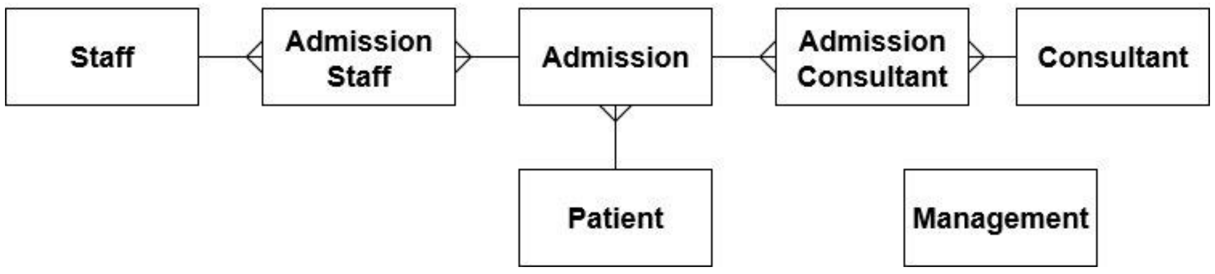
3 – Design

Contents

3.1 Breakdown of Problem

3.11 ER Diagram for Proposed System



3.12 ER Discussion

Patient

This is a key entity of the system, however it will not be the centre of the system, as that would be the admission entity, for reasons I will soon explain . The patient comes with a problem seeking to be treated for the ailment, this will be provided from consultants who propose solutions to resolve it usually either through surgery or medication. The whole purpose of the system for the user is to allow them to view their information and book new appointments if not automated. When coming to the hospital they will fill in a form asking for a few basic symptoms that allows for a broad diagnosis to be determined valuing if it is worth proceeding with an appointment. To uniquely identify every patient, they will have a primary key as **patient ID**, as one id value can't be assigned to two patients. The patient entity will have the foreign key of **Admission ID**, as it is possible that they may have many ailments and therefore admissions. Attributes applied to the entity will be like name, address, age and medical information that only relates to the patient rather than a specific ailment will also be used. The entity relates to the rest through the admission entity as the patient itself have no links but rather the ailment being treated. The entity will be capable of:

- **Creating an account.** Allows the user to create an account on the system. Once starting the system they will be able to initially enter their attributes and after this their file and respective information will be generated and finally have their new account added to the systems files. Once done the user will be logged in and will be able to have full access to the system

- **Archiving an account.** If the user is no longer wanting to use the account on the system, they can archive the account this will disable access to the account and prevent access to it. While it will not delete any data from the account it will remove any link to the information on other accounts or paths to the data.

- **Retrieving medical information from file.** When loading into the system they will need to see their personal information on the system the only way this can be achieved is if the information is retrieved from an external source, this process is to load all the information the patient may need from their respective files and allocate them to the correct fields within the system.

- **Saving new information to medical files.** While the patient can currently amend information this will only affect the attributes held in memory because of this there needs to be a process that permanently alters the information for future interactions with the system. This is the purpose of the process and will write any new/amended information over the old data.

- **Viewing medical demographic** like address, name, etc. This will contain all the information the patient has; nothing will be hidden for this user allowing for complete transparency with the patient. It is important that it is clear what every piece of information contains. This is the most crucial aspect of the system and is needed the most.

- **Regular enforced demographic updates** that their information is current and up to date. This will be to ensure that no information is out of date/ inaccurate such as current addresses or emergency contact information. This will be done every set period to ensure that data is accurate and up to date, however if the account is frequently accessed it shouldn't need to be used.

- **Editing demographic information** at any time. This is important otherwise data would become irrelevant due to external conditions outside the system such as address, resulting in data becoming meaningless.

- **Viewing the complete Admissions file** which includes all their medical documents like referrals, test results and dischargements. This another critical aspect of the system, this should allow for an extensive library of all previous documentation created in one area to be viewed at the patient's freedom.

- **Requesting medical file to be removed,** this will allow for the patient to have unwanted documents that are either no longer necessary or have no need to be archived. This should also allow for storage on the system to be freed up allowing for more storage. Only Consultants should remove the documents and no one else. While the document is removed the request should be logged. No reason should be needed if it meets the conditions.

- **Accessing the medical Jargon Library,** when reading documentation and are struggling to understand a document they will be able to search up the value in the library of words and receive a definition allowing them to now understand what it means. Overall this dramatically aids readability of advanced documents. If no definition currently exists, the word will be added to the list of required words.

- **Searching for induvial documents** by key tags like date. This will allow for instantaneous access to documents making finding them much easier overall and improving accessibility to their documents in general. To allow for unique values to search by a key field will be used.

- **Sorting files into key orders** like admission month. This will again improve accessibility to see certain documents. It will also allow documents to be read in chronological order or listed in a way that is suited to the patient. Sorting by fields will allow for groups to be created in addition to this.

- **Printing documents from the admission.** This will allow for documents that may be needed offline from the system to be kept. This will be an exact physical copy of the screen in a physical form. This could be used as a proof of ailment that could allow the patient to get time to recover from work, or the ability to purchase behind the counter medicines.

- **Viewing all prescriptions.** As the patient may have an assortment of illnesses at one point by having the ability to view all their current prescriptions along with dosage will be a great benefit to use on the system.

- **Simplistic expert system for preadmission diagnosis.** This will allow the staff to set up the initial admission with a predetermined idea of who the best consultant would be for the patient. It will also allow for an early diagnosis producing an estimate for what the ailment is.

- **Deleting notifications.** When the user logs onto the system they will see a collection of notifications that have accumulated over the period of inactivity on the system to bring them up to date with interactions with their account. When satisfied with what they've seen they can then be removed to clear screen real-estate.

Admission

This is the main entity of the system as it will relate all of the other entities together. While the patient entity relates to the actual person as an individual's generic information, the admission is the entity that revolves around their ailment like what the condition is, or who is treating it. An admission is created by the staff entity and is overall manged between them and the consultant. The reason it is to be considered an entity is due to the fact a patient can have multiple admissions and has many attributes like which ward the admission is in, which consultant manages them etc. The primary key is the **Admission ID** as it will uniquely identify the admission for that patient. However, will have a foreign key of **patient ID.** While this entity will not have many independent functions that aren't done by the patient. The actions performed by this entity are directly related to the admission rather than the patient like bookings and prescriptions which while do affect the patient are actually because of an admission as a prescription is needed because of that particular ailment. The entity will be capable of:

- **Creating an admission**. Allows the user to create an admission on the users account. When an admission is needed the user will enter their symptoms onto the system when they have finished their credentials are then added to their file along with their diagnosis and other respective information. Once done the admission is created the patient can then have full access to the admission

- **Searching for an admission.** As the number of admissions will be quite large for some staff or consultants the ability to search admission should be present to them on the system. This will allow for faster access to admissions on the system and should reduce needless navigating to find the desired admission. It will also be utilise when loading them from file and adding them to the data structure they will be held.

- **Archiving an admission.** When an admission is no longer needed it can be archived on the system. Once it is no longer needed for instance the admission is discharged it can then be archived by either the patient or consultant. While it still can be seen unlike archiving patients the ability to amend the information will be unavailable. This action can be undone if for instance the admission needs to be reactivated

- **Retrieving an admission.** To even see the information about the admission, it needs to be read from a permeant backing store. The process to do this will be to read the text file and assign meaning to the data afterwards by attaching induvial data to specific fields of the admission object. After this has been done the new instance of admission will now contain the desired data.

- **Date selection for Admission booking.** By letting the patient have a visual calendar for what date they would like an appointment will allow for an easier booking system. By letting them book it individually it will reduce the need for rescheduling due to the date being chosen by the patient and no external factors such as staff.

- **Viewing appointment dates.** This will allow for the patient to view all the upcoming bookings they have made or have been automatically assigned. This will allow for a coherent layout for all bookings so no mistakes will be made for which appointment is for which admission.

- **Rescheduling appointments for individual admissions.** This will allow for the patient to adjust the date of an appointment on the system, this will in return remove the old date and will add it to the new date. In order to avoid double bookings times that are already occupied will be locked out from selection.

- **Requesting for appointment cancelation.** This will allow the patient to request a cancelling of an appointment. If it is not 24 hours before it should be approved, and the date should be removed from the system. If the time is too close to the appointment it will need to be approved by staff.

- **Request dischargement from admissions.** If the patient believes that they are now no longer burned by the ailment and that sufficient evidence in the documentation is present, the request should be accepted, and the patient should be discharged from the admission.

- **File handling of Documentation.** When a consultant creates a document containing information regarding the admission, it will then be added to the text file containing the rest of those documents. Every admission will contain a different text file for the corresponding types of documentation. When the file s requested it will be read from the corresponding text file decrypted and displayed to the user.

Consultant

This is the other key entity for the system they will manage a patient's admission making sure that they eventually receive some form of treatment. Over time they will gain many patients to look after so it is important to make sure that double bookings do not occur. This entity will be focused on looking after patient health so will give the mundane tasks to the staff entity. The main aspect to the entity is the medical attributes of the consultant, to mimic real hospitals all consultants specialise in certain practises with exception to a few in Euxton practise general medicine. So, because of the countless different attributes to identify a consultant the most obvious one is to use the **Consultant ID** as the primary key. To add to this the consultant entity has a many to many relationships with an Admission, this is because if a consultant needs an extra opinion on something or can't attend an appointment a secondary "on call doctor" will be used. Because of this feature, a link table is needed to map the relationships together. Overall this entity will receive the most functionality in the system and are capable of:

- **Searching for induvial Patients** by key tags like patientID. This will allow for instantaneous access to patients making finding them much easier overall and improving accessibility to their available patients in general. To allow for unique values to search by a key field will be used. Only patients that go to the consultant will be available to them.

- **Retrieving information from file.** Before the account is even accessed all the relative information concerning the account is read from file and then assigned to their respective fields within the data structures attached to the consultant. Without this no information could be seen by the consultant. Because of this it is an important process to have included on the system as otherwise all information would be temporary and meaningless.

- **Saving information to file.** The purpose of this process is to make sure that once any fields in memory have changed are reflected in the backing store of the system. It is important to see this feature though otherwise having the ability to amend information in the first place would become futile. To add to this, this will not just be for the amending of admission data but also for other processes that require file handling and interaction on the system.

- **Printing documents from the admission.** This will allow for documents that may be needed offline from the system to be kept. This will be an exact physical copy of the screen in a physical form. This could be used as a proof of ailment that could allow the patient to get time to recover from work, or the ability to purchase behind the counter medicines.

- **Sorting patients into key orders.** Consultant will be able to sort their patients into key fields such as name, patientID or even age. By giving the consultant this ability, it will allow them to access patients at a much faster rate overall improving efficiency of the system as a whole.

- **Viewing the entire Demographic information regarding a patient.** This will allow the consultant to have the entire contents and all information about the patient available to them to view entirely. However only patients they treat will be available to them.

- **Adding documentation to an admission.** After an appointment or something noteworthy occurs regarding the admission, such as test results returning. The consultant will be able to type up his findings and save them to the patient's admission file. Only the consultant and patient will be able to view these documents.

- **Altering demographic information.** The consultant should have the ability to amend information such as allergies on the patient's admission section. This should only happen for their patients and should be added to the action log afterwards. The information should validated to make sure it is correct.

- **Viewing the appointment schedule.** The consultant should be able to see all upcoming appointments he has for only is patients. If a change is made, were a patient cancels, it should be seen on his end of the calendar. The times and some minor information should also be included for more clarity on the situation.

- **Enable the use of on call consultants.** If the consultant is unable to make a booking, they should be able to appoint a new consultant with similar abilities to take his place before the need to cancel is used. If this happens the patient will see that the consultant has changed only.

- **Requesting Follow up appointments.** If the consultant believes that another appointment is needed, they will request the submission to the staff entity who will set up a booking for them. When it is created the patient will receive a notification that the booking has been made, if they can't attend, they will reschedule it.

- **Discharging admissions that qualify.** After the consultant sees recovery in the patient or lack of said ailment, they can use judgment and then therefore discharge a patient from the admission. From this they will remove any upcoming appointments relating to that admission only. The admission will be declared as discharged also.

- **Reinstating of a prior admission.** If the patient believes that they are seeing previous symptoms or know the treatment hasn't helped, they can request a reinstatement of a prior admission. The consultant will then reactivate the admission allowing for new appointments to be made with them.

- **Prescribing suitable medication to treat ailment.** If the consultant believes that medication is required to help the patient, they will be able to add specific medication along with a dosage and intake times to the patient's admission allowing the patient to recover over time.

Staff(admin)

This entity is needed to reduce the workflow for consultants by offloading monotonous tasks to them. They will receive new requests for appointments by either the consultant (who believes an appointment is needed) or a user wanting to become a patient. The main focus of this entity is to help carry forward old documents from the previous system, they will achieve this by scanning in and adding tags to add meaning to the scans. There are no main attributes for the entity that describes their characteristics or traits, because of this they will have a primary key called **Staff ID** and will have no foreign keys this is because they will have a link table with the admission entity. This is because an admission may be created by one member of staff but could be given to another member later on for maintenance (editing information about the account). They will be capable of:

- **Adding new patients onto the system.** The staff entity should be able to add new patients onto the system. Once a non-user enters a list of systems and is urged to login, they can send their information to set up an account. Once it has been entered the staff will set up an account for them on the system and will set up an initial admission with the symptoms included.

- **Searching for patients on the system.** This should allow the user to search for patients on the system. This will greatly reduce the time needed to navigate the system, it should also be sued to help isolate the correct patients associated with the entity. A binary search obviously suiting the task best as data is long and ordered.

- **Reading information from file.** The purpose of this process will be to allow any information that the staff needs from the backing store to be initially read from file and then assigned to the correct fields of the data structure. This will be needed in order for data to be correct on the system, otherwise constant data entry will be needed. The other large benefit of this being that it only needs to be done once on logging in and then can be left alone for the rest of the time the user is logged in.

- **Saving information to file.** Just like how data needs to be read from file, it also needs to be saved. When the user is happy with the data fields they have entered they will then need to save their information on the system. Not only will this entail updating the fields themselves but also the writing of the data to file. While waiting for the user log of before writing to file is a notable process I believe constant periodic updates will benefit the type of system I am going for, it will also mean that the user will have to wait a substantial period of time when logging of to make sure data has been saved correctly.

- **Removing demographic/admission information.** If the patient no longer wishes certain information to be present on the system, they can request for it to be removed entirely. The staff entity will be in charge of doing this. They will judge if it is able to be removed and will either remove it or delete it respectively.

- **Viewing the nonconfidential Demographic information regarding a patient.** This will allow the staff entity to have the a more restricted view of the file's contents and all information about the patient available to them. The information included will be basic like address contact information, however, will exclude more personal information like contact information blood type and other information.

- **Adding new patient bookings.** The staff entity will be able to add new appointments on behalf of the consultant by choosing the most appropriate date and time to suit both the consultant and patient. They will not be able to see the more advanced parts of the bookings, but just enough to choose the best time and date.

- **Automating regularised appointments.** The staff entity will also be able to generate automated bookings by allowing the system to create new appointments without human interaction. If the desired date/ time is taken it will try and relocate a suitable replacement date/time.

- **Adding archived documentation from the previous system.** The staff entity will also be able to add scanned documents from the old system. Because of this they can also add new admissions and determine if they are new or old with respect to which system they were created on

- **Accessing the medical Jargon Library,** when reading documentation and are struggling to understand a document they will be able to search up the value in the library of words and receive a definition allowing them to now understand what it means. Overall this dramatically aids readability of advanced documents. If no definition currently exists, the word will be added to the list of required words

Management

Management will be responsible for the backend of the system making sure that the backup settings are acceptable. They are also responsible for adding new employees onto the system and can check what an employee has added to patient accounts. This could be what information they have added to a document, when this was done and finally who did it. This will be done through a transaction log unique to every employee. it should be for every employee rather than for every patient as if for whatever reason, a disgruntled employee makes dangerous amendments to patient information, like changing their blood type, it will be recorded to the employee. This allows for the complete list of affected profiles to be seen and then quickly fixed. As a side effect it may also put pressure on the employees to be careful, making sure e very input is correct as any mistakes will be eventually brought back to them. They will be capable of:

- **Adding new employees onto the system.** The management entity should be able to add new employees onto the system. From this they will also be able to set up credentials and other attributes such as wage and EmployeeID. Once created they should be able to access the system with the same rights as any other user that is the same entity.

