May 12, 2020

Bradley De’Ath

DEA18001355

Programming and Software Fundamentals

SCDT41 – Assignment 2

Table of Contents

[Requirements 1](#_Toc40127742)

[Prototype Use Case Diagram 2](#_Toc40127743)

[Prototype Class Diagram 2](#_Toc40127744)

[Prototype Testing 3](#_Toc40127745)

[Unit Testing 4](#_Toc40127746)

[Extra Functionality 5](#_Toc40127747)

[End-User Documentation 6](#_Toc40127748)

[Patients 6](#_Toc40127749)

[Logging In 6](#_Toc40127750)

[Menu Navigation 6](#_Toc40127751)

[Information Output 7](#_Toc40127752)

[Staff Members 7](#_Toc40127753)

[Logging In 7](#_Toc40127754)

[Menu Navigation 8](#_Toc40127755)

[Information Output 8](#_Toc40127756)

[Evaluation 8](#_Toc40127757)

[Programming Methodology 8](#_Toc40127758)

[Data Security 9](#_Toc40127759)

[Extra Functionality 10](#_Toc40127760)

[Testing Appendix 10](#_Toc40127761)

[Bibliography 16](#_Toc40127762)

[Source Code Appendix 16](#_Toc40127763)

[Program.cs 16](#_Toc40127764)

[Admin\_Menu.cs 18](#_Toc40127765)

[Administrator.cs 20](#_Toc40127766)

[Appointment.cs 21](#_Toc40127767)

[Appointment\_Note.cs 25](#_Toc40127768)

[Consultation.cs 25](#_Toc40127769)

[Dentist.cs 28](#_Toc40127770)

[Nurses.cs 30](#_Toc40127771)

[Patient.cs 32](#_Toc40127772)

[Patient\_Class.cs 36](#_Toc40127773)

[Patient\_User.cs 39](#_Toc40127774)

[Practice.cs 40](#_Toc40127775)

[Receptionist.cs 46](#_Toc40127776)

[Staff.cs 48](#_Toc40127777)

[Staff\_Menu.cs 49](#_Toc40127778)

[Ticket.cs 53](#_Toc40127779)

[Treatment.cs 55](#_Toc40127780)

# Requirements

There are several Features and Details that need to be included in the MyDentist software for it to meet the requirements of the client, these can be seen below:

|  |  |
| --- | --- |
| **Requirement** | **Description** |
| **Administrators** | |
| Create and Manage Dental Practices | Admin users are able to register and edit Dental Practices to the system based on location, containing all necessary information |
| Create and Manage Staff Credentials | Admin users are able to register and edit Staff and their logins to the system, containing all necessary information |
| **Receptionists** | |
| Create/Manage and View Patient Information | Receptionist users are able to create and edit Patients and their information to the system. This information can then also be viewed as needed |
| Create/Manage and View Appointments | Receptionist users are able to create and edit Appointments containing all necessary information to the system. This information can then also be viewed as needed |
| View Current Treatments | Receptionists are able to view all treatments and related information when needed |
| **Dentists and Nurses** | |
| View Current Treatments | Dentists and Nurses are able to view all treatments and related information when needed |
| View Patient Information | Information on any current patients can be displayed as needed |
| Leave Notes on Appointments | Dentists and Nurses can leave notes on appointments containing any additional information |
| **Practices** | |
| Based on Location | Practices on the system are managed based on their location |
| Contains Treatment Rooms | Each practice can contain a varying number of treatment rooms |
| Several Dentists and Nurses | Each practice can have several dentists and nurses assigned to them |
| **Treatment Rooms** | |
| Potentially Several | The number of treatment rooms can vary between practices |
| Assigned Staff | Each treatment room has a dentist and nurse assigned |
| **Appointments** | |
| Patient Details | Each appointment contains all relevant patient information |
| Date and Time | The date and time for each appointment scheduled is recorded |
| Requested Treatment | The treatment that is to be carried out for each appointment is recorded |
| Allocated Dentist | Each appointment has a dentist assigned |
| **Appointment Notes** | |
| Date and Time | The date and time that the note is created is recorded |
| Content | Each note has text content that can be viewed |
| By Whom | The dentist/nurse that left the note is recorded |
| **Treatments** | |
| Banded | Each treatment is labelled under a ‘band’ |
| Price | The price of each treatment is recorded |
| Details | What each treatment entails are recorded |
| **Patients** | |
| Legal Name | Each patient’s legal name is recorded |
| Address | Each patient’s address is recorded |
| Gender | Each patient’s gender is recorded |
| Contact Number | A valid contact number is recorded for each patient |

# Prototype Use Case Diagram

This Diagram shows how different user types can interact with the application along what processes are accessible based on the staff role.

# Prototype Class Diagram

This Class Diagram shows the connections between the different classes in the MyDentist application along with what variables and methods are used in each.

# Prototype Testing

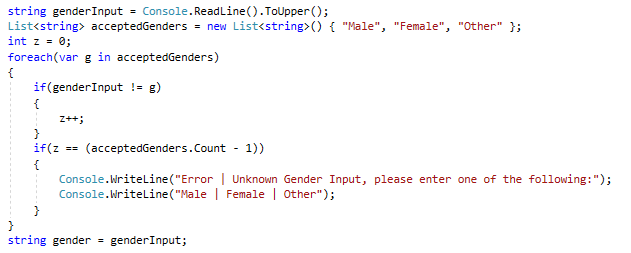
*For the Purpose of Testing, several example accounts have been created for using the system without the need for real credentials, these can be seen below:*

|  |  |  |
| --- | --- | --- |
| **Role** | **Username** | **Password** |
| Admin | admin | password |
| Receptionist | receptionist | password |
| Dentist | dentist | password |
| Nurse | nurse | password |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test No.** | **Test** | **Description** | **Result** | **Implemented Fix (If Applicable)** | **Screenshots** |
| **Administrators** | | | | | |
| **1** | Admin Login | An Admin can log into the system, being directed to the correct menu | Success |  | *Testing Appendix Figures 1.1 & 1.2* |
| **2** | Register Staff | Admins can register new staff members based on their role | Success |  | *Testing Appendix Figures 2.1 & 2.2* |
| **3** | Duplicate Prevention | When creating new staff, validation prevents the same staff member from accidently being created twice | Fail | Error Message alerting Admin of mistake and cancel registration | *Testing Appendix Figures 3.1 & 3.2* |
| **4** | Input Validation | When creating new staff members, the system properly validates inputs | Success |  | *N/A* |
| **5** | Register Practices | Admins can register new Practices based on their location | Success |  | *Testing Appendix Figures 4.1 & 4.2* |
| **6** | Duplicate Prevention | When registering a practice, validation is used to prevent the same practice being added twice | Fail | Error Message alerting Admin of mistake and cancel registration | *Testing Appendix Figures 5.1 & 5.2* |
| **7** | Dentist Validation | When assigning dentists to a practice, proper validation checks if the dentist exists on the system | Success |  | *Testing Appendix Figures 6.1 & 6.2* |
| **8** | Nurse Validation | When assigning nurses to a practice, proper validation checks if the staff exists on the system | Success |  | *Testing Appendix Figures 6.3 & 6.4* |
| **9** | Receptionist Validation | When assigning receptionists to a practice, proper validation checks if the staff exists on the system | Success |  | *Testing Appendix Figures 6.5 & 6.6* |
| **10** | Menu Security | Only Admins can access the tools intended for users of their role | Success |  | *N/A* |
| **Receptionists** | | | | | |
| **11** | Receptionist Login | A Receptionist can log into the system, being directed to the correct menu | Success |  | *Testing Appendix Figures 7.1 & 7.2* |
| **12** | Register Patients | Receptionists can register new patients to the system with all relevant information | Success |  | *Testing Appendix Figures 8.1 & 8.2* |
| **13** | Duplicate Prevention | When registering patients, validation is used to prevent the same patient accidently being added twice | Fail | Error Message alerting user of mistake and cancel creation | *Testing Appendix Figures 9.1 & 9.2* |
| **14** | Input Validation | When registering a patient, the system properly validates inputs | Fail | Error Message alerting user of mistake and cancel creation | *Testing Appendix Figures 10.1 & 10.2* |
| **15** | Create Appointments | Receptionists can create new appointments with all necessary information | Success |  | *Testing Appendix Figures 11.1 & 11.2* |
| **17** | Patient Validation | If a patient does not exist on the system, they cannot be assigned to an appointment | Success |  | *Testing Appendix Figures 12.1 & 12.2* |
| **18** | Dentist Validation | If a dentist does not exist on the system, they cannot be assigned to an appointment | Success |  | *Testing Appendix Figures 12.3 & 12.4* |
| **19** | Practice Validation | If a practice does not exist on the system, it cannot be assigned to an appointment | Success |  | *N/A* |
| **20** | Room Validation | If a room does not exist in the assigned practice, it cannot be assigned to an appointment | Success |  | *Testing Appendix Figures 12.5 & 12.6* |
| **21** | Treatment Validation | If a treatment does not exist on the system, it cannot be assigned to an appointment | Success |  | *Testing Appendix Figures 12.7 & 12.8* |
| **22** | Patient Display | Receptionists can view all patients and related information | Success |  | *Testing Appendix Figures 13.1 & 13.2* |
| **23** | Appointment Display | Receptionists can view all appointments and related information | Success |  | *Testing Appendix Figures 13.3 & 13.4* |
| **24** | Treatment Display | Receptionists can view all current treatments and related information | Success |  | *Testing Appendix Figures 13.5 & 13.6* |
| **25** | Menu Security | Only receptionists can access the tools intended for users of their role | Success |  | *N/A* |
| **Dentists & Nurses** | | | | | |
| **26** | Dentist Login | A Dentist can log into the system, being directed to the correct menu | Success |  | *Testing Appendix Figures 14.1 & 7.2* |
| **27** | Nurse Login | A Nurse can log into the system, being directed to the correct menu | Success |  | *Testing Appendix Figures 14.2 & 7.2* |
| **28** | Create Appointment Notes | These users can leave notes on appointments without altering the appointments themselves | Success |  | *Testing Appendix Figures 15.1 & 15.2* |
| **29** | Duplicate Prevention | Only one note can be attached per appointment, preventing duplicate information | Success |  | *N/A* |
| **30** | Patient Display | These users can view all patients and related information | Success |  | *Testing Appendix Figures 13.1 & 13.2* |
| **31** | Treatment Display | These users can view all current treatments and related information | Success |  | *Testing Appendix Figures 13.5 & 13.6* |
| **32** | Menu Security | Only doctors and nurses can access the tools intended for users of their role | Success |  | *N/A* |
| **Appointment Notes** | | | | | |
| **33** | Time of Creation | The time of note information is automatically recorded and attached to the note | Success |  | *Testing Appendix Figures 15.1 & 15.3* |
| **34** | Author | The user who created the note is automatically recorded and attached to the note | Success |  | *Testing Appendix Figures 15.1 & 15.3* |

## Unit Testing

Along this regular Testing, I used Unit Tests to ensure that inputs by users into fields that are expecting specific results properly handle any values they are not expecting. An example of this can be seen in the screenshot below, where an isolated check for gender values compares the passed input against a stored list of accepted values.

