRSTYLE #####
    app.startSubWindow("Recommend hairstyle") #Creates a new subwindow
    app.setSize("500x350") #Sets the size of the window
    app.setResizable(False) #Prevents change to size window
    app.setBg("#ded8ca", override=True)
    app.addImage("logo9","logo.png", compound=None)

    app.addLabel("R_hairstyle","Answer the questions to generate a recommended hairstyle!",1,0)
    #Lets users what to do
    app.addLabelOptionBox("Describe your face shape:       ",["Round","Square"])
    app.addLabelOptionBox("Describe your face width:       ",["Narrow","Wide"])
    app.addLabelOptionBox("Describe level of hairstyle maintenance:
",["Non/Little","Medium","Lot"])
    app.addLabelOptionBox("Describe your desired length of hair:  ",["Short","Medium","Long"])
    #These provide options for user to select, whilst asking questions
    app.addButton("Generate!",generate_hairstyle)
    #Onces clicked, runs a function to generate hairstyle
    app.addIconButton("home5",home,"home", align=None)
    app.stopSubWindow()

    ##### VIEW RESERVATIONS #####
```

```
    app.startSubWindow("Reservations") #Creates a new subwindow
    app.setSize("550x450") #Sets the size of the window
    app.setResizable(False) #Prevents change to size window
    app.setBg("#ded8ca", override=True)
    app.addImage("logo13","logo.png", compound=None)
    app.addLabel("current_r","All current reservations")
    app.getLabelWidget("current_r").config(font=("Bell MT", "16", "bold","underline"))

    app.addTable("view_r",[["Date", "Time", "Barber", "Hairstyle"]],
action=delete_reservation,actionButton="Delete")
    #Creates table that sets the headers and leaves the rows blank
    app.addIconButton("home6",home,"home", align=None)
    #Home button to lead to menu
    app.stopSubWindow()

    ##### BARBER SELECTS DATE #####
    app.startSubWindow("Barber: select date") #Creates a new subwindow
    app.setSize("350x350") #Sets the size of the window
    app.setResizable(False) #Prevents change to size window
    app.setBg("#ded8ca", override=True)
    app.addImage("logo112","logo.png", compound=None)

    app.addIcon("Search reservation date ","search", compound="right")
    app.addDatePicker("b_date",2,0) #Select any days of the year
    app.setDatePickerRange("b_date", 2023, endYear=2025) #Limits the range to 2025
    app.setDatePicker("b_date")
    app.addIconButton("View reservation",barber_reservation,"arrow-8-right", align="right")
#Confirms and moves to confirmation
    app.addButton("Edit hairstyles",edit_menu)
    app.stopSubWindow()

    ##### BARBER VIEW RESERVATION #####
```

```
    app.startSubWindow("Barber reservations") #Creates a new subwindow
    app.setSize("550x450") #Sets the size of the window
    app.setResizable(False) #Prevents change to size window
    app.setBg("#ded8ca", override=True)
    app.addImage("logo14","logo.png", compound=None)
    app.addIcon("All current reservations","book", compound="right")

    app.addTable("B_view",[["Username","Time","Hairstyle"]],
action=view_more,actionButton="View more")
    #Creates table with it headers defined and records left blank
    app.addIconButton("home16",b_home,"search", align=None)
    #Home button
    app.stopSubWindow()

    ##### BARBER VIEW MORE INFO #####
    app.startSubWindow("View more")
    app.setSize("400x300") #Sets the size of the window
    app.setResizable(False) #Prevents change to size window
    app.setBg("#ded8ca", override=True)
    app.addImage("logo32","logo.png", compound=None)
    app.addLabel("label2","Additional information")

    app.addLabel("extra info",[],2,0)
    #Sets a blank label to set later
    app.stopSubWindow()


    ##### BARBER EDIT HAIRSTYLES #####
    app.startSubWindow("Edit hairstyles")
    app.setSize("550x450") #Sets the size of the window
    app.setResizable(False) #Prevents change to size window
```

```
app.setBg("#ded8ca", override=True)
app.addImage("logo18","logo.png", compound=None)
app.addOptionBox("Edit: ",[])
#Displays updated hairstyle options
app.addLabelEntry("New price: ") #Input new price
app.addLabelOptionBox("New shape: ",["Square","Round"])
app.addLabelOptionBox("New width: ",["Narrow","Wide"])
app.addLabelOptionBox("New maintainence: ",["Non/Little","Medium","Lot"])
app.addLabelOptionBox("New hair length: ",["Short","Medium","Long"])
#Input options for hairstyle attributes
app.addButton("Confirm edit",edit_details)
#Buttons confirms the edit

app.addIconButton("home18",b_home,"search", align=None)
app.stopSubWindow()

##### BARBER NEW HAIRSTYLE #####
app.startSubWindow("New hairstyle")
app.setSize("550x450") #Sets the size of the window
app.setResizable(False) #Prevents change to size window
app.setBg("#ded8ca", override=True)
app.addImage("logo19","logo.png", compound=None)

app.addLabelEntry("Name: ") #Input box for name
app.addLabelEntry("Price: ") #Input box for price
app.addLabelOptionBox("Face shape: ",["Square","Round"])
app.addLabelOptionBox("Face width: ",["Narrow","Wide"])
app.addLabelOptionBox("Maintainence: ",["Non/Little","Medium","Lot"])
app.addLabelOptionBox("Hair length: ",["Short","Medium","Long"])
#Input options for hairstyle attributes
app.addButton("Create hairstyle",insert_hairstyle)
```

```
#Button that creates hairstyle
app.addIconButton("home24",b_home,"search", align=None)
#Home button
app.stopSubWindow()

##### BARBER EDIT MENU #####
app.startSubWindow("Edit menu")
app.setSize("350x300") #Sets the size of the window
app.setResizable(False) #Prevents change to size window
app.setBg("#ded8ca", override=True)
app.addImage("logo20","logo.png", compound=None)

app.addIconButton("Edit ",edit_hairstyle,"file-edit", align="right")
#Leads to change price
app.addIconButton("Add ",new_hairstyle,"file-upload", align="right")
#Leads to create new hairstyle
app.addIconButton("Delete ",delete_hairstyle,"cancel", align="right")
#Leads to delete hairstyle
app.addIconButton("home13",b_home,"search", align=None)
#Home button to select date
app.stopSubWindow()

##### BARBER DELETE HAIRSTYLE #####
app.startSubWindow("Delete hairstyle")
app.setSize("400x350") #Sets the size of the window
app.setResizable(False) #Prevents change to size window
app.setBg("#ded8ca", override=True)
app.addImage("logo27","logo.png", compound=None)
app.addOptionBox("Delete: ",[])
#Displays updated hairstyle options
app.addButton("Delete",confirm_delete_hairstyle)
```

```
    #Button deletes hairstyle
    app.addIconButton("home15",b_home,"search", align=None)
    #Home button to select date
    app.stopSubWindow()

    app.go()


#create_database()
delete_old_reservations()
interface()
```