- **Editing employee credentials.** Once an account has been created the management entity has the ability to alter fields in the particular entity's information such as wage, or hours per week. Once the information has been entered it will be validated to make sure it is correct.

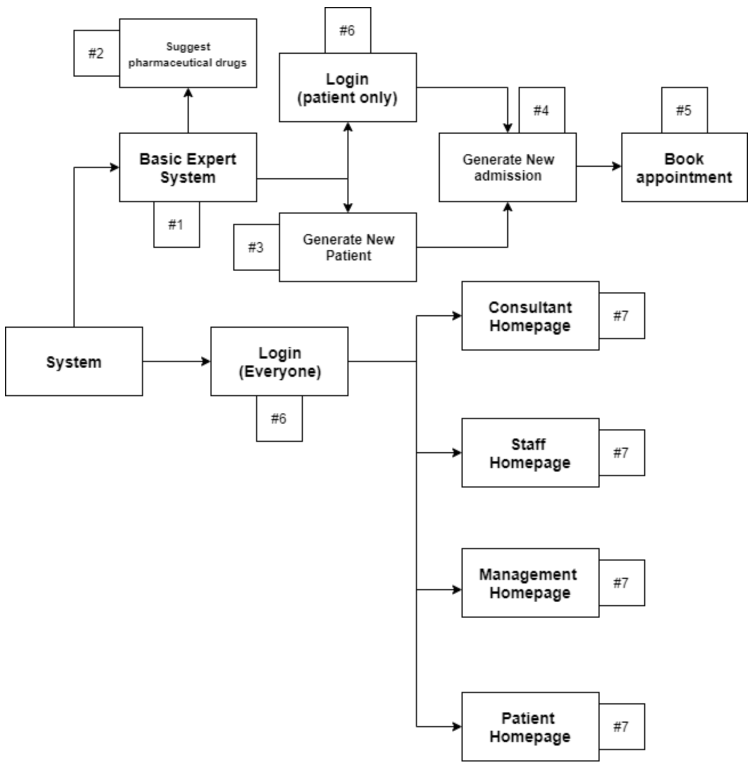
- **Archiving inactive accounts.** While deleting a record may be a much easier answer to removing employees, their records may be required in the future. Because of this they will be archived instead. Once the management entity knows that an employee no longer works on the system they will be able to deactivate the account, preventing logging in or them interacting with other aspects of the system.

- **Managing the action log of entity interactions.** The main focus on the entity will be to manage and view how other employees use and interact with the system. Any actions performed by an employee such as editing a set of documents will be recorded along with the time what was edited and who did it. The logs will be employee specific so that every file is specific to that employee. When looking through the logs two parameters will be needed to distinguish the start and end times to retrieve from file. Only this entity will be allowed this feature.

- **Reading information from file.** The main reason the process is needed is to allow correct data to be retrieved from file. The process will allow for data that is currently held in file to be brought into memory and then allowed to be used freely in the system. The reason this entity will need to utilise it is for the action log, as the employee entries will need to be loaded from file in order to be outputted to the user.

- **Writing demographic information to file.** Finally the last process to mention is the writing of demographic attributes to file, while the process of file handling isn't new the reason this is needed is down to the fact the management entity is responsible for creating new employees on the system. In order for that information to be saved it will need to be written to file.

3.13 Sub-Problems in Proposed Solution



Login Subproblems

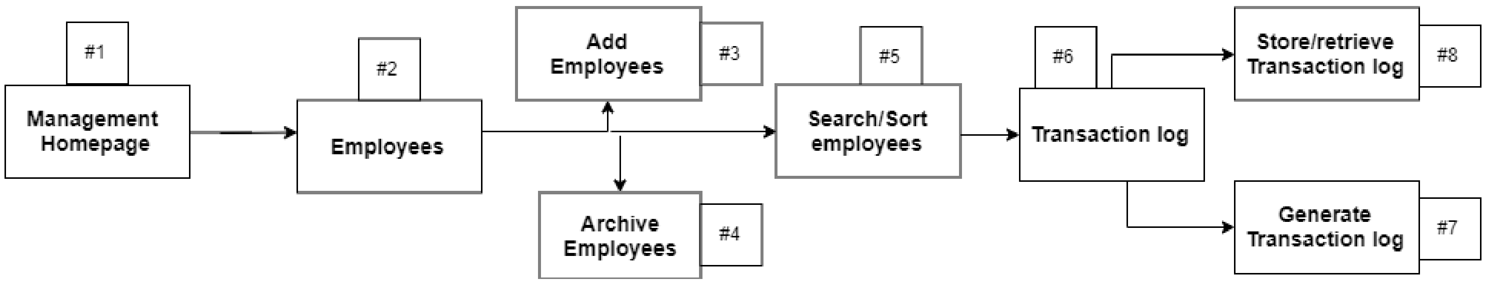
I have broken down my sub problem diagram into 5 key main sections. One is the initial login and account generation screen, this part of the program is to get the user logged onto the system or ask a new user for the symptoms they are facing this will then further prompt to either login and generate a new admission or request for a new account create, either way it finally leads to an appointment being generated for the user. After this, they then can access the system as normal patient and will be able to view the new booking they have just made. For current users they can do this or just press the new admission button on their homepage. Once logged on this then leads onto the rest of the sub problems, each one is for every type of user: Staff, patient, consultant and management.

For the objectives I have numbered each main sub problem and in the table below have linked it to a main objective I made in the investigation. One objective to take note on is seven, this objective is used to load and display the correct screen for the user. While not necessarily needed, it is used to show what the objective is for the home screens, and so the success criteria can be that everything should be correctly generated.

For subproblems 3 and 6 these show the only two possible actions that can take place when finished entering symptoms. The user will be brought to the screen and be given two options either login or generate a new account, while both can't be done simultaneously it will be possible to switch between options if needed for whatever reason like unintentional clicking.

Objective # (Investigation)	Objective Description (Investigation)	Sub-Problem # (Diagram)
#1	**Input/select symptoms into expert system –** This will save the attributes of the patient and eventually assign them to the admission if created, besides that it will be used to generate a diagnosis.	#1
#2	**Determine suggestion and best solution –** Once the patient has entered their information, the system will give a basic diagnosis and provide a subsequent solution to deal with it either create an admission or go the local pharmacy.	#2
#3	**Generating new patients for system –** The process is twofold; they will first enter their demographic attributes. Then this will be used to generate a primary key with all the following data then being written to file.	#3
#4	**Generating a new admission –** Here Once the account is either logged in or generated using the attributes entered and generated from before the information is all combined in the creation of a n new instance of admission and then followed by the writing of the details to file	#4
#5	**Booking a new appointment –** The entails in the generation of a booking with the details being added to the consultants booking file. It will also inform the patient when approaching the date of the appointment to remind them of it.	#5
#6	**Login –** This process will utilise the entered credentials and check whether or not they follow the existing values in the database. If they are accepted the user logs in and their respective information is retrieved. Else the process happens again.	#6
#7	**Display menu options –** This process is mainly just the creation of the GUI and that there is always a panel being shown to the user.	#7

Mangemant Subproblems



Here all the features and actions that the management staff can do is mostly shown here. When they login they are greeted to there home menu, as they will have no need for GUI compontents to use a handfull of features they will be using command line interface as it is ultimately the best option. From this they can then go further into the two main options and backups and managing staff.

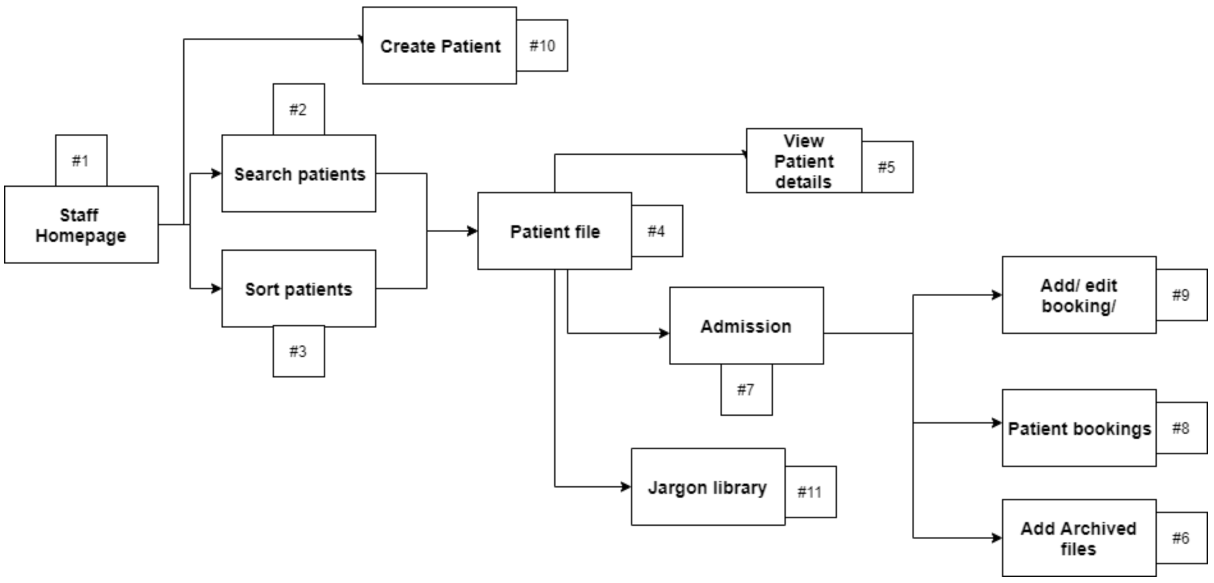
With employees the mangemnet staff have ultimate controll on adding staff and veiwing what they do on the system. They will have the ability to archive staff, this will result in an employee being hidden essentially from the system but their record is not moved or deleted. The transaction log is one of the most important secutiry features in place as it records all inputted data for what an individual employee has done. However as it will be quite large it will need to use paramters to allow for time periods between dates as it will take too long to output years of activtity in the real situation.

Finally they will be in control of back ups. The user here will only be able to ammend a few values like time of backup frequency and a few other basic attributes of a back up.

Objective # (Investigation)	Objective Description (Investigation)	Sub-Problem # (Diagram)
#7	**Display menu options for the entire entity –** This process is mainly just the creation of the GUI and that there is always a panel being shown to the user	#1
#7	**Display menu options for the employee section –** while the same objective as the latter this one is specifically to show that the homepage for the entity is generated correctly along with pulling any correct information from file along with it	#2
#8	**Add employees onto the system –** This will allow the management entity to create new instances of the other two types of employees on the system, once the fields are created they will then be written to file. The login credentials will be automatically supplied but will be changed on the initial logging into the system.	#3
#9	**Archive employees on the system –** just like how this entity can create employees on the system they will also be able to archive them on the system. While this is literally just a change of an attribute it will prevent further interactions on the system until the account is reactivated.	#4
#10	**Sort for employees –** This process will be to allow the management entity to change the order in which all the employees are stored, this will be useful when selecting the patient for the transaction log methods.	#5

#11	**Search for employees –** This process will be for the finding of the desired employee on the system, they will be able to enter the employees primary key on the system for it to return the account. This will ideally be used in the transaction log but may provide further functionality later down the line.	#5
#7	**Display menu options for the transaction log –** All this process is just the outputting in command line a series of questions about the desired employee along with other parameters.	#6
#12	**View an employee's transaction log –** This process is very basic, utilising the previous parameters entered in the prior method, the system will use the search to isolate the dates that lie within the inputs and will then return all the actions that employee has performed.	#7
#13	**Read/write transaction log from file –** In order for the other objectives to work this one will need to be used heavily. The purposes is to be a file handler on the system and act as an intermediary between the program data held in the backing system and the system that is running in memory.	#8

Staff Subproblems



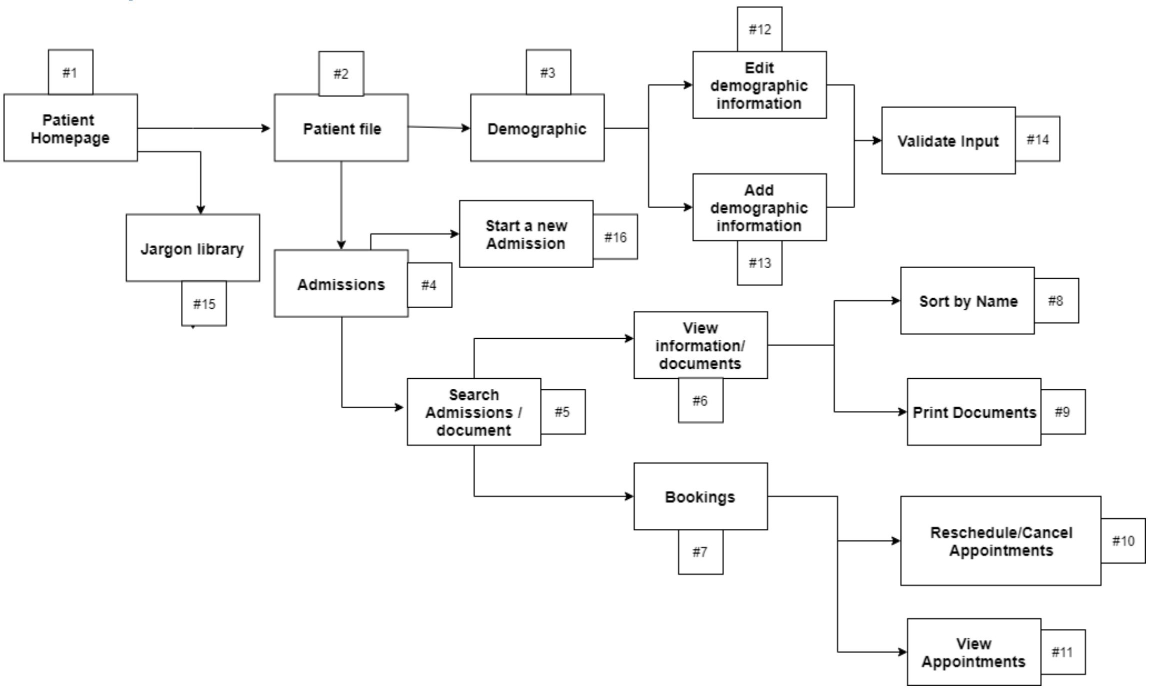
With staff when they log in, they will be greeted to their home screen. From this they will be able to search and sort patients to choose the person they want to check or update their information. The home page will provide many patients to select and will initially sort patients by consultant. The system will need to have an efficient search and sort algorithms in order to access the correct patient in a short time span. If they want to directly search for a patient they can enter a few of the key fields like patient number, name or admission number. Once they have found the patient in particular, they will be sent to the patient's particular homepage this will be a restricted version of the patient's homepage, as unnecessary will be removed. From this they will be able to see the demographic information of the patient, like name allergies and other general information.

They will be able to add archived files to the system. Another feature will be the ability to add new patients onto the system. Once a non-user has entered their symptoms onto the system the details will be available to the staff entity from this they will be able to create a new patient and an admission for them besides adding old documents to the system, the other main focus for this entity is managing booking. It will be up to them to make sure no double bookings or other issues occur they will do this by having the ability to change bookings. While the patient has the ability to select an initial date for an appointment it is for the staff to change it if it needs to happen.