* If the input does not match any of the stored values, the user is presented with an error message containing what went wrong and examples of the accepted values, exiting the object creation.
* If the input does match an accepted value, no further action is taken, and the value is passed on so that it would be used for object creation:

# Extra Functionality

|  |  |
| --- | --- |
| **Feature** | **Description** |
| **Treatments** | |
| New Treatments | New Treatments and necessary information can be created by receptionists for use in the Application |
| **Patients** | |
| Patient Login System | Patients can log into the application with their own user account to access own version of the MyDentist application |
| User Accounts | Each patient has a user account associated with the matching patient object |
| Account Registration | Patients will need to register an account on first use, with a valid PatientID and username/password |
| Request Phone Consultation | Patients can submit tickets to request a phone consultation with a chosen Dentist |
| View Consultation Requests | All submitted requests for consultations can be displayed when needed |
| View Appointments | Patients can view all appointments they have been registered for by a receptionist |
| View Consultations | Patients can view all consultations they have been registered for by a receptionist |
| View Treatments | Patients can view all current treatments and relevant information |
| **Phone Consultations** | |
| Built off Patient Request | Information for new phone consultations can be taken directly off patient-submitted requests |
| Contains all Necessary Info | Phone Consultations will contain all the necessary Information: Time/Date, Patient, Dentist, Phone Number, Additional Notes |
| **Receptionists** | |
| View Consultation Requests | Receptionists can view all patient-submitted requests for phone consultations at their practice |
| Book Phone Consultations | Receptionists can register new phone consultations based on patient-submitted requests |
| View Phone Consultations | Receptionists can view all booked phone consultations at their practice |
| Create Treatments | Receptionists can create new treatments for use in the system |
| **Dentists** | |
| View Consultations | Dentists can view all booked phone consultations and necessary information that they are assigned to |

# End-User Documentation

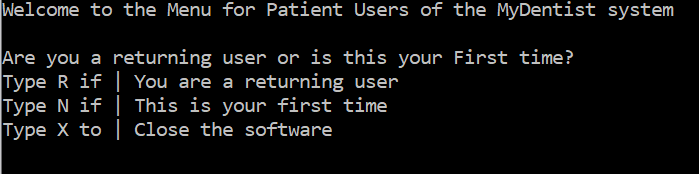
## Patients

### Logging In

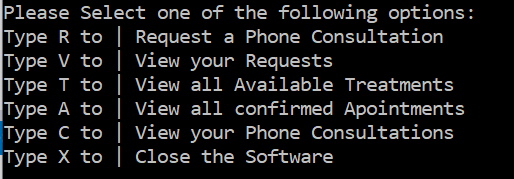
When presented with the initial output upon opening the application, you will be presented with two main options. If you are a patient who is wishing to access the information related to their appointments and phone consultations, then you should type the designated letter for that option and press enter, which will redirect you to the patient-specific login page.

The use of the patient-side of the MyDentist software requires users to have their own account with username and password, due to the need for each user to be able to access information relating specifically to them, that should remain private and not be available to all other users of the software. If you are a user who has already registered an account on the application, then once that option is selected you can enter your chosen username and password. Provided the entered information is valid, you will then be directed to the main Patient Menu for accessing the different options available.

If you are accessing the software for the first time, then you will likely not have an account registered and so will not be able to immediately access the content available. You will need to get a registration ID from the receptionist at your regular dental practice to complete the account creation process. Once the option is selected from the menu, you will be asked to enter the ID, and if it matches you can create your account. Once confirming your desired username and password, the account creation will be successful: logging you in and loading the main patient menu.

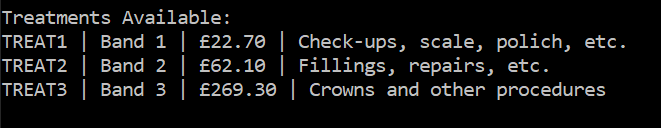


### Menu Navigation

The main patient menu provides access to all options that are available for you to use. All menus in the application are navigated in the same way, with all menu options being associated with a single character than needs to be entered to be able to access it. As can be seen in the screenshot below, based on which letter you enter, a different option will be loaded in the application for you to use. For example, entering ‘R’ when prompted will initiate the process to request a phone consultation with your desired dentist. Once you have completed your desire task of function selected, the application will then automatically relocate you to the menu you had last used, allowing you to easily continue using the software without the need to unnecessarily backtrack.

### Information Output

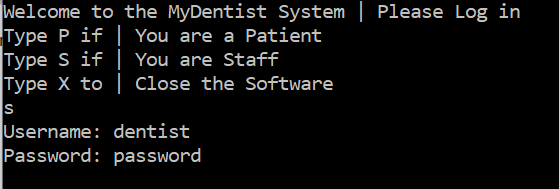
For options where you want content outputted to the screen, such as with displaying all your requested phone consultations, all matching information will be displayed. Starting with the content that is the oldest, all matching information will be written to the screen in order in an easy to read and concise format: allowing you to easily sort through the outputs to find the specific details you are looking for.



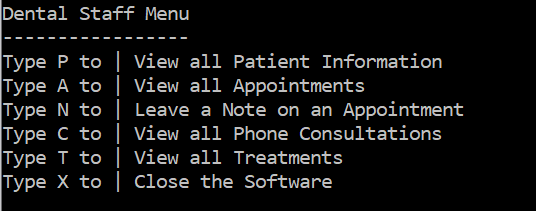
## Staff Members

### Logging In

When presented with the initial output upon opening the application, you will be presented with two main options. If you are a staff member who is intending to manage functions relating to your profession, then you should type the designated character to access the staff login.

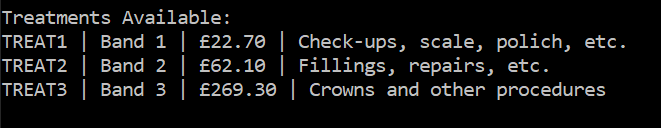
You will then be asked to input your account’s username and password to be able to proceed. Provided that the entered credentials are valid, you will then be redirected to the main menu for staff members of your role, as which options are available vary depending on which staff role you are. *If you are unsure-of or have forgotten your login credentials, you will need to contact your practice’s admin to change the details.*

### Menu Navigation

While the content on the menus vary depending on which role you are (for example, admins will have access to different operations that nurses), the way to navigate between the options remains the same. As can be seen in the screenshot below, the different options on a menu are associated with a unique character, that once entered, will redirect you to the relevant option. Once an option that is selected off a menu is competed, the application will then return you to the menu it was loaded from, allowing for easy navigation without the need for unnecessary backtracking.

### Information Output

There are several options available across the menus that will provide a listed output to the screen for several pieces of information. Depending on your role in the business, the information available to display will vary based on the access level. This can be seen with the ability to output the login credentials such as usernames as passwords. While this option is available in one of the Admin Menus, there is no way for any other type of staff user to access this information.



# Evaluation

## Programming Methodology

For this section of the documentation, the content and design of the MyDentist application’s final solution will be evaluated and justification for the chosen methods and choices leading to the final design of the software.

There are several different styles of programming that can be used when developing an application, all of which have their own methods and features for interacting with and handling data. For the MyDentist software, I chose to use Object-Oriented Programming, due to the way data can be stored as objects for efficient transfer and modification. When data is stored in objects, any properties are bundled together as one to act as a single variable, which can be referenced and modified without the need for meticulously transferring every individual variable that makes up the overall object. Object-Oriented Programming (OOP) allows for a single object to contain several variables of different data types, that can all be referenced and modified uniquely to keep the object usable and up to date.

This helps meet the requirements laid out in the brief due to the need for certain objects in the application to have recent information that is not invalid. This can be seen with the requirement for Receptionist users being able to modify patient details such as phone numbers and addresses, so that in the circumstance that they change: the application is not using redundant data that can cause problems for any processes including it. With properties that need updating within an object, the individual data can be modified without affecting any other information stored there, reducing the risk for methods throughout the system relying on this information breaking due to unexpected changes.

Furthermore, with different objects being needed in several places throughout the application, it can become easy for the system to have too much strain of unnecessary processes passing redundant data in between different methods and classes. This is negated using OOP due to how different properties are compiled together. When an object (for example, a Practice) is called by a method, any information stored within it, such as Location, Number of Rooms, etc. are all passed as well, allowing for several pieces of data to be used whilst only retrieving once. An example of where this is useful in the MyDentist application is when receptionists create and manage appointments. Due to any information relating to existing dentists already being on the system, receptionists only need to reference the dentists themselves to attach any necessary properties to the appointment, such as name and assigned treatment rooms. This same method also applies to any patients and treatments assigned to an appointment, reducing the amount of repeated data entry needed to provide all the necessary information.

Without the use of OOP, the efficiency of data transfer and management is significantly reduced due to the more tedious methods required to refence all required information for a single class. Without objects, any related information would need to be complied into variables such as arrays and lists. Using these variables makes it more difficult to modify existing information efficiently. There are cases where items may need to be remove from the list **(List<T>.Remove(T) Method, n.d.)** and re-added to properly overwrite any redundant data. This can lead to inconsistent ordering of the variables, which has the potential to break methods. Unlike for objects, each variable in a list cannot be refenced by name, but rather by its position, this means that any methods expecting a certain variable at a position, but instead find another, will likely be unable to properly handle the data: causing errors for the user and the application itself. This is the main reasoning behind my choice to use Object-Oriented-Programming over a more simplistic alternative.

## Data Security

One of the main requirements laid out in the brief was for the methods and data used within the application to have property security and only be accessible by those with the proper authority. Using role-dependant staff menus in my software, the ability to access and modify certain content requires users to be the correct role. Upon logging into the system, staff members are presented with different menus depending on the role that they play within the business, with each allowing access to specific actions that can be carried out. For example, while the receptionist menu may contain the option to modify the personal information of the patients on the system, this method cannot be accessed through the Nurse menu, preventing access to sensitive information by those who are unauthorised.

This is an important requirement to meet due to the potential legal ramifications that could occur due to improper handling of the personal information of patients and staff. Some of the information stored about patients on the system, such as their home addresses and phone numbers, are some of the most used when people fall victim to identity theft **(Teraguchi et al., 2004)**. If someone’s data from the MyDentist application were to fall into the wrong hands, the safety of the information’s owner and the reputation of the business are susceptible to sever risk.

## Extra Functionality

On top of meeting the requirements laid out by the clients, I also chose to implement additional features in the MyDentist application to allow for more usability and functionality for users. the largest addition to the software is the inclusion of a patient-only version that allows users to complete actions such as requesting phone consultations with specified dentists and to view their confirmed appointments.

Through having patient accounts build off the existing patient objects, once a username and password have been registered, the patients are able to view and manage content that only pertains to them. This means that information on staff and other patients remain secure and protected from unauthorised access, as content can only be viewed by patients specified in the properties. This can be seen in the application through the method for viewing appointments. Before any appointments are outputted, the current patient’s details are checked against all patients that have appointments, only returning those that match he current user’s details.

This data security can also be seen in the Phone Consultation extra functionality. Like Appointments, Phone Consultations are set up be receptionists and allow for patients to have direct communication with their chosen dentist over the phone. Once a consultation has been set up, after being request by a patient user, the details become available to dentists so that they are aware for any they have been scheduled for. However, dentists are only able to view the information on consultations they themselves are apart of, removing any chance of them seeing personal information belonging to patients that they are not associated with.