Objective # (Investigation)	Objective Description (Investigation)	Sub-Problem # (Diagram)
#7	**Display menu options–** This process is mainly just the creation of the GUI and that there is always a panel being shown to the user	#1
#11	**Search for patients -** allows the staff entity to enter key field of the entity they want to look for, a binary search will be performed and any of the respective data that also satisfies the query are also returned.	#2
#10	**Sort for patients -** The staff entity will be able to order their patients in any key order available to them. This will reorganize them into said order. As of now an insertion sort is planned for use.	#3
#29	**View patient file –** Once located the staff entity will be able view the information related to the patient. In order for this to occur it will need to rely on searching and sorting objectives to isolate the patient initially.	#4
#16	**View patient details –** After the staff has got the chance to view the file they can go into further details and see some information relating to the patient; however this will be restricted to not disclose any personal information.	#5
#17	**Add archived notes from old system –** This will entail adding old documentation from the previous system over to the new one. One the document has been scanned over it can then be attached to the patient's old documentation file so it can be viewed.	#6
#20	**View patient admissions** – the purpose of this process is to allow the staff the ability to see their individual admissions on the system. Just like objective 5 this will for external information outside the entity to be viewed.	#7
#24	**View bookings** – This will also allow for the external entity from the patient part of the system to view the upcoming appointments that they have, this will be needed in case alterations need to be made along with other important processes like objective 9.	#8

#9	**Add/edit bookings –** This will allow the staff entity the ability to generate or update bookings on the system, once the instances holding the data have been changed the new data will overwrite the old in the file.	#9
#3	**Generate a new patient –** Once the staff entity wants to create an account they will enter the fields onto a new instance of the patient class, once done it will then be written to file where it can saved permanently. The desired user will then be notified of the credentials manually	#10
#39	**Use the jargon library to access definitions of medical words –** This entails using the system to enter and search for the definitions they do not understand. The search is binary due to the fact the data size is large and that the order is always sequential. However if more than one word satisfies the query they both will be returned.	#11

Patient Subproblems



#1 Patient Homepage
#2 Patient file
#3 Demographic
#12 Edit demographic information
#13 Add demographic information
#14 Validate Input
Jargon library #15
Admissions #4
Start a new Admission #16
Search Admissions / document #5
View information/ documents #6
Sort by Name #8
Print Documents #9
Bookings #7
Reschedule/Cancel Appointments #10
View Appointments #11

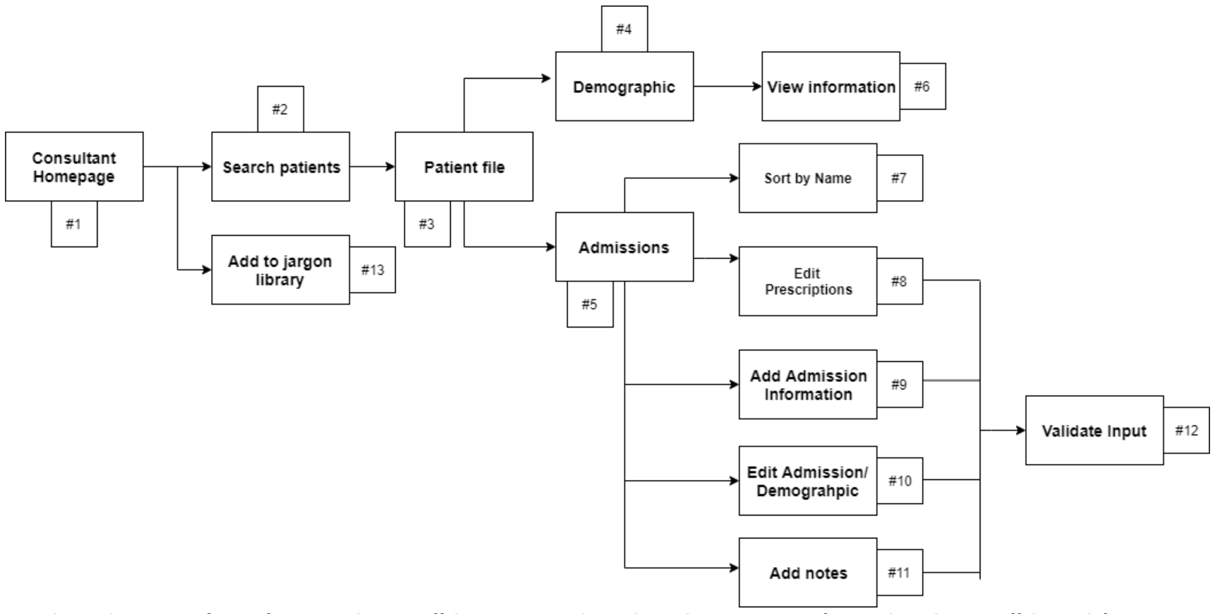
With the patient when they log in, they will be greeted to their homepage. From this they can get access onto the next two pages, the demographic page and admission page. From the admission page they get to search through and sort all their admissions as there may be quite a few they have created over their history with Euxton. Once they have found the correct one, they can see the information regarding the admission or see the booking details of past and upcoming appointments. From the admission information they can search through all the documents that have been accumulated and view any of them at will, they will also have the ability to print them off. On the demographic side of the patient's system they will be able to amend add to or delete information in the demographic, from this the system will validate the information they input to check to see if it is suitable. Other noteworthy subproblems would be the generation of enforced data verification of the patient's demographic this is to ensure that if the patient has been inactive for a few months and something has changed i.e. contact information this will be used to update the system.

Objective # (Investigation)	Objective Description (Investigation)	Sub-Problem # (Diagram)
#7	**Display menu options–** This process is mainly just the creation of the GUI and that there is always a panel being shown to the user	#1
#29	**View patient file –** Once located the patient entity will be able view the information related to the selves. In order for this to occur it will need to rely on searching and sorting objectives to isolate the patient's information from the rest.	#2
#21	**View their Demographic information –** This process will allow the entity to see their own data on the system. This is important otherwise there would be no point in storing the data in first place. It will also allow for further interaction with the data through having the ability to amend it. The process heavily relies on the ability to read the information from file.	#3
#21	**View their Admissions information -** Just like objective 26 this will allow for the information to be viewed, in order for this to work it will need to have a file reader which will retrieve all the information about the admissions from file.	#4
#26	**Search documents –** This will allow the patient the ability to search documents in their admission, this will utilise a binary search and will utilise a number of field keys. If more than one item meets the query then all the items are returned.	#5
#30	**View patient Demographic information –** This will allow the patient the ability to view their own personal information on the system. This will utilise the read methods in order to retrieve the data from the backing store. Without this the ability to hold the demographic information would be meaningless on the system.	#6
#7	**Display menu options (bookings) –** While basic process, this entails the generating of available appointments and will use a graphical date picker to allow the user to select appropriate dates on the system, along with this other text fields will be included to make sure that there is sufficient data to create a booking.	#7
#31	**Sort admission -** The patient entity will be able to order their admissions in any key order available to them. This will reorganize them into said order. As of now an insertion sort is planned for use.	#8

Objective # (Investigation)	Objective Description (Investigation)	Sub-Problem # (Diagram)
#27	**Print documents -** This process will allow the user to print of documents of the system, as the format of the documents still follow in the ratio of A4 paper this should be no problem. It will also allow for documents to be sent over large distances and then can be printed saving the cost of mailing them.	#9
#25	**Add bookings -** This will allow the patient the ability to generate bookings on the system, once the instance has been created along with the associated fields filled in it will be written to file. The consultant will immediately notice the change and can make any adjustments if necessary.	#10
#24	**View bookings –** This will allow the patient the ability to see their bookings on the system, without this there would be no pint in having a booking system anyway. It will utilise file handling in order to retrieve the information it requires.	#11
#22	**edit demographic information –** The purpose of this process it to allow the patient the ability to amend the information on the system This will utilise the file writing of the system. While the environment runs the attributes will be initially updated, but after that to prevent data inconsistency they will also be used to overwrite any old data held on the file that was initially read.	#12
#22	**Add demographic information –** just like sub problem 12 the ability to add new demographic information will be important, otherwise the system would not have any new data on it, to fully work it will utilise a file writing however as no old information is changed there is no data inconsistency as there is no other part of the system that contains that information.	#13
#26	**Validate all input –** Before any new account is made or any information added or altered it will first be passed through validation. Here every field will have associated validation according to what I believe requires it, if all the fields meet the standards they are saved else they are rejected. This will prevent issues with data types and invalid input on the system.	#14
#39	**Use the jargon library to access definitions of medical words –** This entails using the system to enter and search for the definitions they do not understand. The search is binary due to the fact the data size is large and that the order is always sequential. However if more than one word satisfies the query they both will be returned.	#15

Objective # (Investigation)	Objective Description (Investigation)	Sub-Problem # (Diagram)
#4	**Add new admission –** A key process to the entity they will be able to relate new admissions on their account. They will follow the standard method and then result with the new information being saved to a new instance of admission followed by the writing of that data to file.	#16

Consultant Subproblems



When the consultant logs in they will be greeted to their homepage from this they will be able to search through and sort for their patients, this will only be for their patients only. When they get to see the patient account, they will be able to see their information for their demographic. On the other side they get to view their admission information. They will be able to sort the admissions by key fields. To add to this, hey will be able to edit their current subscription, in addition they are the only user able to do this. Even more they will be able to add information to the account like documents and will be able to amend them. Finally, all the information they add will be validated to check that it is correct, this is an important data that will be amended and saved so the validation needs to make sure that all input is correct for the data that they change.

One of the most important features of the system for this user is for the use of an efficient GUI, the ability to get straight onto a patients file is a priority because of this they will need to be able to find the correct patient in a very short amount of time.

Objective # (Investigation)	Objective Description (Investigation)	Sub-Problem # (Diagram)
#7	**Display menu options–** This process is mainly just the creation of the GUI and that there is always a panel being shown to the user	#1
#28	**Search for patients -** allows the consultant entity to enter key field of the entity they want to look for, a binary search will be performed and any of the respective data that also satisfies the query are also returned.	#2
#29	**View patient file** Once located the staff entity will be able view the information related to the patient. In order for this to occur it will need to rely on searching and sorting objectives to isolate the patient initially.	#3
#30	**View patient Demographic information** – This will allow the consultant the ability to view their patients' personal information on the system. This will utilise the read methods in order to retrieve the data from the backing store. Without this the ability to hold the demographic information would be meaningless on the system. While they can see this they will be unable to edit this.	#4
#20	**View patient admissions -** Just like objective 26 this will allow for the information to be viewed, in order for this to work it will need to have a file reader which will retrieve all the information about the admissions from file.	#5
#16	**View patient details –** As consultants are always busy they will occasionally forgot the patient they are meeting with for an appointment. This process will help reduce this as they can just look up their account and will allow for the consultant to see who they are. In order to work this will utilise the file reading method to pull the correct demographic of the patient.	#6
#31	**Sort admission -** The consultant entity will be able to order their admissions in any key order available to them. This will reorganize them into said order. As of now an insertion sort is planned for use.	#7
#32	**Edit Prescriptions –** This will allow the consultant to prescribe medication to the patient on the system, it will use a file writer to overwrite any old prescription on the patient's admission. It can be done whenever the consultant feels that it needs to change and will subsequently inform the patient through the notification system.	#8

#4	**Generating a new admission –** A key process to the entity they will be able to relate new admissions on their patient's account. They will follow the standard method and then result with the new information being saved to a new instance of admission followed by the writing of that data to file. In addition to this it will be added to the action log that this event occurred.	#9
#34	**Edit Admission information –** While the patient can create the admission they are unable to change any information relating to it. Because of this the only entity left to do this would be the consultant. However to ensure that the document is correct before writing to file it will always be validated. In addition to this it will be added to the action log that this event occurred.	#10
#35	**Add notes –** Probably the largest process on the system the ability to create virtual documents was probably the driving force about the systems initial conception. It will entail creating a new instance of document followed by writing it to the patient file. In addition to this it will be added to the action log that this event occurred.	#11
#26	**Validate all input -** This entails using the system to enter and search for the definitions they do not understand. The search is binary due to the fact the data size is large and that the order is always sequential. However if more than one word satisfies the query they both will be returned.	#12
#40	**Adding definitions to jargon library –** This will allow the jargon library to be constantly evolving as terminology never remains the same, having the ability to include new words will become a useful feature on the system. To add to this as medicine is always changing it will mean overtime it will need to be updated, this process will just make it efficient.	#13

3.2 Input and Output Designs

3.21 Identification of Data

As the system contains a varying amount of data throughout it is important that each objective have their data requirements specified, along with this in order to clarify what the intended result is every process will show what the output will be. While every objective will have at least one output, hence validating its need as an objective, there will be also other outputs for instance when creating a patient, not only does the instance of the patient needs to be produced but also the fact that the entity needs to be added to the list of patients and also be written into file. This will also support the linking of objectives through the need of calling other processes while specific objectives won't be made here they will be used as a guide when writing pseudocode.

Objective #	Feature/ Process	Input Required	Output Created	Consideration of Output
1	**Selecting Symptoms**	-Symptoms	-Displays a confirmation message. -List of symptoms	To show that objective 2 is occurring (generating an appropriate diagnosis) the message should be created to validate that a diagnosis is being created. Otherwise it would be meaningless to even input symptoms if no recognition of them is generated. The other output will be non-visual but rather an array holding all the symptoms they have entered; this will be passed into objective 2.
2	**Producing a diagnosis**	-list of symptoms	-General Diagnosis (NOT Specific) -Generating a new admission (objective) -Suggestion for appropriate medicine	Once a general diagnosis has been determined it will always output what the system thinks the user has or a statement determining that a diagnosis was unable to be produced. To add to this it will also provide a statement suggesting if a new admission is needed, otherwise it would always generate an admission. However if the system believes that the information doesn't merit a consultation they can visit a local pharmacy for help. This is in place to still provide help despite being unable to provide any other help, however if the diagnosis appears to be incorrect to the patient they can always request an admission anyway.

3	**Creating a new patient**	-First Name -Surname -DOB -Emergency Contact -Allergies -Religion -Contact information -Address -Gender -Disabilities -Password	-A new patient account added to the patient list -Writing the patient's details to file -A new patient account -Patient ID/Username	The need for the new patient account to be added to the patient list class is because if this doesn't happen when searching for the new account it will not be able to be found and the patient won't be able to access the system. A username will be generated. To create a standardised name for logging into the system. The ID will be used to uniquely identify the entity on the system. After the instance of the patient is created in order to hold the patients as a collective it needs to be added to the list of patients.
4	**Creating new admissions**	User must be logged in. -A list of symptoms (Optional) -Early generated diagnosis	-A new entry in the admissions section of the patient file -Suggestion to book an appointment -Link an appropriate consultant /Ward to the admission -Admission ID - Writing the admission to the file -Adding notification to the patient	This will create a new admission in the patients' file. In the admission it will be very empty besides the information inputted by the patient a consultant will be in there also. The consultant will be the only person who can amend information in this part of the file down to it containing medical information. Finally if an archived admission has similar symptoms no new admission will be created but the archived one will be reinstated after seeing that the patient is still experiencing old symptoms. To add to this in order to make sure that the admission is linked to the correct consultant they too will also have the admission added to their account. Finally to allow the admission to be uniquely identified against the rest of the system it will have a composite key combining the patients ID against the number of the admission it is to create a unique id as admissions are unable to be deleted.

Objective #	Feature/ Process	Input Required	Output Created	Consideration of Output
5	**Booking a new appointment**	-Time -Date -Consultant	-Confirmation message -Location -Requirements/ extra information -Update upcoming appointments -Writing the booking to file -Adding notification to the patient	When the patient has finished booking an appointment, they will receive a message showing all the information regarding the appointment this will be able to be seen in the appointments section afterwards. A key point about this is that the consultant can update information about the booking like if any requirements are needed e.g. not eating before the visit. The update to upcoming appointments is important as it is needed to make sure that the consultant sees that a new admissions is created on the system for that patient.
6	**Logging onto the system**	-Username -Password	-Grant Authorised access to the correct account - Subsequent loading of the correct user panels -Error message informing that credentials are invalid	When the user tries to log onto the system there is only two possible outcomes either they get onto the system as they are granted access where the process in retrieving their information occurs or they have entered a credential wrong and so are rejected from accessing the system, but are informed that the credentials are wrong.
7	**Displaying the GUI**	-The components needed to format and create a usable GUI	N/A	N/A
8	**Adding new employees onto the system**	-First Name -Surname -Hourly pay rate -Department	-Patient ID/Username -Password -Adding the account onto the staff list -Writing the details onto the employee information file.	In order for the account to be re-accessed at later points a username and password have to be generated. This is different than the patient account generation as only management can do this process and as it is an employee account it needs to be standardised so a password should be generated automatically. To add to this the staff themselves need to be added to the system in order to be accessed at a later point by the system, otherwise the account is only accessible through the instance.

9	**Archiving employees**	-Employee account	-Disable interactions between account and system - Updates archive attribute for the account - Overwrites old attribute in file -Adds the employee action to their action log	When an employee no longer works on the system anymore, they should no longer have access to the system in anyway including their account, this is to prevent past employees abusing their old credentials to gain access to the system. This should also be used in making sure that no data leaks can occur or that any inactive accounts have access to restricted data. This will change a state of the account and will blacklist them from gaining access, re-archiving should whitelist them but this can only be done by management
10	**Sorting employees**	-The list of employees on the system -Key fields (Name, Department, ID)	-The sorted version of the list of employees in a specific order	When the sorting has finished the returned list should be shown immediately in the desired order. This is required as there would be no point otherwise in sorting it if it will not be displayed after being ordered or kept after sorting. This will also allow for the system to utilise the new order in any desired fashion, for example binary search.
11	**Searching employees**	-Employee name -Employee ID -Hourly pay rate	-Return all the employees who match any field query -Error message if no employee are returned	As there could be many users on the system that could have hourly pay rates that are the same i.e. minimum wage, the system needs to output all the data that match query, the point in searching is narrowing down the amount of entities that the user has to look through to find the desired one by retuning those that match the search it helps the user greatly. Errors should be used to make sure the user is informed that the search returned nothing.
12	**Viewing the transaction log**	-Starting date -Ending date	-Displaying the transaction log e.g. (DD/MM/YY) (Time) (Action performed) (Data)	When viewing the data from the transaction log it will be displayed in command line and will only show the data between certain dates provided from the user. No data will be viewed if an invalid input has been entered. This is to make sure that the user only sees the data they want to see. As over time the employee will accumulate lots of actions this is to reduce the amount they only want to see.

13	**Reading the transaction file from file**	-Text from the file containing all the transaction log information -Start and end parameters for the time periods	-Transaction log between the start and end times. -Error message if no data to read from file in given time periods - Error message indicating that the search has pulled no patients from the search	The data outputted will contain all the actions performed by the employee, it will contain when it occurred and what was done to the data long with other values such as the original data. If the parameters for time where invalid e.g. a set of dates that include days in the future a message will need to be shown informing the invalid input. This is to prevent the user thinking that the system returning nothing indicates that the employee has done nothing.
14	**Sorting patients**	-The list of patients on the system -key field	-The sorted version of the list of Patients in a specific order	The need for the sort to be used in the first place is to have the data returned in a desired order, therefore outputting the list in the correct order is essential. It will also be needed by the system in using the order for other important tasks like binary search where the order is important in a divide and conquer algorithm.
15	**Searching for patients (staff)**	-Patient ID -Consultant -Admission ID	-Patients who match any field query -Error message if no employee are returned	When the system searches through to return all the patients who match the search values, no entities that meet the search query should be excluded from the return. If a query returns no values a message should be returned and displayed to the user to inform them of a null return.
16	**Viewing patient details**	-The patients' file	-Patients' basic demographic information e.g Name Address DOB Number of admissions Allergies	The output data is needed to allow the staff member to view information held in the file, if this didn't occur there would be no way to view the data from an employee side of the system. This is different compared to the demographic as this has a more restricted access to data and so will be needed to prevent disclosing sensitive information, however, provide sufficient information there is some knowledge about the patient.

Objective #	Feature/ Process	Input Required	Output Created	Consideration of Output
17	**Adding old notes onto an admission**	-Scans of old documentation -Tags identifying a scan	-A document that is attached to a patient's archived admission -Adding a notification to the patient -Adds the employee action to their action log	The output is important because the system changeover occurs by transferring older documents to the new system. Without this it will be pointless having a new system due to the fact the predominant userbase will be on the old system. So the obvious choice would be to transfer active patients on the old system first then followed by subsequent admissions of that patient.
18	**editing of bookings by non-patients**	-Changes made to a booking	-Message confirming update of data -Validation of amendments -Updated version of the booking -Adding a notification to the patient -Adds the employee action to their action log	When a change needs to happen to booking data, it will need to overwrite previous version of the booking this will prevent loss of data integrity, where an update to one booking is not reflected in other areas of the system, this also allows for new bookings to fill in the spaces of old ones also. This process will be important to allow for the most correct data to be reflected in the system.
19	**Viewing booking information for non-patients**	The patient's appointment	-Date of appointment -Time - Ward/ Location -Consultant	The information in the output is important because in order to know about a booking the details need to be able to be viewed by the user. However as the entity looking at this data will be a non-patient it is important that no sensitive data is divulged when outputting the data.
20	**Viewing patient admissions**	-Patient admission file	-Admission ID -Consultant -Brief description -Ward -Next Appointment -Any documents	The output data needs to be used in order to allow the person viewing the admission to comprehend what the patient's admission is about. Otherwise there would be no point in holding information about the admission if it can't be seen. With this information being slightly sensitive as it contains medical diagnosis the output in this case will be restricted for non-important entities have the ability to acknowledge its existence but not know the details.

21	**Viewing patient Demographic**	-Patients demographic file	-First Name -Surname -DOB -Emergency Contact -Allergies -Religion -Contact information -Address -Gender -Disabilities -Password -Blood type	This is one of the most important outputs in the system as it holds all of the non-admission information regarding a patient, this will be important to keep so having the data to view is a priority as it will allow users to get a clear understanding of the patient's information. This will however only be useful if the details are accurate so to enforce verification of data integrity is important.
22	**Amend demographic information**	-Original demographic information -Changes to information	-New updated demographic information -Message that information has been amended -Adding a notification to the patient -Validation of amendments	The output data would be the new version of the information. This needs to be used in order to update a files contents, otherwise the new information needing to be amended won't be saved and therefore the original data will become redundant with it being irrelevant. This output will be the most important result regarding the demographic as described in objective 21.
23	**Validate information (Patient)**	-Data to validate -Validation criteria	-Message accepting the input as it is suitable -Message informing that data entry is invalid -Validation of amendments	This is another important output and process as it is used to make sure that data entry is valid and acceptable, otherwise the data input may be false or unacceptable and resulting in invalid data being saved to a field. Multiple validations may be used for the data to determine if data is allowed to be saved. While not all data will need to be validated those fields that do will be thoroughly checked to make sure invalid data entry is avoided.
24	**Viewing booking information for the patient**	The patient's appointment	-Date of appointment -Time - Ward/ Location -Consultant -Extra requirements -Days until appointment	The information in the output is important because in order to know about a booking the details need to be able to be viewed by the user. What makes this different is that it is for the patient and will have data in more detail that would be unneeded for anyone else. This information should show the user the exact details about the booking for the user, without this the system would have no ability to utilise bookings as they are never saved or outputted.

25	**Adding new bookings**	-The fields that make up a booking	-Message informing that the booking has been created -Adding a notification to the patient -Adds the employee action to their action log	This is output data is important as it is needed to amend data regarding an appointment. If the patient can't attend the appointment this can avoid the consultants time being wasted and could allow another patient to view them instead. Without this having the ability to view booking appointments will become futile as there would be no appointments to view.
26 A	**Sort Documents**	-List of documents in admission to sort -Key field to sort by, e.g. Date	-The new sorted list of documents in order by requested field	This output is important too because a patient will collect lots of documents over the course of their treatment and it is hard to search through in an unordered list. By having the ability to sort documents is a major benefit.
26 B	**Search Documents**	-List of documents in admission to search -Key field to search by, e.g. Date	-The list of documents that meet the query -A message that informs that no documents exist	The output has been used as it will return the patients that match the query this is useful as there are many documents to look through by reducing that to a handful is vital. The message is also important as it informs the user that the search as not returned any results.
27	**Printing documents**	-Document	-A physical copy of the document printed off -Adding a notification to the patient -Adds the employee action to their action log (if performed by the consultant)	This is important as it allows for data to leave the system and allows for data to be viewed offline. It also allows patients to access the information at home which is a major benefit. While the feature is in place to allow patients to have a physical copy it won't on the system to just print off every document, for this will just introduce the current issue in Euxton.
28	**Searching for patients (Consultants)**	-Patient ID -Consultant -Admission ID	-Patients who match any fields presented in the query -Error message if no employee is returned	When the system searches through to return all the patients who match the search values, none should be missed. If a query returns no values a message should be returned and displayed to the user.

Objective #	Feature/ Process	Input Required	Output Created	Consideration of Output
29	**Viewing a patient file (Consultant)**	-Patient file	-Patient homepage -Patient Demographic -Patient Admissions	The output data is important because the consultant will need to access all the information the patient has as they will treat their aliment. Because of this all the information should be displayed to them and available to view at their own availability. The biggest benefit of this is that the file will no longer have to be requested but rather immediately accessed.
30	**Viewing patient Demographic information (Consultant)**	-Patients demographic file	-First Name -Surname -DOB -Emergency Contact -Allergies -Religion -Contact information -Address -Gender -Disabilities -Password -Blood type -	This is one of the most important outputs in the system as it holds all of the non-admission information regarding a patient, this will be important to keep so having the data to view is a priority as it will allow consultants to get a clear understanding of the patient's information. However a key part to note about this is that the consultant will be unable to alter this information as it has no relationship to them.
31	**Sorting admissions**	-List of admissions in patient file to sort -Key field to sort by, e.g. Date, consultant ward	-The new sorted list of admissions in order by requested field	This output is important too because a patient will create lots of admissions over the course of the patient's treatment and will be hard to search through in an unordered list. By having the ability to sort documents is a major benefit
32	**Editing prescriptions (consultant)**	-Original prescription -Changes to be made	-New updated prescription -Adding a notification to the patient -Validation of amendments -Adds the employee action to their action log	The outputs for this are needed as overtime a patient may recover and require a smaller dosage, or on the flip side stronger dosages. This allows for the prescription to be altered and will notify the patient of it, this will allow for the prescription to become a dynamic part of the account evolving over time rather than just staying as a static value, as in real life this is not the case.

33	**Adding admission information (consultant)**	-Admission information to be added: Symptoms diagnosis Treatment - Admission file	-Ward -Room -Consultant Name -Date of Next Appointment -list Of Symptoms -Current Diagnosis -Staff Name -Active -New updated version of file with the new information -Adding a notification to the patient -Adds the employee action to their action log -Validation of amendment	This is important as initially when an admission has been created certain areas of information like diagnosis may be blank by having the ability to add information to the admission allows for the file to be up to date. While editing information is different this will be the case when an admission is created and in some examples the expert system is unable to diagnose a patient. Because of this this output will correctly write to file the information to the correct field.
34	**Editing admission information (consultant)**	-Original admission information -Changes to information	-Ward -Room -Consultant Name -Date of Next Appointment -list Of Symptoms -Current Diagnosis -Staff Name -Active -Message that information has been amended -Adds the employee action to their action log -Validation of amendments	The output data would be the new version of the field about the admission. This needs to be used in order to update a files contents, otherwise the new information needing to be amended won't be saved. To add to this for instance as I undoubtably assume the expert system for diagnosing patient's symptoms will never be correct all the time and may make errors, because of this it is important that the ability to rectify these mistakes are in place.
35	**Adding documents (consultant)**	-Documents for an admission -Admission fie	-Creating a new document -Adding the document to the list of documents attached to the admission -Adding a notification to the patient -Adds the employee action to their action log -Validation of additions	The output data or the new admission file is important because it will contain the new document that has been added to the file, without this process, the file will never receive new documents. The output is needed without it the documents will never be attached to the patients admissions file.

36	**Encrypting data before being written to file**	-Data needing encrypted, any normal text	-Encrypted data, that is unable to be read unless decryption cypher is known	The output is needed because the whole point of the method is to scramble the data to a point where it can't be read normally. By doing so the data will be unable to be red by an outside party who don't know the encryption cypher. As the system holds very sensitive data it is vital that no one besides the desired parties can understand it.
37	**Decrypting that has been read from file**	-The encrypted text	-The original text that was put into the encrypting algorithm	The who point of encryption and decryption is to only allow authorised users to read the data if only encryption was used so no one would be able to understand any text, so decrypting scrambled data allows meaning to be given the text. While encrypting it is important the process of unscrambling it is more important, in order for the system to retrieve the data the system will need to decrypt the data and pass it through to the front end.
38	**Using the Jargon library**	-A word that the user does not understand	-Either: The description of that word Or A request to add the definition to the library	From the investigation a common problem with document inspection is understanding complex medical terminology. This allows for the description of a medical word and allows non-medical educated users (non-consultants) the ability to understand what they mean. The purpose of the output is to allow the user to understand definitions for words that they couldn't understand.
39	**Adding to the Jargon library**	The definition and word of a medical term	A new entry in the terminology library includes the name of the term and a following password -Adds the employee action to their action log -Validation of inclusions	It is important to have this output otherwise no terms could be added to the library after development, as I have very little training in this area the library would only consist of a handful of definitions. It also allows for new terminology to be added over time. This will prevent the feature from being a static list of values. To add to this as new terminology is constantly being added it is important that this list of words updates along with it.

40	**Search through Bookings**	The search patient	Return the correct booking for the correct admission from the consultants list.	This is very important as the booking file will contain the information regarding the appointment for the consultant so if one in specific needs to be found this objective will return it. This will also be important in reducing the available amount to the consultant who will have many over the course of a week.
41	**Search through demographic information**	The search patient	Return the correct demographic of the desired patient	This is a very important output as it will allow for demographic information to be retrieved from the file they are all stored in, otherwise they will not be able to be used. This will be needed every time any demographic information is retrieved, edited or used so making sure it's performance is great is important.

3.22 House Style

Colour scheme and visual impact

For the house style of the system I wanted it to seem professional and minimalistic but not seem too simplistic and basic. I have drawn inspiration from a few types of medical software including CloudCare, a piece of software I researched during my investigation, as well as the bespoke software currently in place at Euxton and finally the operating software that my home computer runs on. Overall, I want the colour scheme to look similar to those seen on all Euxton documentation, a primary white, with a royal blue as a secondary colour. In some areas I intend the design and layout to be aimed towards efficiency and fast access such as the consultant and management side of the system, whereas in other parts I want it to be simple and clear what every feature does. However I don't want the system as of now seem to be contrasted into 2 separate sections with that being staff and patient, as it would make the system feel almost two separate systems which isn't the case. However an important focus is to get screen designs to feel full of features but not to the extent they seem cluttered or disordered.

Intended Usability

Overall, I want the design to be intuitive and welcoming. I intend for all users besides management to use a graphical user interface. To avoid the layout from becoming two tone I will also use a selection of greys to avoid over exposure of the plain white colour. To highlight buttons and objects with actions attached to them I will use a darker grey boarding on a black, the contrast created will clearly indicate which components are interactable on the window, which should help improve usability.

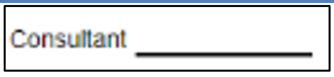
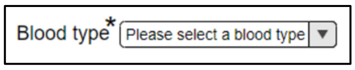
However, the main drawback to the design and layout will be accessibility features. While it would be ideal to have touch screen accommodations for disabled users, the layout for other users would mean screen real estate would be wasted for desktop users and nonideal for selecting components with a mouse and therefore such design preferences have been obscured from the design. In addition to this I want the system to look professional, when having large text and components I think it would diminish the effect I want. Finally as the main platform the system will be ran on will be personal computers I think compensating for a small minority while a nice gesture to accommodate for, would hamper the user experience for most.