# Testing Appendix

This Appendix contains all the figures referenced to in the testing table, for the evidence and source code supporting any test information:

|  |  |
| --- | --- |
|  |  |
| *Figure 1.1: Successful Admin Login* | *Figure 1.2: Source code showing how different logins are handled* |
|  |  |
| *Figure 2.1: The Admin Menu for registering new Staff* | *Figure 2.2: Example source code of abstract staff creation* |
|  |  |
| *Figure 3.1: Fix to prevent duplicate dentists* | *Figure 3.2: Source code for duplicate prevention* |
|  |  |
| *Figure 4.1: Admin view for Practice Creation* | *Figure 4.2: Source Code Snippet of Practice Creation* |
|  |  |
| *Figure 5.1: Fix to prevent duplicate practices* | *Figure 5.2: Source code for duplicate prevention* |
|  |  |
| *Figure 6.1: Admin View if unregistered Dentist is used* | *Figure 6.2: Source Code Snippet for Dentist Validation* |
|  |  |
| *Figure 6.3: Admin view if unregistered Nurse is used* | *Figure 6.4: Source Code snippet for Nurse Validation* |
|  |  |
| *Figure 6.5: Admin view if unregistered Receptionist is used* | *Figure 6.6: Source Code snippet for Receptionist Validation* |
|  |  |
| *Figure 7.1: Successful Receptionist Login* | *Figure 7.2: Source code showing how different logins are handled* |
|  |  |
| *Figure 8.1: Receptionist view for Patient Creation* | *Figure 8.2: Source code Snippet for Patient Creation* |
|  |  |
| *Figure 9.1: Fix to prevent duplicate patients* | *Figure 9.2: Source code for duplicate prevention* |
|  |  |
| *Figure 10.1: Fix to validate inputs* | *Figure 10.2 Source code for input validation* |
|  |  |
| *Figure 11.1: Receptionist View for Appointment Creation* | *Figure 11.2: Source code snippet for Appointment Creation* |
|  |  |
| *Figure 12.1: Error Message for Unknown Patient* | *Figure 12.2: Source code Snippet for Patient Validation* |
|  |  |
| *Figure 12.3: Error Message for Unknown Dentist* | *Figure 12.4: Source code Snippet for Dentist Validation* |
|  |  |
| *Figure 12.5: Error Message for Unknown Treatment* | *Figure 12.6: Source code Snippet for Treatment Validation* |
|  |  |
| *Figure 12.7: Error Message for Unknown Room* | *Figure 12.8: Source code Snippet for Room Validation* |
|  |  |
| *Figure 13.1: Display of all Patients* | *Figure 13.2: Source code Snippet for Patient Display* |
|  |  |
| *Figure 13.3 Display of all Appointments* | *Figure 13.4 Source code for Appointment Display* |
|  |  |
| *Figure 13.5: Display of all Current Treatments* | *Figure 13.6 Source code for Treatment Display* |
|  |  |
| *Figure 14.1: Successful Dentist Login* | *Figure 14.2: Success Nurse Login* |
|  |  |
| *Figure 15.1: User view for added Appointment Note* | *Figure 15.2: Source code Snippet for Creating Appointment Note* |
|  | |
| *Figure 15.3: Source code Snippet for Current Date/Time and User added to Note* | |

# Bibliography

Castagna, G., 2012. *Object-Oriented Programming A Unified Foundation*. Springer Science & Business Media. (pp 80-83).

Docs.microsoft.com. n.d. *List<T>.Remove(T) Method*. [online] Available at: <https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1.remove?view=netframework-4.8> [Accessed 21 April 2020].

Teraguchi, N.C.R.L.Y. and Mitchell, J.C., 2004. Client-side defense against web-based identity theft. *Computer Science Department, Stanford University. Available: http://crypto. stanford. edu/SpoofGuard/webspoof. pdf*.

# Source Code Appendix

## Program.cs

**class** **Program**

{

**public** **static** List<string> loginDetails = **new** List<string>(); *//Storage of inputted login details by user*

**static** **void** Main(string[] args) *//Run at application launch*

{

bool x = **false**;

**do**

{

Console.WriteLine(Environment.NewLine + "Welcome to the MyDentist System | Please Log in"); *//Welcome message to the user*

Console.WriteLine("Type P if | You are a Patient" + Environment.NewLine + "Type S if | You are Staff" + Environment.NewLine + "Type X to | Close the Software");

string userInput = Console.ReadLine().ToUpper();

**switch** (userInput)

{

**case** "P":

Patient\_Class.patientStartup(); *//loads login menu for patients*

**break**;

**case** "S":

staffLogin(); *//loads login menu for staff*

**break**;

**case** "X":

Environment.Exit(1); *//closes the application*

**break**;

}

} **while** (!x);

}

**static** **void** staffLogin() *//method for staff members to log in*

{

bool loginState = **false**;

string loginRole = "";

**do**

{

loginDetails.Clear();

Console.Write("Username: ");

string usernameTemp = Console.ReadLine().ToUpper(); *//reads user's input for their username*

loginDetails.Add(usernameTemp);

Console.Write("Password: ");

string passwordTemp = Console.ReadLine(); *//reads user's input for their password*

loginDetails.Add(passwordTemp); *//adds inputs to the temporany input storage*

loginRole = Staff.checkStaff(loginDetails); *//runs method to check wether inputs are valid, and if so: which role the user is*

**if** (loginRole == "Error") *//if the method returns that the login is unrecognised, the login fails*

{

loginState = **false**;

}

**else** *//otherwise, continue with process*

{

loginState = **true**;

}

} **while** (loginState == **false**); *//run the login request until a recognised login is entered*

List<string> login = **new** List<string>() { loginDetails.ElementAt(0), loginRole }; *//information to pass to the Dental menu*

**switch** (loginRole) *//runs based on which role the user is*

{

**case** "Dentist": *//if they're a dentist, load the dental menu*

Staff\_Menus.dentalMenu(login);

**break**;

**case** "Nurse":

Staff\_Menus.dentalMenu(login); *//if they're a nurse, load the dental menu*

**break**;

**case** "Receptionist": *//if they're a receptionist, load the receptionist menu*

Staff\_Menus.receptionistMenu(loginDetails);

**break**;

**case** "Admin": *//if they're an admin, load the admin-only menu*

Admin\_Menu.adminMenu();

**break**;

}

}

}

## Admin\_Menu.cs

**class** **Admin\_Menu**

{

**static** bool constantMenu = **false**;

**public** **static** **void** adminMenu() *//Menu that is loaded when an admin is loaded in*

{

Console.WriteLine(Environment.NewLine + "Admin Menu" + Environment.NewLine + "------------------------");

Console.WriteLine("Type S to | Manage Staff" + Environment.NewLine + "Type P to | Manage Practices" + Environment.NewLine + "Type X to | Close the Software");

string adminChoice = Console.ReadLine().ToUpper();

**switch** (adminChoice) *//runs different methods based on what the user inputs*

{

**case** "S":

adminStaff(); *//run menu to modify staff credentials*

**break**;

**case** "P":

adminPractice(); *//run menu to modify practices*

**break**;

**case** "X":

Environment.Exit(1); *//close the application*

**break**;

}

}

**protected** **static** **void** adminPractice() *//menu for managing practices*

{

**do**

{

Console.WriteLine(Environment.NewLine + "Practice Management Menu" + Environment.NewLine + "------------------------");

Console.WriteLine("Type A to | Add a Practice" + Environment.NewLine + "Type E to | Edit a Practice" + Environment.NewLine + "Type V to | View all Practices" + Environment.NewLine + "Type B to | Return to Previous Menu");

string adminChoice = Console.ReadLine().ToUpper();

**switch** (adminChoice) *//runs different methods based on the users input*

{

**case** "A":

Practice.addPractice(); *//runs method for adding a new practice*

**break**;

**case** "E":

Practice.editPractice(); *//runs method to edit an existing practice*

**break**;

**case** "V":

Practice.displayPractices(); *//runs method to display a listed view of all practices*

**break**;

**case** "B":

adminMenu(); *//returns to the previous menu*

**break**;

}

} **while** (constantMenu == **false**);

}

**protected** **static** **void** adminStaff() *//menu to manage staff*

{

**do**

{

Console.WriteLine(Environment.NewLine + "Staff Management Menu" + Environment.NewLine + "------------------------");

Console.WriteLine("Type D to | Register a new Dentist" + Environment.NewLine + "Type N to | Register a new Nurse" + Environment.NewLine + "Type R to | Register a new Receptionist" + Environment.NewLine + "Type B to | Return to Previous Menu");

string adminChoice = Console.ReadLine().ToUpper();

**switch** (adminChoice) *//runs different methods based on the user's inputs*

{

**case** "D":

manageDentists(); *//loads menu for managing dentists*

**break**;

**case** "N":

manageNurses(); *//loads menu for managing nurses*

**break**;

**case** "R":

manageReceptionists(); *//loads menu for managing receptionists*

**break**;

**case** "B":

adminMenu(); *//returns to previous menu*

**break**;

}

} **while** (constantMenu == **false**);

}

**protected** **static** bool manageDentists() *//method to manage dentists*

{

**if** (Dentist.addDentist()) { **return** **true**; } *//calls method to manage dentists from the Dentist class*

**return** **false**;

}

**protected** **static** bool manageNurses() *//method to manage nurses*

{

**if** (Nurse.addNurse()) { **return** **true**; } *//calls method to manage nurses from the Nurse class*

**return** **false**;

}

**protected** **static** bool manageReceptionists() { *//method to manage receptionists*

**if** (Receptionist.addReceptionist()) { **return** **true**; } *//calls method to manage receptionists from the Receptionist class*

**return** **false**;

}

}

## Administrator.cs

**class** **Administrator** : Staff

{

string staffID;

string username;

string password;

string name;

**public** Administrator(string staffID, string username, string password, string name) *//Object for an Admin user of the system*

{

**this**.staffID = staffID;

**this**.username = username;

**this**.password = password;

**this**.name = name;

}

**public** string StaffID { **get** => staffID; **set** => staffID = **value**; }

**public** string Username { **get** => username; **set** => username = **value**; }

**public** string Password { **get** => password; **set** => password = **value**; }

**public** string Name { **get** => name; **set** => name = **value**; }

**protected** **static** List<Administrator> allAdmins = **new** List<Administrator> *//List of all Admins on the system*

{

**new** Administrator("00000","ADMIN","password","Admin") *//example admin*

};

**public** **static** bool checkAdmin(List<string> input) *//method to check an admin's login*

{

int y = 0;

**foreach** (var x **in** allAdmins)

{

**if** (input.ElementAt(0) == allAdmins.ElementAt(y).Username & input.ElementAt(1) == allAdmins.ElementAt(y).Password)

{ *//if the login information entered on the main program matches information for an admin user on the system, return that the user is of 'Admin' role*

**return** **true**;

}

**else**

{

y++;

}

}

**return** **false**;

}

}

## Appointment.cs

**class** **Appointment**

{

string appointmentID;

Patient patient;

Dentist dentist;

Treatment treatment;

Practice practice;

string room;

DateTime date;

Appointment\_Note note;

**public** Appointment(string appointmentID, Patient patient, Dentist dentist, Treatment treatment, Practice practice, string room, DateTime date, Appointment\_Note note)