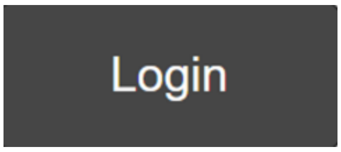
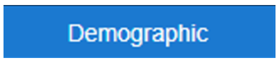
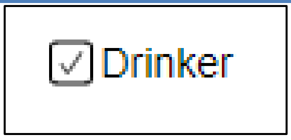
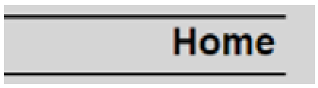
Swapping between screens

As the system deals with many screens, it is important that the correct method of loading a panel is used in order to allow the system to deal with swapping between them. The first thing to make sure that occurs is that the loading of panels causes no visible discrepancies which could hinder the user experience. The way to address all the panels I will utilise a 2D array of panels from which I can use to correctly call the correct instance of that panel without having to create a new ne and therefore save on resources. In case of slow loading times a simple loading panel will be inserted between pages loading to hide the generation of any component, while basic it will avoid any empty pages being seen. This will mainly be for presentation but could allow for features to be correctly loaded in. This will be necessary as the performance of the software will vary based on hardware so a default loading screen should be present regardless of the fact if it does or does not need one.

Component layout and design

While java does provide a set of standardised formats for graphical components in java swing, they all have the ability to be formatted, either through convenient senses such as .Size or .Font methods. However in order to completely customise the system some overloading of methods may need to be incorporated.

Element	Font Type, Size, Colour	Example	Justification
Labels	Font: Helvetica Size: 14 Text colour: Black (#000000) Background colour: None	 Consultant	Labels will be needed on the system to allow for text to be shown like in the example. There is nothing special for this component however I have decided on the font of Helvetica as it is a common font used for general text. Size 14 seems to be a large enough size, so that it is readable but does not cover the entire screen. If the size needs adjusting afterwards it will be done in post prototyping and testing .
Combo Boxes	Font: Helvetica Size: 14 Text colour: Black (#000000) Background colour: White (#ffffff)	 Blood type* Please select a blood type	Combo boxes are also used because they are an essential component for selecting only one input from a predetermined list. This will greatly reduce the need for validation in some areas of the system. There is little needed for the design justification besides that the text font will also follow the format of labels for the reason they are easy to read.

Selectable buttons	Font: Helvetica Size: 16 Text colour: White (#ffffff) Background colour: Dark grey (#444444)		The reason this button has a strong colour difference from any other component is because I intend for these to grab the user's attention. This will allow for an easy identification of selectable buttons. By having this as a standard scheme it will allow for instantaneous knowledge of the components function. The shade of grey I feel allows for a good contrast to the lighter colours of the system.
Selected buttons	Font: Helvetica Size: 16 Text colour: White (#ffffff) Background colour: Blue (#3a78d7)		As some buttons will be selected such as the tabs by having a strong vibrant colour to contrast the entire system, it will clearly show components that have been selected. Blue was chosen as it is a main colour used throughout Ramsay branding and also makes the system feel less black white and grey.
Checkboxes	Font: Helvetica Size: 14 Text colour: Black (#000000) Background colour: None		There is little justification for these components design except for the text. The text will be exactly the same as the format used for the labels this allows for a universal standard allowing for a similar reading experience across all components
Titles	Font: Helvetica Size: 24 Text colour: Black (#000000) Background colour: None Bolded		The reason I will have the title so large is because I want to reduce the confusion of where the user may be by having a clear title to every page it will allow for some idea for what the page should contain. By having button labels have the same text as the title it will allow for a clear route to every section of the system

Main background	colour: Light grey (#d6d6d6)	⬜	I believe that a light grey background is needed because it will highlight the main components of the other system through providing a contrast in colour. By having dark colours in front of the background it will grab the user's attention. I feel that the gradient is not too light to appear too white, but not too dark to provide a contrast between components.
Minor panel backgrounds	colour: Dark grey (#7a7a7a)	⬛	This will be a colour that is used as a secondary colour at most and will only be used to compliment the other colours of the system. It will be used to reduce the amount of light grey from the background in same areas that I believe to be too much. While it is a similar shade to the button colour the two will not be directly touching each other and will have a lighter colour separating them

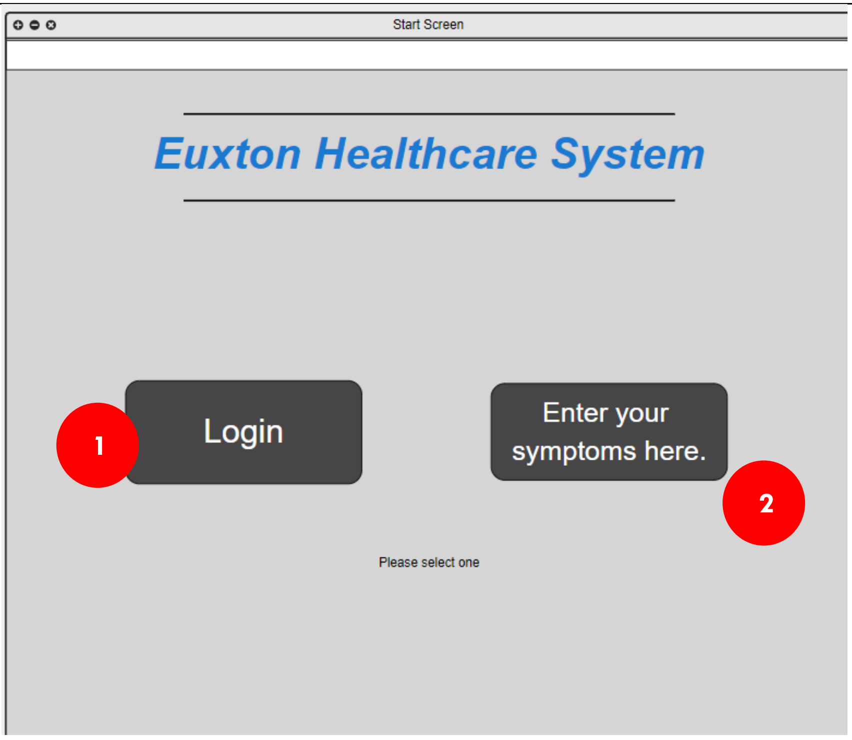
Pop ups and minor panels

With other windows that are not used by the main system frequently, I intend for a similar UI design and formatting layout to the one I have planned for main windows. Pop ups will be primarily used to display error messages to the user, the best example will be invalid data input. If an error has been found the pop-up window will display the message and will have a button allowing the user to re-enter data. Other windows included would be the jargon library used for advanced medical terminology this will be a window that can be opened and ran concurrently if the user finds a word that is unknown to them, they can enter it on this window and receive a definition. The main point of these popup windows is to help inform the user without distracting them or pulling them away to another panel preventing for the action to happen again for instance having the ability to research for a word.

3.22 Designs of Input & Output

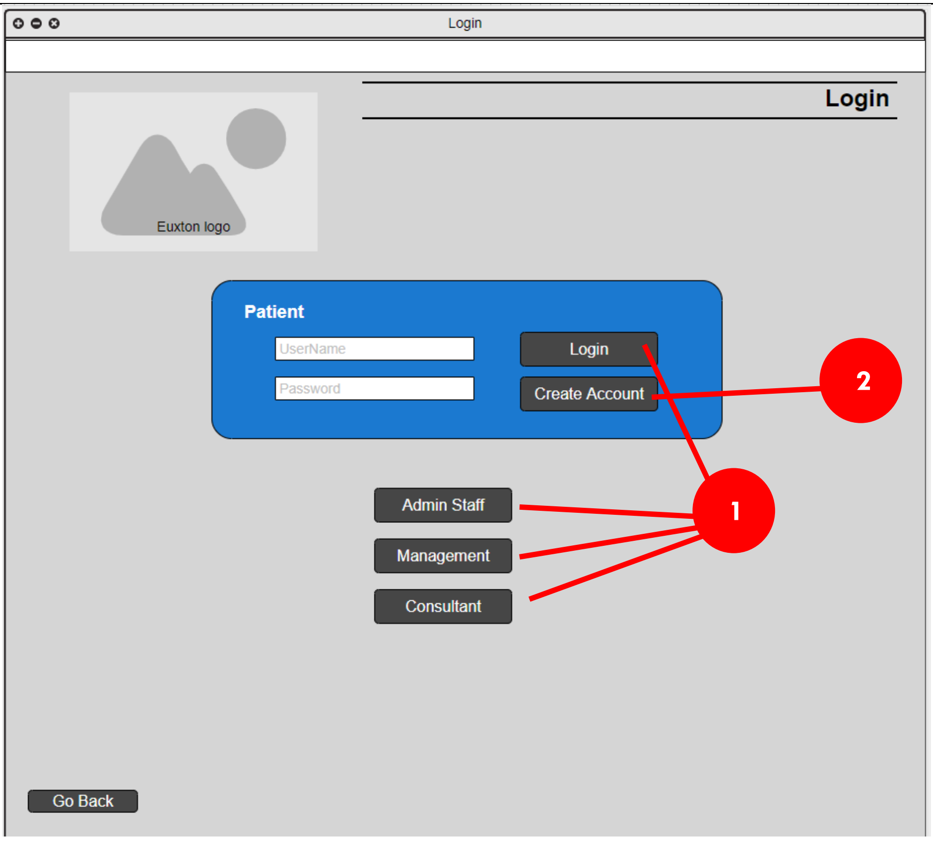
To distinguish whether the current user is a patient or consultant their respective fields on the system will vary between being editable or not. The best example would be between admissions and the demographic. For patients they will be able to amend update and add any information to the demographic but no other user will be able to unless creating a new account. For the admission the patient will be unable to edit any information on that but the other users will be able to. However for the sake of reducing the size of the document only one version will be shown to the user. Throughout the I/O the focus has been to show the user the features of the system but not to the point of being overwhelmed.

Virtual Screens



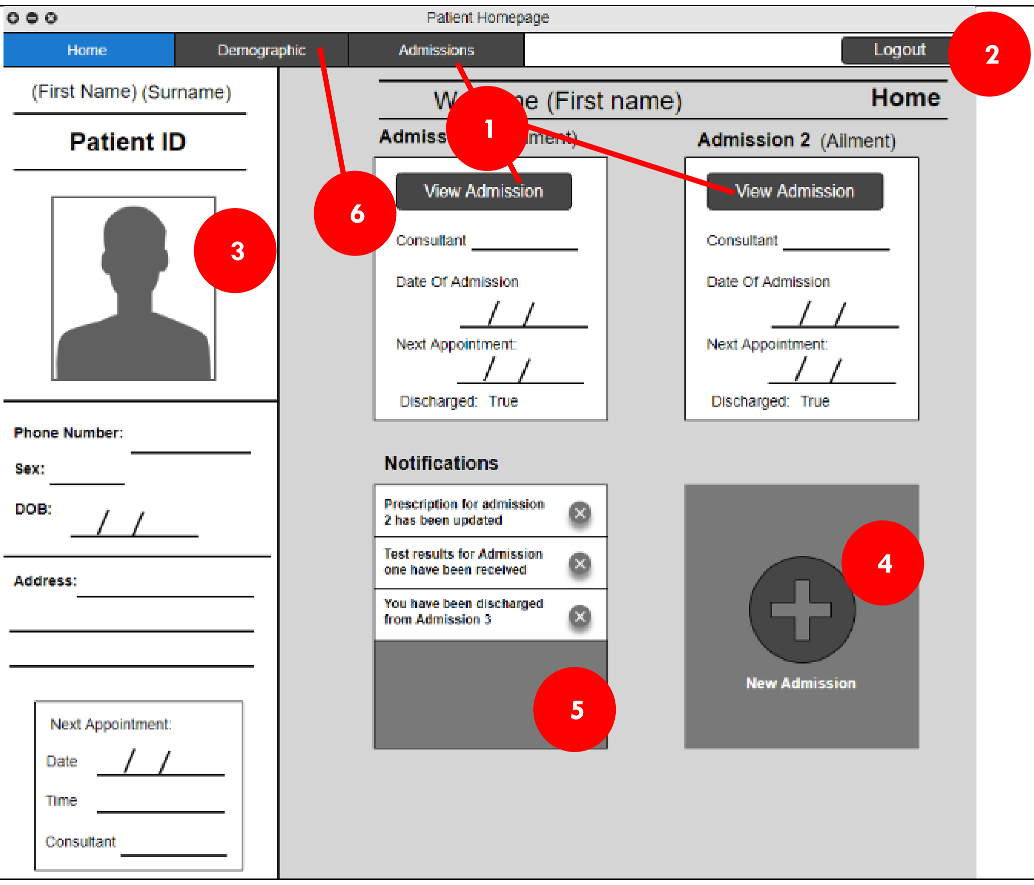
The design above is a predeveloped rendering of what the user interface will be and was created using Moqups. It is not the actual design that will be implemented as it will be subject to changes later on

Name of Design	Format	Consideration of Design & Purpose	Features Included
Start Screen	Screen	When the program is ran this is the first screen that is presented to the user. This has one of two available options: Login to the system or use the expert system. I wanted this to be the initial screen due to the fact nonusers may not have an account and may wish to enter their symptoms without having the hassle of creating an existing account. I chose the layout of the two buttons in this format as it feels to be large enough to resemble a mobile application and receive a simplistic look but seem small enough to still be reasonable to click with a mouse and not feel as if it would be better off as a touch screen interface. If they were any bigger, I would feel they would be out of place however if too small the screen would feel too empty. The input is needed inorder for the user to access the system, without it the option to login would be unavailable. Besides selecting the start page there is nothing else to this panel.	**-Load the login page** (objective 7) Button 1 **-Load the Expert system page** (objective 7) Button 2 **-The Panel that is loaded after logging out.** (objective 7) **-Welcome The user**



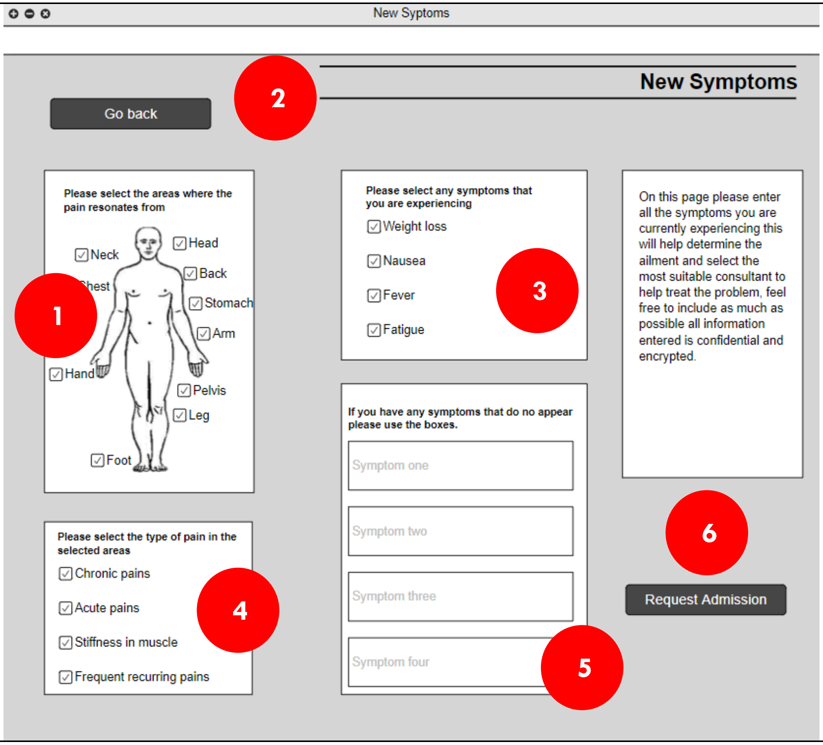
The design above is a predeveloped rendering of what the user interface will be and was created using Moqups. It is not the actual design that will be implemented as it will be subject to changes later on