{ *//Object for appointments*

**this**.appointmentID = appointmentID;

**this**.patient = patient;

**this**.dentist = dentist;

**this**.treatment = treatment;

**this**.practice = practice;

**this**.room = room;

**this**.date = date;

**this**.note = note;

}

**public** string AppointmentID { **get** => appointmentID; **set** => appointmentID = **value**; }

**public** Patient Patient{ **get** => patient; **set** => patient = **value**; }

**public** Dentist Dentist { **get** => dentist; **set** => dentist = **value**; }

**public** Treatment Treatment { **get** => treatment; **set** => treatment = **value**; }

**public** Practice Practice { **get** => practice; **set** => practice = **value**; }

**public** string Room { **get** => room; **set** => room = **value**; }

**public** DateTime Date { **get** => date; **set** => date = **value**; }

**public** Appointment\_Note Note { **get** => note; **set** => note = **value**; }

**protected** **static** List<Appointment> allApointments = **new** List<Appointment>() *//List of all appointments stored in the system*

{

**new** Appointment("00000",**new** Patient("67890","EXAMPLE","SURNAME","","","","","","000001111"),**new** Dentist("GHJKL","USERNAME","password","DENTIST","SURNAME"), **new** Treatment("TREAT1","Band 1","22.70","Check-ups, scale, polich, etc."),**new** Practice("ABCDE","TAUNTON",1,**new** Dictionary<string, Dentist>(),**new** Dictionary<string, Nurse>(), **new** Receptionist("","","","","")),"1",DateTime.Now,**null**) *//Example Appointment*

};

**public** **static** bool newAppointment(Practice x) *//Method to create a new appointment*

{

Console.WriteLine(Environment.NewLine + "Enter the Details for the Appointment Below:" + Environment.NewLine);

Console.Write("The ID of the Patient: ");

string patientID = Console.ReadLine().ToUpper();

Patient patient = Patient.checkPatient(patientID); *//checks if inputted ID matches a patient on the system*

**if**(patient == **null**) {

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine(Environment.NewLine + "Error | Patient not found");

Console.ForegroundColor = ConsoleColor.White;

**return** **false**; }

Console.Write("The ID of the Dentist: ");

string dentistID = Console.ReadLine().ToUpper();

Dentist dentist = Dentist.roomDentist(dentistID); *//checks if inputted ID matches a Dentist on the system*

**if**(dentist == **null**) {

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine(Environment.NewLine + "Error | Dentist not found");

Console.ForegroundColor = ConsoleColor.White;

**return** **false**; }

Console.Write("The ID of the Treatment: ");

string treatmentID = Console.ReadLine().ToUpper();

Treatment treatment = Treatment.checkTreatment(treatmentID); *//checks if inputted ID matches a treatment on the system*

**if**(treatment == **null**) {

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine(Environment.NewLine + "Error | Treatment not found");

Console.ForegroundColor = ConsoleColor.White;

**return** **false**; }

Practice pracitce = x;

Console.Write("Designated Room Number: ");

string room = Console.ReadLine().ToUpper();

**if**(int.Parse(room) > x.RoomCount) *//Checks if the inputted room number exists in the current practice*

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine(Environment.NewLine + "Error | Could not find specified Room");

Console.ForegroundColor = ConsoleColor.White;

**return** **false**;

}

Console.Write("Date for the Appointment: ");

DateTime date = Convert.ToDateTime(Console.ReadLine());

Console.Write("Time of the Appointment: ");

DateTime time = Convert.ToDateTime(Console.ReadLine());

date = date.Date.Add(time.TimeOfDay); *//combines inputted date and time into a single variable*

string ID = Guid.NewGuid().ToString(); *//Generates a unique 5 digit ID for the Appointment ID*

char[] idCharacters = ID.Take(5).ToArray();

ID = **new** string(idCharacters).ToUpper();

allApointments.Add(**new** Appointment(ID, patient, dentist, treatment, pracitce, room, date, **new** Appointment\_Note("",**new** Dentist("","","","",""),**new** Nurse("","","","",""), DateTime.Parse("")))); *//adds user's inputted information as a new appointment*

**return** **true**;

}

**public** **static** **void** displayAppointments() *//method to output all appointments on the system*

{

**foreach**(var a **in** allApointments)

{

Console.WriteLine("------------------------------------");

Console.WriteLine("Appointment ID: {0}", a.AppointmentID);

Console.WriteLine("Patient: {0}, {1}", a.Patient.Surname, a.Patient.FirstName);

Console.WriteLine("Patient Contact Number: {0}", a.Patient.PhoneNumber);

Console.WriteLine("Dentist: Dr {0}", a.Dentist.Surname);

Console.WriteLine("Treatment: {0} ({1})", a.Treatment.Name, a.Treatment.Details);

Console.WriteLine("Practice: {0}", a.Practice.Location);

Console.WriteLine("Treatment Room: {0}", a.Room);

Console.WriteLine("Date: {0}", a.Date);

**if** (a.Note != **null**)

{

**if** (a.Note.Dentist == **null**)

{

Console.WriteLine("Addittional Notes: {0} ({1} {2}, {3})", a.Note.Note, a.Note.Nurse.Surname, a.Note.Nurse.FirstName, a.Note.When);

}

**else** if (a.Note.Nurse == **null**)

{

Console.WriteLine("Addittional Notes: {0} ({1} {2}, {3})", a.Note.Note, a.Note.Dentist.Surname, a.Note.Dentist.FirstName, a.Note.When);

}

}

Console.WriteLine("------------------------------------");

}

}

**protected** **static** Appointment checkID(string x) *//method to check if a appointment exists on the system*

{

**foreach**(var a **in** allApointments)

{

**if**(x == a.AppointmentID) { **return** a; }

}

**return** **null**;

}

**public** **static** bool addNote(List<string> login) *//method for dentists/nurses to add a note an existing appointment*

{

Console.Write("Enter the ID of the Appointment to attach a note to: ");

string appointmentID = Console.ReadLine().ToUpper();

Appointment y = checkID(appointmentID);

**if** (y == **null**) { **return** **false**; }

Console.WriteLine("Enter the Note to be attached:");

string note = Console.ReadLine();

**if** (login.ElementAt(1) == "Dentist") *//if the user is a dentist*

{

Dentist dentist = Dentist.loginInformation(login);

**foreach** (var a **in** allApointments)

{

**if** (y.AppointmentID == a.AppointmentID) { a.Note = **new** Appointment\_Note(note, dentist, **null**, DateTime.Now);} *//add note to appointment*

}

}

**else** if (login.ElementAt(1) == "Nurse") *//if the user is a nurse*

{

Nurse nurse = Nurse.loginInformation(login);

**foreach** (var a **in** allApointments)

{

**if** (y.AppointmentID == a.AppointmentID) { a.Note = **new** Appointment\_Note(note, **null**, nurse, DateTime.Now); } *//add note to appointment*

}

}

**return** **true**;

}

**public** **static** **void** patientCheck(Patient\_User user)

{

**foreach**(var a **in** allApointments)

{

**if**(a.Patient == user.Patient)

{

Console.WriteLine("------------------------------------");

Console.WriteLine("Appointment ID: {0}", a.AppointmentID);

Console.WriteLine("Dentist: Dr {0}", a.Dentist.Surname);

Console.WriteLine("Treatment: {0} ({1})", a.Treatment.Name, a.Treatment.Details);

Console.WriteLine("Practice: {0}", a.Practice.Location);

Console.WriteLine("Treatment Room: {0}", a.Room);

Console.WriteLine("Date: {0}", a.Date);

Console.WriteLine("------------------------------------" + Environment.NewLine);

}

}

}

}

## Appointment\_Note.cs

**class** **Appointment\_Note**

{

string note;

Dentist dentist;

Nurse nurse;

DateTime when;

**public** Appointment\_Note(string note, Dentist dentist, Nurse nurse, DateTime when) *//object for appointment notes*

{

**this**.note = note;

**this**.dentist = dentist;

**this**.nurse = nurse;

**this**.when = when;

}

**public** string Note { **get** => note; **set** => note = **value**; }

**public** Dentist Dentist { **get** => dentist; **set** => dentist = **value**; }

**public** Nurse Nurse { **get** => nurse; **set** => nurse = **value**; }

**public** DateTime When { **get** => when; **set** => when = **value**; }

}

## Consultation.cs

**class** **Consultation**

{

**public** **static** List<Consultation> allConsultations = **new** List<Consultation>(); *//list of all phone consultations*

string consultationID;

string phone;

Patient patient;

Dentist dentist;

DateTime date;

Ticket ticket;

Practice practice;

**public** Consultation(string consultationID, string phone, Patient patient, Dentist dentist, DateTime date, Ticket ticket, Practice practice)

{

**this**.consultationID = consultationID;

**this**.phone = phone;

**this**.patient = patient;

**this**.dentist = dentist;

**this**.date = date;

**this**.ticket = ticket;

**this**.practice = practice;

}

**public** string ConsultationID { **get** => consultationID; **set** => consultationID = **value**; }

**public** string Phone { **get** => phone; **set** => phone = **value**; }

**public** Patient Patient { **get** => patient; **set** => patient = **value**; }

**public** Dentist Dentist { **get** => dentist; **set** => dentist = **value**; }

**public** DateTime Date { **get** => date; **set** => date = **value**; }

**public** Ticket Ticket { **get** => ticket; **set** => ticket = **value**; }

**public** Practice Practice { **get** => practice; **set** => practice = **value**; }

**public** **static** bool newConsultation(Receptionist receptionist) *//method to create a new phone consultation appointment*

{

DateTime date;

Practice practice = Practice.checkReceptionist(receptionist);

Ticket.receptionistView(practice);

Console.Write(Environment.NewLine + "Enter the ID of the Request to create a Consultation for: ");

string chosenID = Console.ReadLine().ToUpper();

Ticket ticket = Ticket.checkTicket(chosenID); *//the request ticket for the phone consultation*

**if**(ticket == **null**)

{

Console.WriteLine("Error | Unrecognised ID");

**return** **false**;

}

Console.WriteLine("Requested Date: {0}", ticket.Day , Environment.NewLine);

Console.WriteLine("Fill in the Required details for the Consultation: ");

Console.WriteLine("Patient: {0}, {1}", ticket.Patient.Surname, ticket.Patient.FirstName); *//uses information in patient ticket to autofill information*

Console.WriteLine("Dentist: Dr {0}", ticket.Dentist.Surname);

Console.WriteLine("Contact Details: {0} | {1}", ticket.Patient.PhoneNumber, ticket.Email);

Console.Write("Consultation Date (dd/mm/yyyy): ");

string conDate = Console.ReadLine();

**try**

{

date = DateTime.Parse(conDate);

}

**catch**

{

Console.WriteLine("Error | Incorrect Date Format"); *//value validation for date input*

**return** **false**;

}

Console.Write("Consultation Time (hh/mm): ");

string conTime = Console.ReadLine();

**try**

{

date = date.Date.Add(DateTime.Parse(conTime).TimeOfDay);

}

**catch**

{

Console.WriteLine("Error | Incorrect Time Format");

**return** **false**;

}

string ID = Guid.NewGuid().ToString(); *//Generates a unique 5 digit ID for the Consultation ID*

char[] idCharacters = ID.Take(5).ToArray();