Name of Design	Format	Consideration of Design & Purpose	Features Included
Patient homepage	Screen	This is the user login screen, this will be the same for all 4 types of user, when they start the program, and have selected this screen they will be greeted to this login screen to reduce the search times and granting authorisation 4 buttons will be used to split up the users into the respective groups. If, however the users decide to go back they have the option to do so. I have chosen this layout because of its similar layout used throughout most login services. I have tried to reduce unneeded buttons where possible as this doesn't need to be cluttered with objects and distracting features. Overall the use is to get the user to the correct part of the system they want to get to. The format follows conventional methods of doing so The reason the login credentials will be needed is to allow a query to be made with the system allowing the program the ability to compare the inputs against the system and determine whether or not they exactly meet what is in the data base. Without these inputs no output could occur on this end.	**-Login the patient** (objective 6) Button 1 **-Patient account creation** (objective 3) Button 2 **-Login the Staff** (objective 6) Button 1 **-Login the Consultant** (objective 6) Button 1 **-Login the Management** (objective 6) Button 1



The design above is a predeveloped rendering of what the user interface will be and was created using Moqups. It is not the actual design that will be implemented as it will be subject to changes later on

Name of Design	Format	Consideration of Design & Purpose	Features Included
Patient homepage	Screen	This is the Patient homepage, when patients log on they are greeted with this screen. This will contain a wide range of quick access features to allow a fast access to the entirety of the patient's system. A feature I have been inspired from was the left-hand panel containing some brief information about the patient from the software CloudCare which I researched during the investigation into the system. The purpose mainly of this page is to greet the user and provide a structure linking both the demographic and admission parts of the system together. The notifications box will be used to show any updates to the user they may be unaware of. Another design aspect worth mentioning is the next appointment tab. This is included to allow for instantaneous access for viewing when the next appointment is. The bottom right box is a large button that will allow the patient to quickly create admissions if they wish. The two top boxes are quick access cards to the admissions on the account, these will be for the previously accessed admissions, and will act like a bookmark of sorts. The only inputs are user interactions with graphical objects without these the user would be unable to navigate through the system.	**-Quick access to view admissions** (objective 21) Button 1 **-Logout** Button 2 **-View basic demographic** (objective 21) Section 3 **-Provide a starting screen** (links other tabs together) (objective 7) **-Add a new admission** (objective 7) Button 4 **-Clear notifications** Button 5 **-View demographic** (objective 21) Button 6



New Syptoms

New Symptoms

Go back

Please select the areas where the pain resonates from

☑ Neck ☑ Head ☑ Back ☑ Chest ☑ Stomach ☑ Arm ☑ Hand ☑ Pelvis ☑ Leg ☑ Foot

Please select any symptoms that you are experiencing

☑ Weight loss

☑ Nausea

☑ Fever

☑ Fatigue

On this page please enter all the symptoms you are currently experiencing this will help determine the ailment and select the most suitable consultant to help treat the problem, feel free to include as much as possible all information entered is confidential and encrypted.

If you have any symptoms that do no appear please use the boxes.

Symptom one

Symptom two

Symptom three

Symptom four

Please select the type of pain in the selected areas

☑ Chronic pains

☑ Acute pains

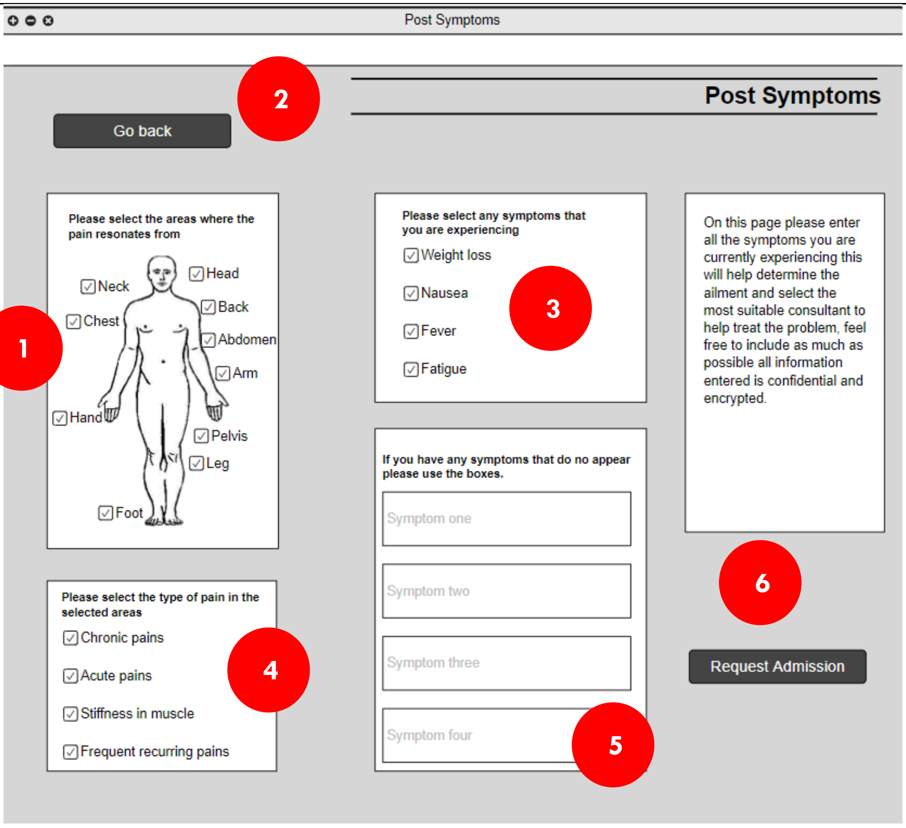
☑ Stiffness in muscle

☑ Frequent recurring pains

Request Admission

The design above is a predeveloped rendering of what the user interface will be and was created using Moqups. It is not the actual design that will be implemented as it will be subject to changes later on

Name of Design	Format	Consideration of Design & Purpose	Features Included
(initial)Symptom input	Screen	Before the user is to generate a new account on the system they first need to fill in the symptoms on the system to see if there is any need at all. Here they will enter what they are experiencing. This is very similar to the new admission section for a pre-existing patient, however the main difference being it does not include any of the account specific regions unique to the user. This is because it will need to be used by all new users. The main point is to allow users on the system to see what this user is feeling and therefore allow them to create an account with the system. The options available to them allow for a wide range of symptoms to be conveyed, along with a visual image to also help provide a basic aid. The purpose of this part of the symptom selection is to reduce the need of an advanced practitioner, this is a person who sees the patient before the consultant and deems the severity of the situation, this should help circumvent this need and would allow for a faster experience for the patient. The checkboxes will allow for quick fast selection of the areas affected without the need of going too into detail. All of these inputs will be heavily used in order to produce a basic diagnose on the system, without it the admission and patient and would not have a diagnosis and would defeat the purpose of the system, the different components just reduce the hassle in entering them.	**-Select areas the patient feels pain** (objective 1) Area 1 **-Select what type of pain the patient feels** (objective 1) Tick Box 4 **-Select symptoms they may be experiencing** (objective 1) Tick Box 3 **-Enter symptoms they have selectable** (objective 1) Text field 5 **-Go back to prior panel** (objective 7) Button 2 **-Request an admission to be generated** (objective 7) Button 6



Post Symptoms

Go back

Post Symptoms

Please select the areas where the pain resonates from

☑ Neck ☑ Head ☑ Chest ☑ Back ☑ Abdomen ☑ Arm ☑ Hand ☑ Pelvis ☑ Leg ☑ Foot

Please select the type of pain in the selected areas

☑ Chronic pains
☑ Acute pains
☑ Stiffness in muscle
☑ Frequent recurring pains

Please select any symptoms that you are experiencing

☑ Weight loss
☑ Nausea
☑ Fever
☑ Fatigue

If you have any symptoms that do no appear please use the boxes.

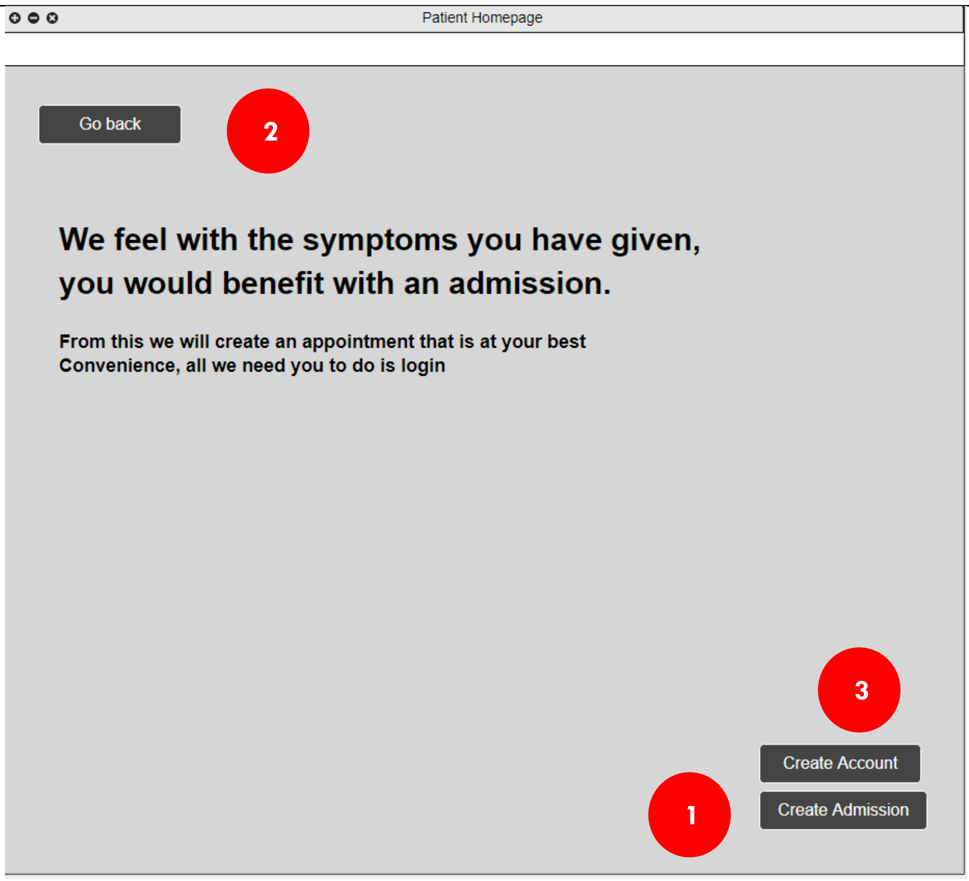
Symptom one
Symptom two
Symptom three
Symptom four

On this page please enter all the symptoms you are currently experiencing this will help determine the ailment and select the most suitable consultant to help treat the problem, feel free to include as much as possible all information entered is confidential and encrypted.

Request Admission

The design above is a predeveloped rendering of what the user interface will be and was created using Moqups. It is not the actual design that will be implemented as it will be subject to changes later on

Name of Design	Format	Consideration of Design & Purpose	Features Included
Post symptoms	Screen	After an admission has been discharged, the ailment may come back for any reason, however what the patient may feel may be different to what they experienced from last time. Because of this the system needs to be prepared to expect new symptoms from the patient, however these should not be immediately attached to the current admission as the patient may be experiencing a completely new ailment, just much closer to the end of a recent discharge. By doing this the chance of making the mistake of assuming it to be an old admission can be avoided. Again, the point of this page is to replace the advanced practitioner whose actual job is to determine severity before a patient comes to see the consultant, this helps remove the need for them. By having checkboxes this can allow for quick input without the need of manually entering symptoms on the system improving user experience. All of these inputs will be heavily used in order to produce a basic diagnose on the system, without it the admission and patient and would not have a diagnosis and would defeat the purpose of the system, the different components just reduce the hassle in entering them.	*Same features as other versions but also including:* **The new symptoms will be sent to the consultant informing them of the reinstatement request allowing them to decide if it is the same ailment or new.**

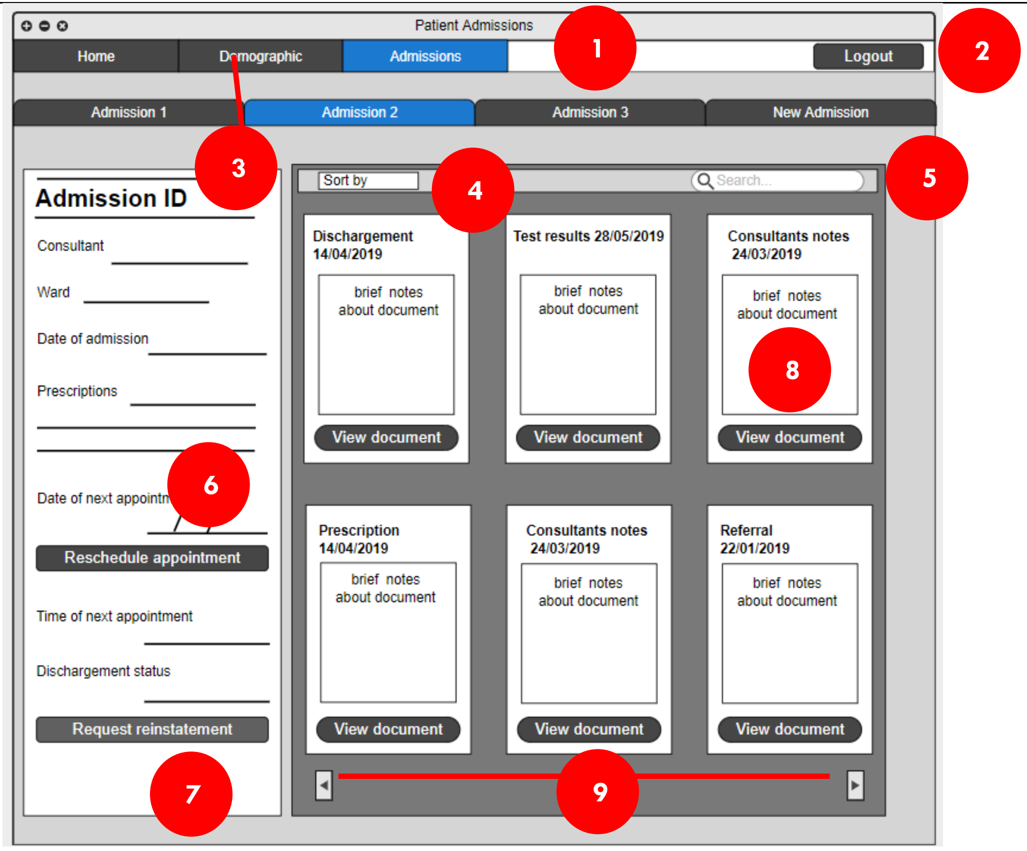


We feel with the symptoms you have given, you would benefit with an admission.