ID = **new** string(idCharacters).ToUpper();

allConsultations.Add(**new** Consultation(ID, ticket.Patient.PhoneNumber, ticket.Patient, ticket.Dentist, date, ticket, practice));

**return** **true**;

}

**public** **static** **void** patientView(Patient\_User u) *//patient method to view all current consultations*

{

**foreach**(var c **in** allConsultations)

{

**if**(c.Patient == u.Patient)

{

Console.WriteLine(Environment.NewLine + "------------------------------------");

Console.WriteLine("Consultation Date: {0}", c.Date.Date);

Console.WriteLine("Consultation Time: {0}", c.Date.TimeOfDay);

Console.WriteLine("Dentist: Dr {0}", c.Dentist.Surname);

Console.WriteLine("Contact Number: {0}", c.Patient.PhoneNumber);

Console.WriteLine("------------------------------------");

}

}

}

**public** **static** **void** receptionistView(Practice p) *//receptionist method to view all current consultations*

{

**foreach**(var c **in** allConsultations)

{

**if**(p == c.Practice)

{

Console.WriteLine(Environment.NewLine + "------------------------------------");

Console.WriteLine("Consultation Date: {0}", c.Date.Date);

Console.WriteLine("Consultation Time: {0}", c.Date.TimeOfDay);

Console.WriteLine("Dentist: Dr {0}", c.Dentist.Surname);

Console.WriteLine("Patient {0}, {1}", c.Patient.Surname, c.Patient.FirstName);

Console.WriteLine("Contact Number: {0}", c.Patient.PhoneNumber);

Console.WriteLine("------------------------------------");

}

}

}

**public** **static** **void** dentistView(Dentist d) *//dentist method to view all current phone consultations*

{

**foreach**(var c **in** allConsultations)

{

**if** (d == c.Dentist)

{

Console.WriteLine(Environment.NewLine + "------------------------------------");

Console.WriteLine("Consultation Date: {0}", c.Date.Date);

Console.WriteLine("Consultation Time: {0}", c.Date.TimeOfDay);

Console.WriteLine("Patient {0}, {1}", c.Patient.Surname, c.Patient.FirstName);

Console.WriteLine("Contact Number: {0}", c.Patient.PhoneNumber);

Console.WriteLine("------------------------------------");

}

}

}

}

## Dentist.cs

**public** **class** **Dentist** : Staff

{

string staffID;

string username;

string password;

string firstName;

string surname;

**public** Dentist(string staffID, string username, string password, string firstName, string surname) *//object for dentists*

{

**this**.staffID = staffID;

**this**.username = username;

**this**.password = password;

**this**.firstName = firstName;

**this**.surname = surname;

}

**public** string StaffID { **get** => staffID; **set** => staffID = **value**; }

**public** string Username { **get** => username; **set** => username = **value**; }

**public** string Password { **get** => password; **set** => password = **value**; }

**public** string FirstName { **get** => firstName; **set** => firstName = **value**; }

**public** string Surname { **get** => surname; **set** => surname = **value**; }

**protected** **static** List<Dentist> allDentists = **new** List<Dentist>{

**new** Dentist("12345", "DENTIST", "password", "GENERIC", "NAME") *//example dentist*

};

**public** **static** bool checkDentist(List<string> input) *//method to check if dentist exists*

{

int y = 0;

**foreach** (var x **in** allDentists) *// checks if inputted login information matches a dentist on the system*

{

**if** (input.ElementAt(0) == allDentists.ElementAt(y).Username & input.ElementAt(1) == allDentists.ElementAt(y).Password)

{

**return** **true**;

}

**else**

{

y++;

}

}

**return** **false**;

}

**public** **static** Dentist roomDentist(string inputID) *//method to check if dentist exsits on the system*

{

**foreach**(var d **in** allDentists)

{

**if**(inputID == d.StaffID)

{

**return** d; *//return the dentist object for that ID*

}

}

**return** **null**;

}

**public** **static** bool addDentist() *//method to add new dentist*

{

List<string> input = addStaff();

**foreach**(var d **in** allDentists)

{

**if**(input[1] == d.FirstName || input[4] == d.Username)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine(Environment.NewLine + "Error | Dentist Already Exists");

Console.ForegroundColor = ConsoleColor.White;

**return** **false**;

}

}

allDentists.Add(**new** Dentist(input[0], input[3], input[4], input[1], input[2])); *//adds new dentist based on inputted information*

*//Dentist creation Successful*

**return** **true**;

}

**public** **static** Dentist loginInformation(List<string> loginDetails) *//checks dentist login information*

{

**foreach**(var d **in** allDentists)

{

**if**(loginDetails.ElementAt(0) == d.Username)

{

**return** d;

}

}

**return** **null**;

}

**public** **static** List<Dentist> returnAll()

{

**return** allDentists;

}

}

## Nurses.cs

**public** **class** **Nurse** : Staff

{

string staffID;

string username;

string password;

string firstName;

string surname;

**public** Nurse(string staffID, string username, string password, string firstName, string surname) *//object for each nurse on the system*

{

**this**.staffID = staffID;

**this**.username = username;

**this**.password = password;

**this**.firstName = firstName;

**this**.surname = surname;

}

**public** string StaffID { **get** => staffID; **set** => staffID = **value**; }

**public** string Username { **get** => username; **set** => username = **value**; }

**public** string Password { **get** => password; **set** => password = **value**; }

**public** string FirstName { **get** => firstName; **set** => firstName = **value**; }

**public** string Surname { **get** => surname; **set** => surname = **value**; }

**protected** **static** List<Nurse> allNurses = **new** List<Nurse> *//list of all nurses*

{

**new** Nurse("67890","NURSE","password","GENERIC","NAME")

};

**public** **static** bool checkNurse(List<string> input) *//method to check if nurse exists on the system*

{

int y = 0;

**foreach** (var x **in** allNurses)

{

**if** (input.ElementAt(0) == allNurses.ElementAt(y).Username & input.ElementAt(1) == allNurses.ElementAt(y).Password) *//if the inputted login matches a nurse*

{

**return** **true**;

}

**else**

{

y++;

}

}

**return** **false**;

}

**public** **static** Nurse roomNurse(string inputID) *//method to check if inputted ID matches that of a nurse*

{

**foreach** (var n **in** allNurses)

{

**if** (inputID == n.StaffID)

{

**return** n;

}

}

**return** **null**;

}

**public** **static** bool addNurse() *//method to add a new nurse*

{

List<string> input = addStaff();

**foreach** (var n **in** allNurses)

{

**if** (input[1] == n.FirstName & input[4] == n.Username)

{

*//Nurse already exists*

**return** **false**;

}

}

allNurses.Add(**new** Nurse(input[0], input[3], input[4], input[1], input[2])); *//create new nurse with inputted information*

*//Nurse creation Successful*

**return** **true**;

}

**public** **static** Nurse loginInformation(List<string> loginDetails)*//checks login information against nurses on the sustem*

{

**foreach** (var d **in** allNurses)

{

**if** (loginDetails.ElementAt(0) == d.Username)

{

**return** d; *//returns the matching nurse*

}

}

**return** **null**;

}

}

## Patient.cs

**class** **Patient**

{

**public** **static** List<Patient> allPatients = **new** List<Patient>() { **new** Patient("12345", "EXAMPLE", "PATIENT", "", "", "", "", "", "075192835")};

string patientID;

string firstName;

string surname;

string houseNumber;

string street;

string town;

string postcode;

string gender;

string phoneNumber;

**public** Patient(string patientID, string firstName, string surname, string houseNumber, string street, string town, string postcode, string gender, string phoneNumber)

{ *//object for all patients*

**this**.patientID = patientID;

**this**.firstName = firstName;

**this**.surname = surname;

**this**.houseNumber = houseNumber;

**this**.street = street;

**this**.town = town;

**this**.postcode = postcode;

**this**.gender = gender;

**this**.phoneNumber = phoneNumber;

}

**public** string PatientID { **get** => patientID; **set** => patientID = **value**; }

**public** string FirstName { **get** => firstName; **set** => firstName = **value**; }

**public** string Surname { **get** => surname; **set** => surname = **value**; }

**public** string HouseNumber { **get** => houseNumber; **set** => houseNumber = **value**; }

**public** string Street { **get** => street; **set** => street = **value**; }

**public** string Town { **get** => town; **set** => town = **value**; }

**public** string Postcode { **get** => postcode; **set** => postcode = **value**; }

**public** string Gender { **get** => gender; **set** => gender = **value**; }

**public** string PhoneNumber { **get** => phoneNumber; **set** => phoneNumber = **value**; }

**public** **static** bool addPatient() *//method to add new patient to the system*

{

string patientID;

string firstName;

string surname;

string houseNumber;

string street;

string town;

string postcode;

string gender;

string phoneNumber;

Console.WriteLine("Enter the Patient's Details Below:" + Environment.NewLine);

Console.Write("First Name: ");

firstName = Console.ReadLine().ToUpper();

Console.Write("Surname: ");

surname = Console.ReadLine().ToUpper();

Console.Write("Address | House Number: ");

houseNumber = Console.ReadLine().ToUpper();

Console.Write("Address | Street: ");

street = Console.ReadLine().ToUpper();

Console.Write("Address | Town: ");

town = Console.ReadLine().ToUpper();

Console.Write("Address | Postcode: ");

postcode = Console.ReadLine().ToUpper();

Console.Write("Gender : ");

gender = Console.ReadLine().ToUpper();

List<string> genderOptions = **new** List<string>() { "male", "female", "other" };

int z = 0;

**foreach** (var g **in** genderOptions)

{

**if**(gender != g)

{

z++;

}

**if**(z == 2)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("Error | Unrecognised Input");

Console.ForegroundColor = ConsoleColor.White;

**return** **false**;

}

}

Console.Write("Phone Number: ");

phoneNumber = Console.ReadLine().ToUpper();

**foreach**(var p **in** allPatients)

{

**if**(firstName == p.FirstName & surname == p.Surname & postcode == p.Postcode)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine(Environment.NewLine + "Error | Patient Already Exists");

Console.ForegroundColor = ConsoleColor.White;

**return** **false**;

}

}

patientID = Guid.NewGuid().ToString(); *//Generates a unique 5 digit ID for the patient ID*

char[] idCharacters = patientID.Take(5).ToArray();

patientID = **new** string(idCharacters).ToUpper();

allPatients.Add(**new** Patient(patientID, firstName, surname, houseNumber, street, town, postcode, gender, phoneNumber));

**return** **true**; *//add new patient based on the inputted information*

}

**public** **static** **void** viewPatients() *//method to display all patients on the system*

{

**foreach**(var p **in** allPatients)

{

Console.WriteLine(Environment.NewLine + "{0} | {1} {2} | {3} | {4}, {5}, {6}, {7} | {8}", p.PatientID, p.FirstName, p.Surname, p.Gender, p.HouseNumber, p.Street, p.Town, p.Postcode, p.PhoneNumber);

}

}

**public** **static** bool editPatient() *//method to edit existing patient's information*

{

Console.Write("Enter the ID of the Patient to Change: ");

string changeID = Console.ReadLine().ToUpper();