From this we will create an appointment that is at your best Convenience, all we need you to do is login

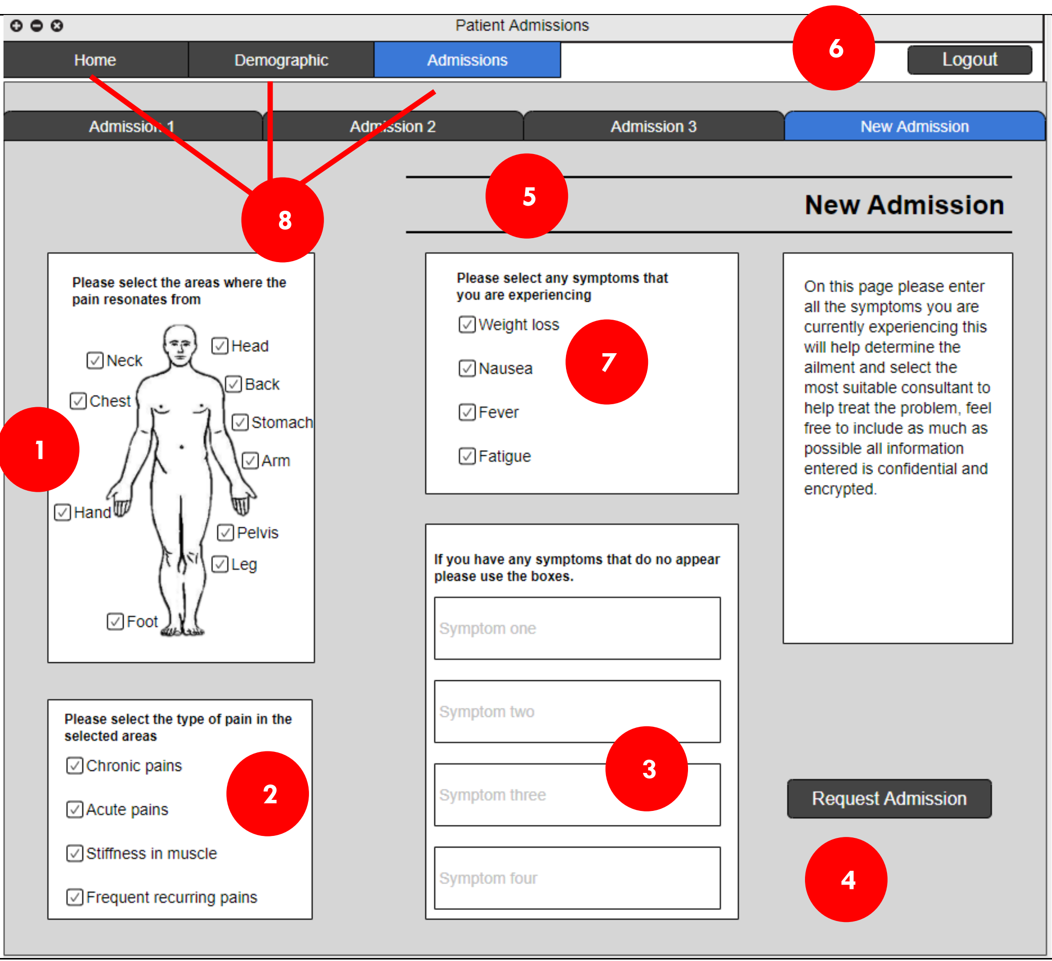
The design above is a predeveloped rendering of what the user interface will be and was created using Moqups. It is not the actual design that will be implemented as it will be subject to changes later on

Name of Design	Format	Consideration of Design & Purpose	Features Included
Symptom recommendation	Screen	The purpose of the page is to inform the user on if the symptoms they have truly merit an admission. If it is the option to create an admission, there is enough evidence to do so and one can be created. Else, they are informed that they cannot justify creating an admission and that creating one wastes resources. From this the user has to confirm they want the admission by clicking radio buttons and that their appointments may get postponed for any user who may have a higher priority over that admission. The purpose of the page is to act as a deterrent to the patient if the system thinks the admission is not of any major importance. The reason for this panel is to split the admissions into two sets, those who need the attention and then those who don't need help from Euxton. However no patient will be turned away, just importance is focused in those who need it. As there is only one feature to the panel mainly there doesn't need to be too much focus on the design, however this may be subject to change The only inputs needed are the interactions with the graphical objects without it the user would have no way to control the interface.	**-Can create an admission** (objective 4) Button 1 **-Go back to prior panel** (objective 7) Button 2 **-Login to account** (objective 6) Button 1 **-Create account** (objective 3) Button 3



The design above is a predeveloped rendering of what the user interface will be and was created using Moqups. It is not the actual design that will be implemented as it will be subject to changes later on			
Name of Design	**Format**	**Consideration of Design & Purpose**	**Features Included**
Admission homepage	Screen	This will be the homepage for an admission, it will hold all the information about an admission regarding the patient. A notable similarity between pages, is the left side panel which holds the general information this is also seen on the patient homepage, this in result creates a standard when it comes to displaying data to the user. The rest of the page is for the ability to view through documents and rescheduling appointments. The other panel on the page is the documents panel, this will contain all the documents attached to the patient's admission. The design I feel compliments the white of each individual document and does not feel too similar to the shade of grey to the buttons. On here they will be able to search for documents and order them into any way they would like. They will also be able to view the appointments along with the ability to generate new admissions. *(consultant's version will see a red button that will bring them onto the reinstatement page if requested by the patient)* The main purpose of the panel is to view all the documents in one place and find the desired one. To add to this they will also be able to (depending on the type of user) load the panel that holds the admissions	**-Quick access to view admissions** (objective 21) Button 1 **-Logout** Button 2 **-View basic demographic** (objective 21) Button 3 **-Provide a starting screen for the patient (links other tabs together)** (objective 7) **-Search Documents** (objective 26B)

		information. The use of the lower tab should allow a swift access between admissions as well. The only inputs that have been included for this page would be the search box and the selecting of sort order from the combo box, without these the user would have to manually search through every document until they have found their desired one. With the rest of the interactions of the system they are GUI focused features these will allow the user to move between different sections of the system to get to the feature shown on the button.	Text field 5 -**Sort Documents** (objective 26A) Combo box 4 -**Amend appointment** (objective 18) Button 6 -**Archive/ reactivate admission** (objective 34) Button 7 -**View individual documents** Button 8 -**Navigate Documents** Buttons 9

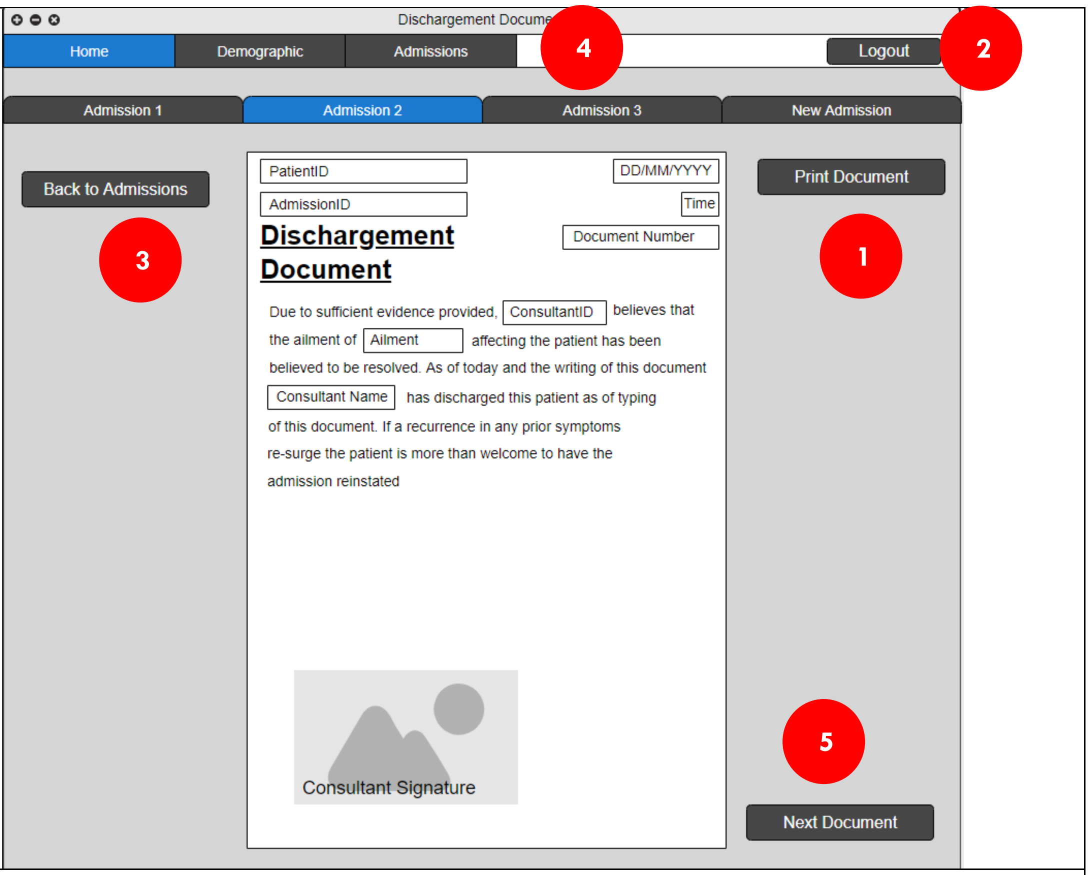


On this page please enter all the symptoms you are currently experiencing this will help determine the ailment and select the most suitable consultant to help treat the problem, feel free to include as much as possible all information entered is confidential and encrypted.

The design above is a predeveloped rendering of what the user interface will be and was created using Moqups. It is not the actual design that will be implemented as it will be subject to changes later on

Name of Design	Format	Consideration of Design & Purpose	Features Included
New Admission	Screen	This is the new admission page for the system, its purpose is to allow the user to enter the symptoms they are experiencing from this the staff entity will correctly allocate the most suitable consultant for them to receive treatment from(if unable to be done by the expert system). One unique aspect of the design is the use of a human silhouette this can allow for a more accurate description to locate where the pain is situated. Another key design choice was the number of selectable symptoms the check boxes, while having hundreds would remove the need for validation with the text fields having a handful of common symptoms and the ability to enter more rare symptoms will allow for efficiency more often. Finally, the colour scheme is still consistent following all the designs I had planned to use initially. The use of the human should allow for a more realistic approach to symptom entering to try and avoid the idea that the user is just filling in a form. This is to reduce the more basic and boring screen layout common in some of the documents I saw while inspecting documents. The input for this panel is very important as	**-Select areas the patient feels pain** (objective 1) Area 1 **-Select what type of pain the patient feels** (objective 1) Tick Box 2 **-Select symptoms they may be experiencing** (objective 1) Text field 3 **-Enter symptoms they have**

		it is needed for objective 2. Without it no diagnosis could be generated on any admission as there would be no symptoms to go from. While it could be seen as unnecessary to include another page when there will already be two more variants of the panel the idea to keep in mind is that this one is user specific with creating a new admission without any other factors affecting it, for instance requesting an admission without an account.	**but is not selectable** (objective 1) Tick Box 7 **-Request that an admission is created** (objective 2) Button 4 **-Select admissions using tabs** (objective 7) Button 5 **-Logout** Button 6 **-Select main pages using tabs** Button 8



Dischargement Document

PatientID DD/MM/YYYY

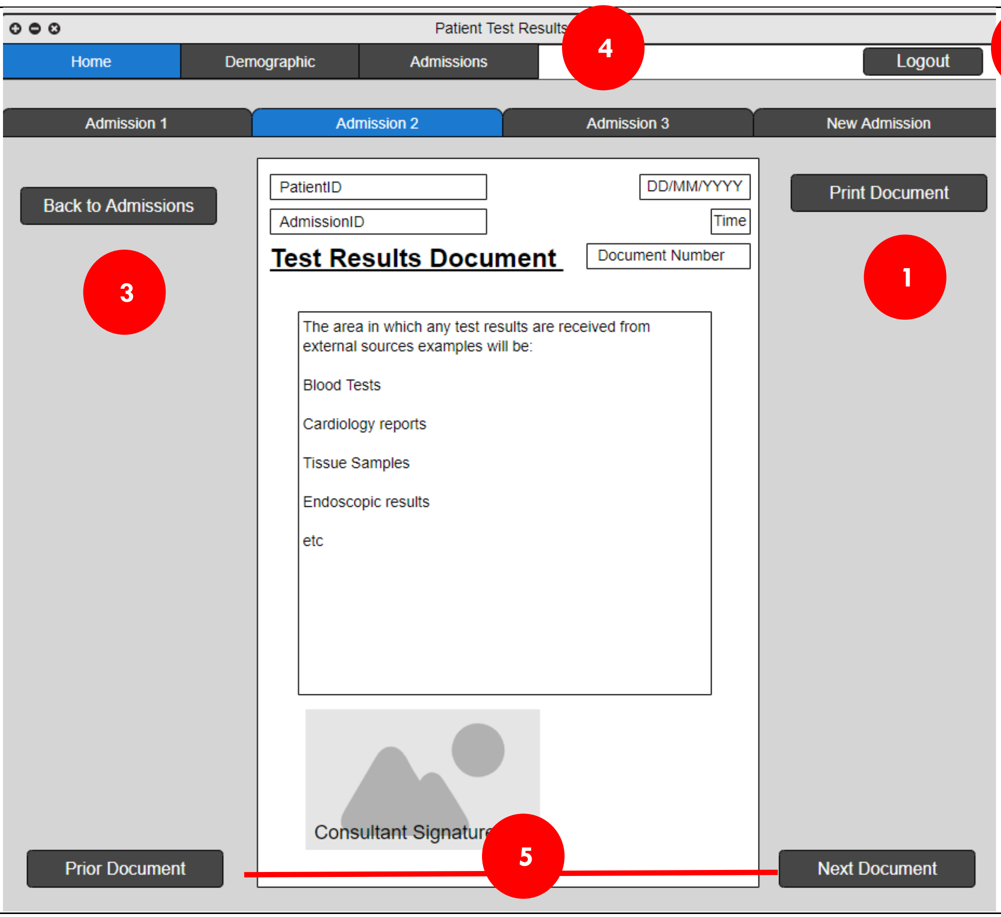
AdmissionID Time

Dischargement Document Document Number

Due to sufficient evidence provided, ConsultantID believes that the ailment of Ailment affecting the patient has been believed to be resolved. As of today and the writing of this document Consultant Name has discharged this patient as of typing of this document. If a recurrence in any prior symptoms re-surge the patient is more than welcome to have the admission reinstated

The design above is a predeveloped rendering of what the user interface will be and was created using Moqups. It is not the actual design that will be implemented as it will be subject to changes later on

Name of Design	Format	Consideration of Design & Purpose	Features Included
Dischargement document	Screen	This will be the document used to inform the user that the admission has been closed and that the patient has been discharged from the ward. On this page the user will also be able to move between other documents by selecting the *Next Document* button. If the patient wishes to print the document off, they will be able to and the document should resemble the one on the screen. I have designed the document to resemble some aspects of old documentation, like key information being at the very top of the document this will allow for easier viewings trying to see which patient and or admission this document resides in. Every discharge document will follow the exact same layout as this, therefore by having the document generalised it should only allow for a few entries of data to be required. This will be in the boxed areas, by reducing the amount required to be typed this will reduce workload as only a handful of words will be actually required. The only input for this panel will be for the navigation of the system however one key output would be the ability to print off documents, it was included here because it is the most ideal panel as it is unique to every specific document in each case.	**- View the document** (objective 20) **-Print the document** (objective 27) Button 1 **-Logout** Button 2 **-Go back to prior panel** (objective 7) Button 3 **-Quick access to view admissions** (objective 21) Button 4 **-Navigate documents on the admission** Button 5



The area in which any test results are received from external sources examples will be:

Blood Tests

Cardiology reports

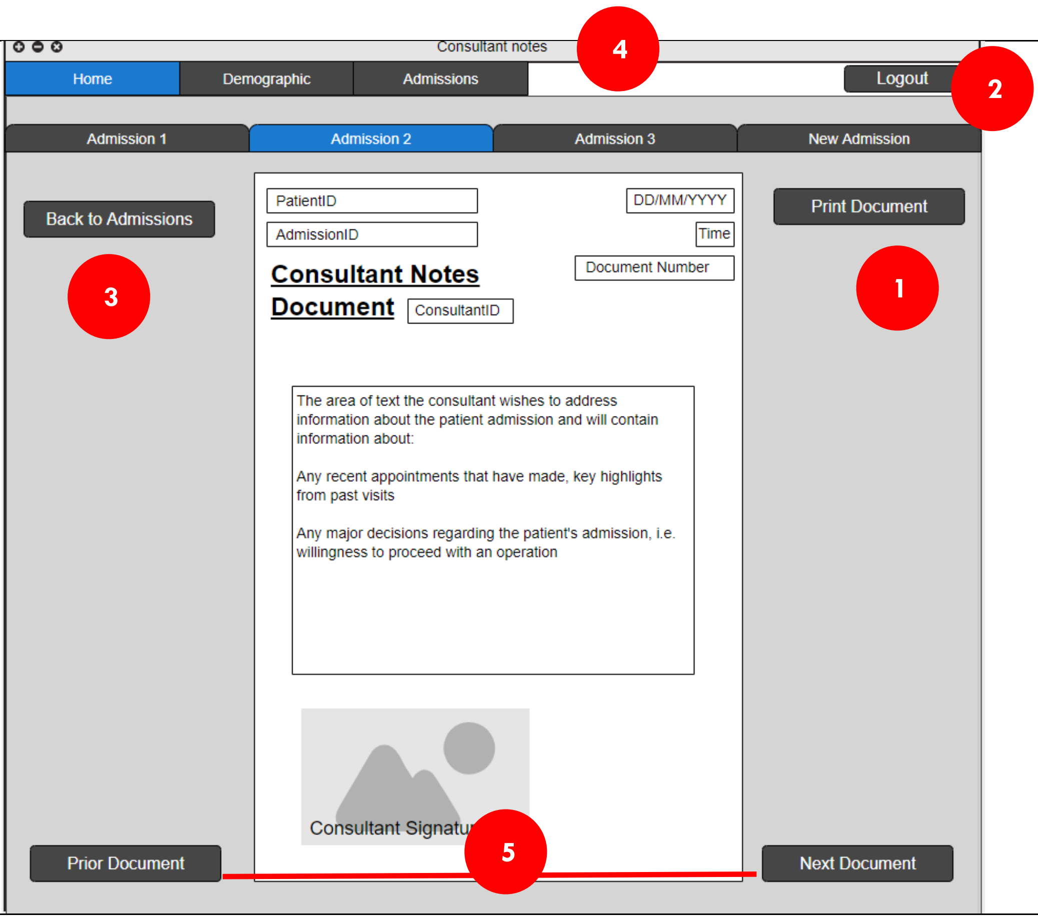
Tissue Samples

Endoscopic results

etc

The design above is a predeveloped rendering of what the user interface will be and was created using Moqups. It is not the actual design that will be implemented as it will be subject to changes later on