**foreach**(var p **in** allPatients)

{

**if** (changeID == p.patientID)

{

Console.WriteLine("------------------------------------");

Console.WriteLine("{0} {1}: Current Information" + Environment.NewLine, p.FirstName, p.surname);

Console.WriteLine("Patient ID: {0}", p.PatientID);

Console.WriteLine("Gender: {0}", p.Gender);

Console.WriteLine("Address: {0}, {1}, {2}, {3}", p.HouseNumber, p.Street, p.Town, p.Postcode);

Console.WriteLine("Phone Number: {0}", p.PhoneNumber);

Console.WriteLine("------------------------------------");

Console.WriteLine(Environment.NewLine + "Enter the New Details Below:");

Console.Write("First Name: ");

string firstName = Console.ReadLine().ToUpper();

Console.Write("Surname: ");

string surname = Console.ReadLine().ToUpper();

Console.Write("Address | House Number: ");

string houseNumber = Console.ReadLine().ToUpper();

Console.Write("Address | Street: ");

string street = Console.ReadLine().ToUpper();

Console.Write("Address | Town: ");

string town = Console.ReadLine().ToUpper();

Console.Write("Address | Postcode: ");

string postcode = Console.ReadLine().ToUpper();

Console.Write("Gender : ");

string gender = Console.ReadLine().ToUpper();

Console.Write("Phone Number: ");

string phoneNumber = Console.ReadLine().ToUpper();

p.FirstName = firstName; *//overwrites properties of selected patient*

p.Surname = surname;

p.HouseNumber = houseNumber;

p.Street = street;

p.Town = town;

p.Postcode = postcode;

p.Gender = gender;

p.PhoneNumber = phoneNumber;

}

}

**return** **true**;

}

**public** **static** Patient checkPatient(string inputID) *//method to check if ID matches that of a patient*

{

**foreach**(var i **in** allPatients)

{

**if**(inputID == i.PatientID)

{

**return** i; *//return matching patient*

}

}

**return** **null**;

}

**public** **static** Patient patientUser(string inputID) *//Validates if a patient user's enterred referal ID matches that of an existing patient on the system*

{

**foreach**(var p **in** allPatients)

{

**if**(inputID == p.patientID)

{

**return** p;

}

}

Console.WriteLine("Error | Entered ID is not recognised"); *//return error message*

**return** **null**;

}

}

## Patient\_Class.cs

**class** **Patient\_Class**

{

**public** **static** bool constantMenu = **false**;

**public** **static** **void** patientStartup() *//Login Menu for Patient Users*

{

**do**

{

Console.WriteLine("Welcome to the Menu for Patient Users of the MyDentist system" + Environment.NewLine);

Console.WriteLine("Are you a returning user or is this your First time?");

Console.WriteLine("Type R if | You are a returning user" + Environment.NewLine + "Type N if | This is your first time" + Environment.NewLine + "Type X to | Close the software");

string userChoice = Console.ReadLine().ToUpper();

**switch** (userChoice)

{

**case** "R":

userLogin(); *//Method for if user has logged into the system before*

**break**;

**case** "N":

newUser(); *//Method or if this is the user's first time on the system*

**break**;

**case** "X":

Environment.Exit(1); *//closes the application*

**break**;

}

} **while** (!constantMenu);

}

**protected** **static** **void** newUser() *//Method to create new user*

{

bool usernameMatch = **false**;

bool passwordMatch = **false**;

string username = "";

string password = "";

Console.Write("Please Enter your Provided User ID: ");

string x = Console.ReadLine().ToUpper();

Patient systemPatient = Patient.patientUser(x);

**do**

{

Console.Write("Please enter your desired Username: ");

string a = Console.ReadLine().ToUpper();

Console.Write("Please Re-enter the Username: ");

string b = Console.ReadLine().ToUpper();

**if** (a == b) { usernameMatch = **true**; }

**else** { Console.WriteLine("Error | Usernames did not Match"); }

**if** (!Patient\_User.checkUser(a))

{

Console.WriteLine("Error | Username already in Use");

}

**else** if (Patient\_User.checkUser(a))

{

usernameMatch = **true**;

username = a;

}

}**while**(usernameMatch == **false**);

**do**

{

Console.Write("Please enter a Password: ");

string c = Console.ReadLine();

Console.Write("Please Confirm your Password: ");

string d = Console.ReadLine();

**if**(c == d)

{

passwordMatch = **true**;

password = d;

}

**else**

{

Console.WriteLine("Error | Passwords did not match");

}

} **while** (passwordMatch == **false**);

Patient\_User y = **new** Patient\_User(systemPatient, username, password);

Patient\_User.addUser(y);

userMenu(y); *//loads user into main login menu*

}

**protected** **static** **void** userLogin() *//method for exisiting users to log in*

{

**do**

{

Console.WriteLine("Please enter your Username: ");

string username = Console.ReadLine().ToUpper();

Console.WriteLine("Please enter your Password: ");

string password = Console.ReadLine();

Patient\_User currentUser = Patient\_User.loginCheck(**new** List<string>() { username, password });

**if** (currentUser != **null**) { userMenu(currentUser); }

} **while** (!constantMenu);

}

**protected** **static** **void** userMenu(Patient\_User user) *//Main menu for patient users*

{

**do**

{

Console.WriteLine(Environment.NewLine + "Please Select one of the following options:");

Console.WriteLine("Type R to | Request a Phone Consultation" + Environment.NewLine + "Type V to | View your Requests" + Environment.NewLine + "Type T to | View all Available Treatments");

Console.WriteLine("Type A to | View all confirmed Apointments" + Environment.NewLine + "Type C to | View your Phone Consultations" + Environment.NewLine + "Type X to | Close the Software");

string userChoice = Console.ReadLine().ToUpper();

**switch** (userChoice)

{

**case** "R":

Ticket.newTicket(user); *//method to create request ticket for phone consultation*

**break**;

**case** "V":

Ticket.viewTickets(user); *//method to view all active request tickets*

**break**;

**case** "T":

Treatment.treatmentDisplay(); *//method to display all available treatments*

**break**;

**case** "A":

Appointment.patientCheck(user); *//method to view all current appointments*

**break**;

**case** "C":

Consultation.patientView(user); *//method to view all confirmed phone consultations*

**break**;

**case** "X":

Environment.Exit(1); *//close the application*

**break**;

}

} **while** (!constantMenu);

}

}

## Patient\_User.cs

**class** **Patient\_User**

{

Patient patient;

string username;

string password;

**public** **static** List<Patient\_User> allUsers = **new** List<Patient\_User>()

{ **new** Patient\_User(**new** Patient("12345","Bradley","De'Ath","","","","","",""),"USERNAME","password")

};

**public** Patient\_User(Patient patient, string username, string password) *//New Patient Object*

{

**this**.patient = patient;

**this**.username = username;

**this**.password = password;

}

**public** Patient Patient { **get** => patient; **set** => patient = **value**; }

**public** string Username { **get** => username; **set** => username = **value**; }

**public** string Password { **get** => password; **set** => password = **value**; }

**public** **static** bool checkUser(string input) *//Validates if user details already exisit on system*

{

**foreach**(var u **in** allUsers)

{

**if**(input == u.Username) { **return** **false**; }

}

**return** **true**;

}

**public** **static** **void** addUser(Patient\_User newUser) *//Method for creating new user login*

{

allUsers.Add(newUser);

Console.WriteLine("Account Successfully Created");

}

**public** **static** Patient\_User loginCheck(List<string> x) *//checks login credentials of user*

{

**foreach**(var l **in** allUsers)

{

**if**(x.ElementAt(0) == l.Username & x.ElementAt(1) == l.Password)

{

**return** l;

}

}

Console.WriteLine("Error | Incorrect Login, Please Try Again"); *//return error*

**return** **null**;

}

}

## Practice.cs

**class** **Practice**

{

string practiceID;

string location;

int roomCount;

Dictionary<string, Dentist> roomDentist = **new** Dictionary<string, Dentist>();

Dictionary<string, Nurse> roomNurse = **new** Dictionary<string, Nurse>();

Receptionist receptionist;

**public** Practice(string practiceID, string location, int roomCount, Dictionary<string, Dentist> roomDentist,Dictionary<string,Nurse> roomNurse,Receptionist receptionist)

{ *//object for practices*

**this**.practiceID = practiceID;

**this**.location = location;

**this**.roomCount = roomCount;

**this**.roomDentist = roomDentist;

**this**.roomNurse = roomNurse;

**this**.receptionist = receptionist;

}

**public** string PracticeID { **get** => practiceID; **set** => practiceID = **value**; }

**public** string Location { **get** => location; **set** => location = **value**; }

**public** int RoomCount { **get** => roomCount; **set** => roomCount = **value**; }

**public** Dictionary<string,Dentist> RoomDentist { **get** => roomDentist; **set** => roomDentist = **value**; }

**public** Dictionary<string,Nurse> RoomNurse { **get** => roomNurse; **set** => roomNurse = **value**; }

**public** Receptionist Receptionist { **get** => receptionist; **set** => receptionist = **value**; }

**protected** **static** List<Practice> allPracticies = **new** List<Practice>(); *//List of all practices*

**public** **static** bool addPractice() *//method to add new practice*

{

Dictionary<string, Dentist> roomDentist = **new** Dictionary<string, Dentist>();

Dictionary<string, Nurse> roomNurse = **new** Dictionary<string, Nurse>();

string[] rooms = **new** string[10] *//maximum of 10 rooms per practice*

{

"Room 1","Room 2","Room 3","Room 4","Room 5","Room 6","Room 7","Room 8","Room 9","Room 10"

};

int x = 0;

List<string> currentRooms = **new** List<string>();

Console.WriteLine(Environment.NewLine + "Enter the Practice's Details Below:" + Environment.NewLine);

Console.Write("Location: ");

string location = Console.ReadLine().ToUpper();

Console.Write("Practice Room Count (Min: 1, Max: 10) :");

int roomCount = int.Parse(Console.ReadLine()); *//number of rooms*

Console.Write("Receptionist's Staff ID: ");

string receptionistID = Console.ReadLine().ToUpper(); *//checks if receptionist exists on the system*

Receptionist check3 = Receptionist.roomReceptionist(receptionistID);

**if** (check3 == **null**)

{

Console.WriteLine("Receptionist Error");

**return** **false**;

}

**do**

{

currentRooms.Add(rooms.ElementAt(x));

x++;

}**while**(x <= roomCount-1);

**foreach**(var r **in** currentRooms) *//for every room in the dental practice*

{

Console.WriteLine(r);

Console.Write("Assigned Dentist's Staff ID: ");

string dentistID = Console.ReadLine().ToUpper();

Dentist check = Dentist.roomDentist(dentistID);

**if**(check == **null**)

{

Console.WriteLine("Dentist Error");

**return** **false**;

}

**else**

{

roomDentist.Add(r, check); *//assigns dentist to current room*

}

Console.Write("Assigned Nurse's Staff ID: ");

string nurseID = Console.ReadLine().ToUpper();

Nurse check2 = Nurse.roomNurse(nurseID);

**if**(check2 == **null**)

{

Console.WriteLine("Nurse Error");

**return** **false**;

}

**else**

{

roomNurse.Add(r, check2); *//assigns nurse to current room*

}

}

string practiceID = Guid.NewGuid().ToString(); *//Generates a unique 5 digit ID for the Practice ID*

char[] idCharacters = practiceID.Take(5).ToArray();

practiceID = **new** string(idCharacters).ToUpper();

Practice newPractice = **new** Practice(practiceID, location, roomCount, roomDentist, roomNurse, check3); *//creates new practice with inputted information*

**foreach** (var p **in** allPracticies) *//checks if practice already exists*

{

**if**(p.Location == newPractice.Location)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine(Environment.NewLine + "Error | Practice Already Exists");

Console.ForegroundColor = ConsoleColor.White;

**return** **false**;

}

}

allPracticies.Add(newPractice);

**return** **true**;

}

**public** **static** bool displayPractices() *//method to output all practices on the system*

{

**foreach**(var p **in** allPracticies)

{

Console.WriteLine("------------------------------------");

Console.WriteLine("Practice ID: {0}", p.PracticeID);

Console.WriteLine("Location: {0}", p.Location);

Console.WriteLine("Number of Rooms: {0}", p.roomCount);

Console.WriteLine("Receptionist: {0}, {1}", p.Receptionist.Surname, p.Receptionist.FirstName + Environment.NewLine);

var x = 0;

**foreach** (var d **in** p.roomDentist) *//displays information for each room in the practice*

{

Console.ForegroundColor = ConsoleColor.Cyan;

Console.WriteLine(Environment.NewLine + "{0}:", d.Key);

Console.ForegroundColor = ConsoleColor.White;

Console.WriteLine("Dentist: {0}, {1}", d.Value.Surname, d.Value.FirstName);

Console.WriteLine("Nurse: {0}, {1}", p.roomNurse.ElementAt(x).Value.Surname, p.roomNurse.ElementAt(x).Value.FirstName);

x++;

}

Console.WriteLine("------------------------------------");

}

Console.ReadLine();

**return** **true**;

}

**public** **static** bool editPractice() *//method to edit practice properties*

{

Console.Write("Enter the ID of the Practice to Change: ");

string changeID = Console.ReadLine().ToUpper();

**foreach**(var i **in** allPracticies)

{

**if** (changeID == i.practiceID)

{

Console.WriteLine("------------------------------------");

Console.WriteLine("{0} Practice: Current Information", i.Location + Environment.NewLine);

Console.WriteLine("Practice ID: {0}", i.PracticeID);

Console.WriteLine("Location: {0}", i.Location);

Console.WriteLine("Number of Rooms: {0}", i.roomCount);

Console.WriteLine("Receptionist: {0}, {1}", i.Receptionist.Surname, i.Receptionist.FirstName + Environment.NewLine);

var x = 0;

**foreach** (var d **in** i.roomDentist)

{

Console.ForegroundColor = ConsoleColor.Cyan;

Console.WriteLine(Environment.NewLine + "{0}:", d.Key);

Console.ForegroundColor = ConsoleColor.White;

Console.WriteLine("Dentist: {0}, {1}", d.Value.Surname, d.Value.FirstName);

Console.WriteLine("Nurse: {0}, {1}", i.roomNurse.ElementAt(x).Value.Surname, i.roomNurse.ElementAt(x).Value.FirstName);

x++;

}

Console.WriteLine("------------------------------------");

Console.WriteLine("Enter the New Details Below:" + Environment.NewLine);

Dictionary<string, Dentist> roomDentist = **new** Dictionary<string, Dentist>();

Dictionary<string, Nurse> roomNurse = **new** Dictionary<string, Nurse>();

string[] rooms = **new** string[10]

{

"Room 1","Room 2","Room 3","Room 4","Room 5","Room 6","Room 7","Room 8","Room 9","Room 10"

};

int y = 0;

List<string> currentRooms = **new** List<string>();

Console.Write("Practice Room Count (Min: 1, Max: 10) :");

int roomCount = int.Parse(Console.ReadLine());

Console.Write("Receptionist's Staff ID: ");

string receptionistID = Console.ReadLine().ToUpper();

Receptionist check3 = Receptionist.roomReceptionist(receptionistID);

**if** (check3 == **null**)

{

Console.WriteLine("Receptionist Error");

**return** **false**;

}

**do**

{

currentRooms.Add(rooms.ElementAt(y));

y++;

} **while** (y <= roomCount - 1);

**foreach** (var r **in** currentRooms)

{

Console.WriteLine(r);

Console.Write("Assigned Dentist's Staff ID: ");

string dentistID = Console.ReadLine().ToUpper();

Dentist check = Dentist.roomDentist(dentistID);

**if** (check == **null**)

{

Console.WriteLine("Dentist Error");

**return** **false**;

}

**else**

{

roomDentist.Add(r, check);

}

Console.Write("Assigned Nurse's Staff ID: ");

string nurseID = Console.ReadLine().ToUpper();

Nurse check2 = Nurse.roomNurse(nurseID);

**if** (check2 == **null**)

{

Console.WriteLine("Nurse Error");

**return** **false**;

}

**else**

{

roomNurse.Add(r, check2);

}

i.RoomCount = roomCount;

i.RoomDentist = roomDentist;

i.RoomNurse = roomNurse;

i.Receptionist = check3;

}

}

}

Console.WriteLine("Error | Could not find specified Practice");

**return** **false**;

}

**public** **static** Practice currentPractice(Receptionist user) *//method to retrive which practice the current receptionist user is assigned to*

{

**foreach**(var u **in** allPracticies)

{

**if**(u.Receptionist == user)

{

**return** u;

}

}

**return** **null**;

}

**public** **static** Practice checkLocation(string x)

{

**foreach**(var p **in** allPracticies)

{

**if**(x == p.Location) { **return** p; };

}

**return** **null**;

}

**public** **static** **void** viewDentists(Practice p)

{

**foreach**(var d **in** p.RoomDentist.Values)

{

Console.WriteLine("{0} | Dr {1}", d.StaffID, d.Surname);

}

}

**public** **static** Practice checkReceptionist(Receptionist r)

{

**foreach** (var p **in** allPracticies)

{

**if**(p.Receptionist == r)

{

**return** p;

}

}

**return** **null**;

}

}

## Receptionist.cs

**class** **Receptionist** : Staff

{

string staffID;

string username;

string password;

string firstName;

string surname;

**public** Receptionist(string staffID, string username, string password, string firstName, string surname) *//object for receptionists*

{

**this**.staffID = staffID;

**this**.username = username;

**this**.password = password;

**this**.firstName = firstName;

**this**.surname = surname;

}

**public** string StaffID { **get** => staffID; **set** => staffID = **value**; }

**public** string Username { **get** => username; **set** => username = **value**; }

**public** string Password { **get** => password; **set** => password = **value**; }

**public** string FirstName { **get** => firstName; **set** => firstName = **value**; }

**public** string Surname { **get** => surname; **set** => surname = **value**; }

**protected** **static** List<Receptionist> allReceptionists = **new** List<Receptionist> *//list of all receptionists on the system*

{

**new** Receptionist("ABCDE","RECEPTIONIST","password","GENERIC","NAME"),

};

**public** **static** Receptionist roomReceptionist(string inputID) *//method to check if inputted ID matches that of a receptionist*

{

**foreach**(var r **in** allReceptionists)

{

**if**(inputID == r.StaffID)

{

**return** r; *//return matching receptionist*

}

}

**return** **null**;

}

**public** **static** bool checkReceptionist(List<string> input) *//method to check login input against receptionists*

{

int y = 0;

**foreach** (var x **in** allReceptionists)

{

**if** (input.ElementAt(0) == allReceptionists.ElementAt(y).Username & input.ElementAt(1) == allReceptionists.ElementAt(y).Password)

{ *//if inputted username & password match those of a recognised receptionist*

**return** **true**;

}

**else**

{

y++;

}

}

**return** **false**;

}

**public** **static** Receptionist loginInformation(List<string> loginDetails) *//method to check receptionist login information*

{

**foreach** (var r **in** allReceptionists)

{

**if** (loginDetails.ElementAt(0) == r.Username & loginDetails.ElementAt(1) == r.Password)

{

**return** r;

}

}

**return** **null**;

}

**public** **static** bool addReceptionist() *//method to add new receptionist*

{

List<string> input = addStaff(); *//calls abstract class for staff creation*

**foreach**(var r **in** allReceptionists) { **if** (input[1] == r.FirstName & input[4] == r.Username) { **return** **false**; } } *//Receptionist already exists*

allReceptionists.Add(**new** Receptionist(input[0], input[3], input[4], input[1], input[2])); *//Register Successful*

**return** **true**;

}

}

## Staff.cs

**public** **abstract** **class** **Staff**

{

*//abstract class for staff creation, acts as the basis for all staff role creation: with added information being altered based on which role is needed*

**protected** **static** List<string> addStaff() *//method to add new staff member*

{

string staffID;

string username;

string password;

string firstName;

string surname;

Console.WriteLine("Enter the Staff's Details Below:" + Environment.NewLine);

Console.Write("First Name: ");

firstName = Console.ReadLine().ToUpper();

Console.Write("Surname: ");

surname = Console.ReadLine().ToUpper();

Console.Write("Username: ");

username = Console.ReadLine().ToUpper();

Console.Write("Password: ");

password = Console.ReadLine();

staffID = Guid.NewGuid().ToString(); *//Generates a unique 5 digit ID for the Staff ID*

char[] idCharacters = staffID.Take(5).ToArray();

staffID = **new** string(idCharacters).ToUpper();

List<string> input = **new** List<string>(){staffID, firstName, surname, username, password};

**return** input;

}

**public** **static** string checkStaff(List<string> userInput) *//method to check if inputted login information matches an exisitng login and role*

{

string loginRole = "";

bool loginBool = **false**;

loginBool = Dentist.checkDentist(userInput);

**if**(loginBool == **true**)

{

loginRole = "Dentist";

**return** loginRole;

}

loginBool = Nurse.checkNurse(userInput);

**if** (loginBool == **true**)

{

loginRole = "Nurse";

**return** loginRole;

}

loginBool = Receptionist.checkReceptionist(userInput);

**if** (loginBool == **true**)

{

loginRole = "Receptionist";

**return** loginRole;

}

loginBool = Administrator.checkAdmin(userInput);

**if** (loginBool == **true**)

{

loginRole = "Admin";

**return** loginRole;

}

loginRole = "Error";

**return** loginRole;

}

}

## Staff\_Menu.cs

**class** **Staff\_Menus**

{

**static** bool constantMenu = **false**;

**public** **static** List<string> loginBackup = **new** List<string>();

**public** **static** **void** receptionistMenu(List<string> loginDetails) *//initial menu for receptionists afer login*

{

**do**

{

loginBackup = loginDetails;

Receptionist currentLogin = Receptionist.loginInformation(loginDetails);

Console.WriteLine(Environment.NewLine + "Receptionist Menu" + Environment.NewLine + "-----------------");

Console.WriteLine("Type T to | Manage Treatments" + Environment.NewLine + "Type P to | Manage Patients" + Environment.NewLine + "Type A to | Manage Appointments" + Environment.NewLine + "Type X to | Exit the Software");

string menuChoice = Console.ReadLine().ToUpper();

**switch** (menuChoice)

{

**case** "T":

receptionistTreatments(); *//runs menu to manage treatments*

**break**;

**case** "P":

receptionistPatients(); *//loads menu to manage patients*

**break**;

**case** "A":

receptionistAppointment(currentLogin); *//loads menu to manage appointments*

**break**;

**case** "X":

Environment.Exit(1); *//closes the application*

**break**;

}

} **while** (!constantMenu);

}

**public** **static** **void** receptionistPatients() *//menu for receptionists to manage patients*

{

**do**

{

Console.WriteLine(Environment.NewLine + "Patient Management Menu" + Environment.NewLine + "----------------------");

Console.WriteLine("Type A to | Add a Patient" + Environment.NewLine + "Type E to | Edit a Patient" + Environment.NewLine + "Type V to | View all Patients" + Environment.NewLine + "Type B to | Return to the Previous Menu");

string menuChoice = Console.ReadLine().ToUpper();

**switch** (menuChoice)

{

**case** "A":

Patient.addPatient(); *//calls method to add a new patient*

**break**;

**case** "E":

Patient.editPatient(); *//calls method to edit existing patient*

**break**;

**case** "V":

Patient.viewPatients(); *//calls method to view all patients on the system*

**break**;

**case** "B":

receptionistMenu(loginBackup); *//loads previous menu*

**break**;

}

} **while** (!constantMenu);

}

**public** **static** **void** receptionistAppointment(Receptionist user) *//menu for receptionists to manage appointments*

{

**do**

{

Console.WriteLine(Environment.NewLine + "Appointment Management Menu" + Environment.NewLine + "----------------------");

Console.WriteLine("Type A to | Add an Appointment" + Environment.NewLine + "Type V to | View all Appointments" + Environment.NewLine + "Type C to | Book a Phone Consultation" + Environment.NewLine + "Type P to | View all Phone Consultations" + Environment.NewLine + "Type B to | Return to the Previous Menu");

string menuChoice = Console.ReadLine().ToUpper();

Practice currentPractice = Practice.currentPractice(user);

**switch** (menuChoice)

{

**case** "A":

Appointment.newAppointment(currentPractice); *//method to create new appointment*

**break**;

**case** "V":

Appointment.displayAppointments(); *//method to display all appointments*

**break**;

**case** "C":

Consultation.newConsultation(user); *//method to create a new Phone Consultation Appointment based on Patient-requested tickets*

**break**;

**case** "P":

Consultation.receptionistView(currentPractice); *//method to display all phone consultation appointments*

**break**;

**case** "B":

receptionistMenu(loginBackup); *//load previous menu*

**break**;

}

} **while** (!constantMenu);

}

**public** **static** **void** receptionistTreatments()

{

**do**

{

Console.WriteLine(Environment.NewLine + "Treatment Management Menu" + Environment.NewLine + "----------------------");

Console.WriteLine("Type A to | Add a new Treatment" + Environment.NewLine + "Type V to | View all Treatments" + Environment.NewLine + "Type B to | Return to the Previous Menu");

string menuChoice = Console.ReadLine().ToUpper();

**switch** (menuChoice)

{

**case** "A":

Treatment.newTreatment(); *//method to create new treatment*

**break**;

**case** "V":

Treatment.treatmentDisplay(); *//method to display all treatments*

**break**;

**case** "B":

receptionistMenu(loginBackup); *//load previous menu*

**break**;

}

} **while** (!constantMenu);

}

**public** **static** **void** dentalMenu(List<string> user) *//menu for dentists and nurses*

{

**do**

{

Console.WriteLine(Environment.NewLine + "Dental Staff Menu" + Environment.NewLine + "-----------------");

Console.WriteLine("Type P to | View all Patient Information" + Environment.NewLine + "Type A to | View all Appointments" + Environment.NewLine + "Type N to | Leave a Note on an Appointment" + Environment.NewLine + "Type C to | View all Phone Consultations" + Environment.NewLine + "Type T to | View all Treatments" + Environment.NewLine + "Type X to | Close the Software");

string menuChoice = Console.ReadLine().ToUpper();

**switch** (menuChoice)

{

**case** "P":

Patient.viewPatients(); *//method to display all patients*

**break**;

**case** "A":

Appointment.displayAppointments(); *//method to display all appointments*

**break**;

**case** "N":

Appointment.addNote(user); *//method to add note to an appointment*

**break**;

**case** "T":

Treatment.treatmentDisplay(); *//method to display all treatments*

**break**;

**case** "X":

Environment.Exit(1); *//close the application*

**break**;

**case** "C":

**if**(user.ElementAt(1) == "Nurse") *//Prevents access if user is a Nurse*

{

Console.WriteLine("Error | This Action is for Dentists Only");

}

**else**

{

Consultation.dentistView(Dentist.loginInformation(user)); *//Load Method to view all Phone Consultations for that dentist*

}

**break**;

}

} **while** (!constantMenu);

}

}

## Ticket.cs

**class** **Ticket**

{

**public** **static** List<Ticket> allTickets = **new** List<Ticket>(); *//list of all tickets on system*

string ticketID;

Patient patient;

Dentist dentist;

DateTime day;

string email;

Practice practice;

**public** Ticket(string ticketID, Patient patient, Dentist dentist, string email, DateTime day, Practice practice)

{

**this**.ticketID = ticketID;

**this**.patient = patient;

**this**.dentist = dentist;

**this**.day = day;

**this**.email = email;

**this**.practice = practice;

}

**public** string TicketID { **get** => ticketID; **set** => ticketID = **value**; }

**public** Patient Patient { **get** => patient; **set** => patient = **value**; }

**public** Dentist Dentist { **get** => dentist; **set** => dentist = **value**; }

**public** DateTime Day { **get** => day; **set** => day = **value**; }

**public** string Email { **get** => email; **set** => email = **value**; }

**public** Practice Practice { **get** => practice; **set** => practice = **value**; }

**public** **static** bool newTicket(Patient\_User user) *//method to create new ticket by patients*

{

Console.Write("Enter the location of your regular practice: ");

string userPractice = Console.ReadLine().ToUpper();

Practice x = Practice.checkLocation(userPractice);

**if**(x == **null**)

{

Console.WriteLine("Error | Practice not found");

**return** **false**;

}

Practice.viewDentists(x);

Console.Write(Environment.NewLine + "Enter the ID of your desired Dentist: ");

string dentistID = Console.ReadLine().ToUpper();

Dentist dentist = Dentist.roomDentist(dentistID);

Patient patient = user.Patient;

Console.WriteLine("Please enter a date that would be convenient");

Console.WriteLine("Please Note, this date is not garunteed, but acts merely as a guidline for when would be best: ");

Console.Write("Date (dd/mm/yyyy): ");

string y = Console.ReadLine().ToUpper();

DateTime day;

**try**

{

day = Convert.ToDateTime(y);

}

**catch**

{

Console.WriteLine("Error | Unreadable Date");

**return** **false**;

}

Console.WriteLine("Please enter an email address for updates on your Phone Consultation: ");

string email = Console.ReadLine();

string ticketID = Guid.NewGuid().ToString(); *//Generates a unique 5 digit ID for the Practice ID*

char[] idCharacters = ticketID.Take(5).ToArray();

ticketID = **new** string(idCharacters).ToUpper();

allTickets.Add(**new** Ticket(ticketID, patient, dentist, email, day, x));

Console.WriteLine("Your Consultation appointment has been requested | Be sure to check your email for any updates");

**return** **true**;

}

**public** **static** **void** viewTickets(Patient\_User user) *//method to view all of a patient's active tickets*

{

**foreach**(var t **in** allTickets)

{

**if**(user.Patient == t.Patient)

{

Console.WriteLine("------------------------------");

Console.WriteLine("Requested Day: {0}", t.Day.Day);

Console.WriteLine("Requested Dentist: Dr {0}", t.Dentist.Surname);

Console.WriteLine("Provided Email: {0}",t.Email);

Console.WriteLine("------------------------------");

}

}

}

**public** **static** **void** receptionistView(Practice p) *//receptionist method to view all current tickets*

{

**foreach**(var t **in** allTickets)

{

**if**(t.Practice == p)

{

Console.WriteLine("----------------------------------");

Console.WriteLine("Request ID: {0}", t.TicketID);

Console.WriteLine("Requested Day: {0}", t.Day);

Console.WriteLine("Requested Dentist: Dr {0}", t.Dentist.Surname);

Console.WriteLine("Requested By: {0} {1} (ID: {2})", t.Patient.FirstName, t.Patient.Surname, t.Patient.PatientID);

Console.WriteLine("Provided Email: {0}", t.Email);

Console.WriteLine("----------------------------------");

}

}

}

**public** **static** Ticket checkTicket(string i) *//method to validate a ticket ID*

{

**foreach**(var t **in** allTickets)

{

**if**(i == t.TicketID)

{

**return** t;

}

}

**return** **null**;

}

}

## Treatment.cs

**class** **Treatment**

{

string treatmentID;

string name;

string price;

string details;

**public** Treatment(string treatmentID, string name, string price, string details) *//object for each treatment*

{

**this**.treatmentID = treatmentID;

**this**.name = name;

**this**.price = price;

**this**.details = details;

}

**public** string TreatmentID { **get** => treatmentID; **set** => treatmentID = **value**; }

**public** string Name { **get** => name; **set** => name = **value**; }

**public** string Price { **get** => price; **set** => price = **value**; }

**public** string Details { **get** => details; **set** => details = **value**; }

**protected** **static** List<Treatment> allTreatments = **new** List<Treatment> *//list of all treatments*

{

**new** Treatment("TREAT1","Band 1","22.70","Check-ups, scale, polich, etc."),

**new** Treatment("TREAT2","Band 2","62.10","Fillings, repairs, etc."),

**new** Treatment("TREAT3","Band 3","269.30","Crowns and other procedures")

};

**public** **static** **void** treatmentDisplay() *//method to display all treatments*

{

Console.WriteLine(Environment.NewLine + "Treatments Available:");

**foreach**(var t **in** allTreatments)

{

Console.WriteLine("{0} | {1} | £{2} | {3}", t.TreatmentID, t.Name, t.Price, t.Details);

}

}

**public** **static** Treatment checkTreatment(string inputID) *//method to check if treatment exsists*

{

**foreach**(var t **in** allTreatments)

{

**if**(inputID == t.treatmentID)

{

**return** t; *//return matching treatment*

}

}

**return** **null**;

}

**public** **static** bool newTreatment()

{

double price;

Console.Write("Enter the Band Name for the Treatments offered: ");

string band = Console.ReadLine().ToUpper();

Console.Write("Enter the Price for the Treatment: £");

string priceString = Console.ReadLine();

**try**

{

price = Convert.ToDouble(priceString);

}

**catch**

{

Console.WriteLine("Error | Incorrect Price Format");

}

Console.Write("Enter the details of what this treatment band Entails: ");

string details = Console.ReadLine();

**foreach**(var b **in** allTreatments)

{

**if**(b.Name == band)

{

Console.WriteLine("Error | Treatment already Exists");

**return** **false**;

}

}

string ID = Guid.NewGuid().ToString(); *//Generates a unique 5 digit ID for the Treatment ID*

char[] idCharacters = ID.Take(5).ToArray();

ID = **new** string(idCharacters).ToUpper();

allTreatments.Add(**new** Treatment(ID, band, priceString, details));

**return** **true**;

}

}