Name of Design	Format	Consideration of Design & Purpose	Features Included
Test results document	Screen	The purpose of this page is to allow for freedom of input regarding test results. This page still follows the format of the other documents but instead of using individual fields to hold data it will be a single variable. This is because the results in question will change quite a lot during the documents use and will also prevent the consultant from being constricted to single field entry. While the input will be more likely to be prone to errors this will allow the consultant to enter what they want. The design of the page is to be consistent with other designs of other documents such as available functions and operations like printing them of or returning back to the admission. The actual layout of the screen will follow similar button positions that are used throughout the system to also allow the user to remember one location on the screen. The only input for this panel will be for the navigation of the system however one key output would be the ability to print off documents, it was included here because it is the most ideal panel as it is unique to every specific document in each case.	**- View the document** (objective 20) **-Print the document** (objective 27) Button 1 **-Logout** Button 2 **-Go back to prior panel** (objective 7) Button 3- **Quick access to view admissions** (objective 21) Button 4 **-Navigate documents on the admission** Button 5



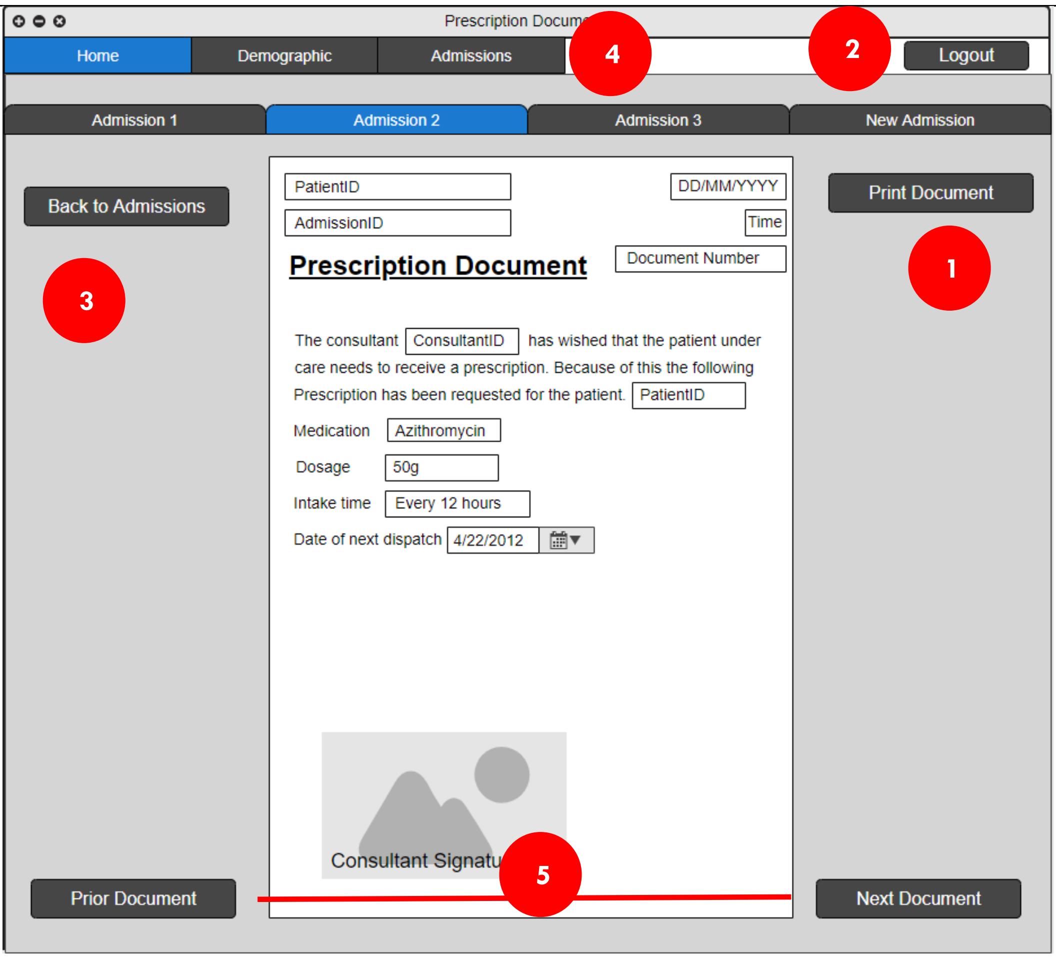
The area of text the consultant wishes to address information about the patient admission and will contain information about:

Any recent appointments that have made, key highlights from past visits

Any major decisions regarding the patient's admission, i.e. willingness to proceed with an operation

The design above is a predeveloped rendering of what the user interface will be and was created using Moqups. It is not the actual design that will be implemented as it will be subject to changes later on

Name of Design	Format	Consideration of Design & Purpose	Features Included
Consultant Notes document	Screen	The purpose of this page is to allow the consultant to enter documentation without being constrained by individual fields. While some are included these are for single small fields like the date and time, that are need so the document can be stored correctly and searched for. The design of the page will mimic the format of every other document to allow for a sense of consistency, this occurs through button positioning and where the document actually is on the screen. I intend on using a single variable to store the notes as these documents are very broad in use and can convey countless different messages such as for arranging upcoming appointments to a brief sentence informing the system the patient failed to turn up to an appointment. Because of this and to reduce the number of total documents I intend on leaving the field the way it is, while looking through the investigation I intended on reducing this issue, only now have I realised how hard it would be. The only input for this panel will be for the navigation of the system however one key output would be the ability to print off documents, it was included here because it is the most ideal panel as it is unique to every specific document in each case.	**- View the document** (objective 20) **-Print the document** (objective 27) Button 1 **-Logout** Button 2 **-Go back to prior panel** (objective 7) Button 3 **-Quick access to view admissions** (objective 21) Button 4 **-Navigate documents on the admission** Button 5



PatientID | DD/MM/YYYY
AdmissionID | Time

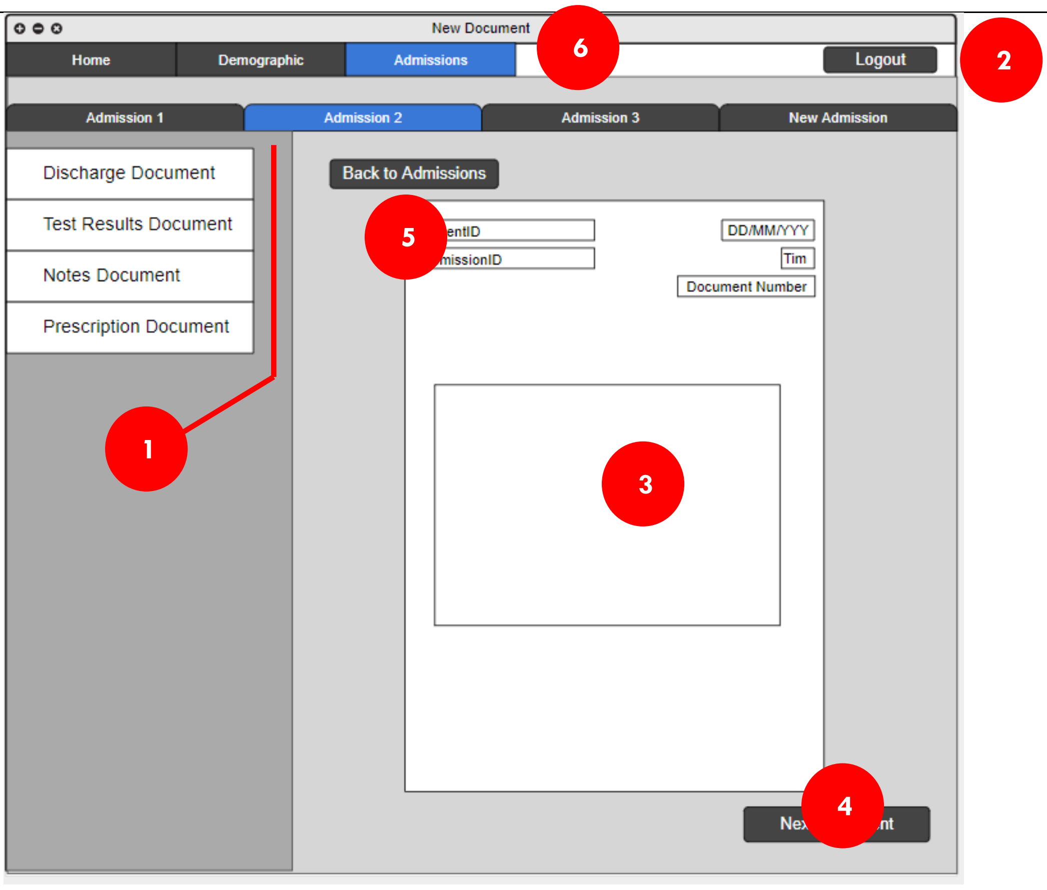
Prescription Document | Document Number

The consultant ConsultantID has wished that the patient under care needs to receive a prescription. Because of this the following Prescription has been requested for the patient. PatientID

Medication: Azithromycin
Dosage: 50g
Intake time: Every 12 hours
Date of next dispatch: 4/22/2012

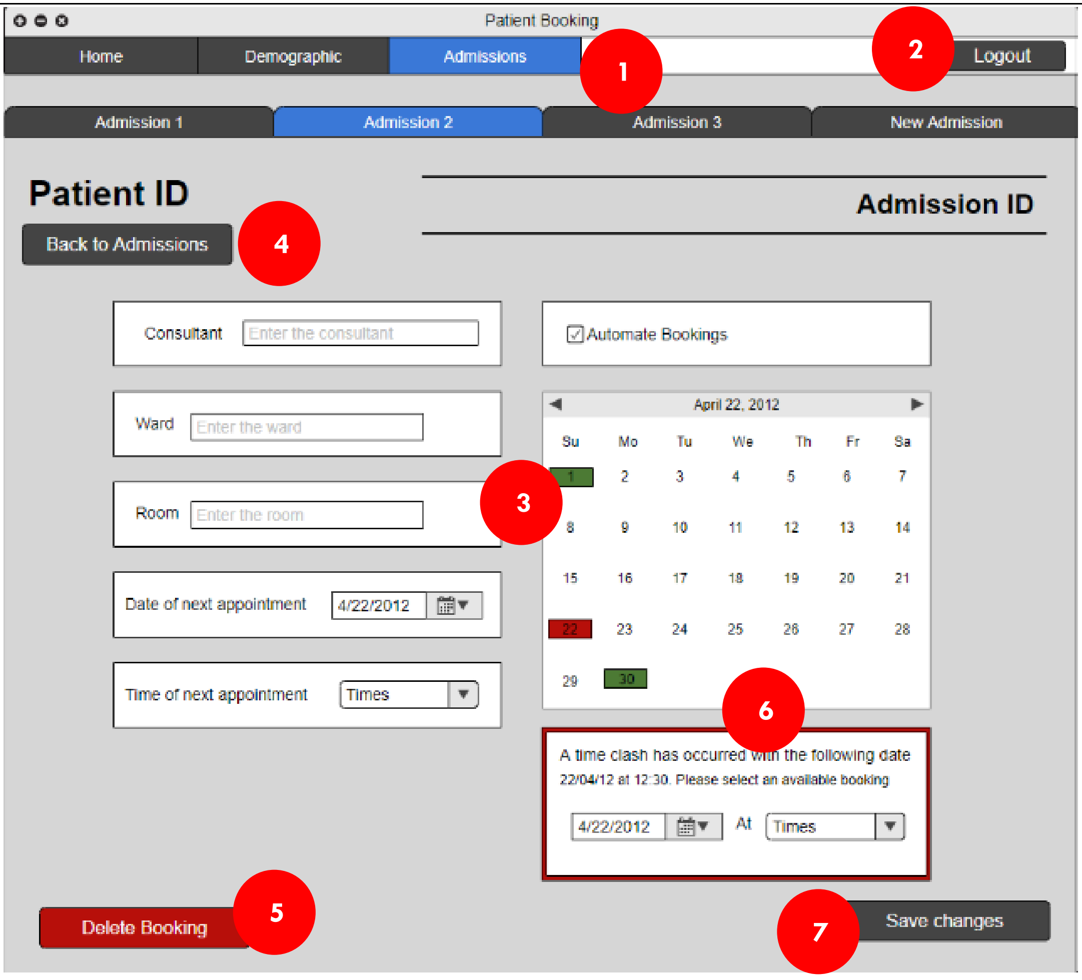
The design above is a predeveloped rendering of what the user interface will be and was created using Moqups. It is not the actual design that will be implemented as it will be subject to changes later on

Name of Design	Format	Consideration of Design & Purpose	Features Included
Prescription	Screen	This is the final variation of documents for the system. This will allow the consultant to enter a prescription for the patient to supply themselves with what the consultant thinks is best. Unlike the other documents the design is intended to limit user input. What is needed to be entered by the consultant is heavily constricted as these fields are very important and so will be validated accordingly. Besides this nothing different is apparent with the design, all the components and layout are the exact same to the rest of the designs for documentation. This will allow for a sense of coherence producing a better experience allowing for easier use over long periods of time. The only input for this panel will be for the navigation of the system however one key output would be the ability to print off documents, it was included here because it is the most ideal panel as it is unique to every specific document in each case.	**- View document** (objective 20) **-Print document** (objective 27) Button 1 **-Logout** Button 2 **-Go back** (objective 7) Button 3 **-Quick access to view admissions** (objective 21) Button 4 **-Navigate documents on the admission** Button 5



The design above is a predeveloped rendering of what the user interface will be and was created using Moqups. It is not the actual design that will be implemented as it will be subject to changes later on

Name of Design	Format	Consideration of Design & Purpose	Features Included
New Documents	Screen	The purpose of this screen is simple; it will allow consultants to write up documents onto the system, allowing for the old system at Euxton to become obsolete. As of now they have 4 options either dischargement document,(which will also discharge the patient as well) test results,(used to store information for test results), notes(used to hold notes about the admission) and finally prescription (which simply generates prescriptions and sets them to the account) The whole point is that here they can choose what document they want and enter the most important parts of the document. Minor aspects such as time date and patient are automatically assigned so don't need inputting. Once done the document will then be written into the patient's file so it can be accessed. The input for this panel is probably the most important on the system, without this no documents could be generated and then subsequently outputted in the form of writing them to file. This has been included here as the main panels focus is to use as a template for creating new documents on the system.	**-Amend information on the document** (objective 35) Text field 3 **-Create new documents of this type** (objective 35) Button 1 **-Logout** Button 2 **-Go back to prior panel** (objective 7) Button 5 **-Quick access to view admissions** (objective 21) Button 6 - Write document to file/Add document to list Button 4

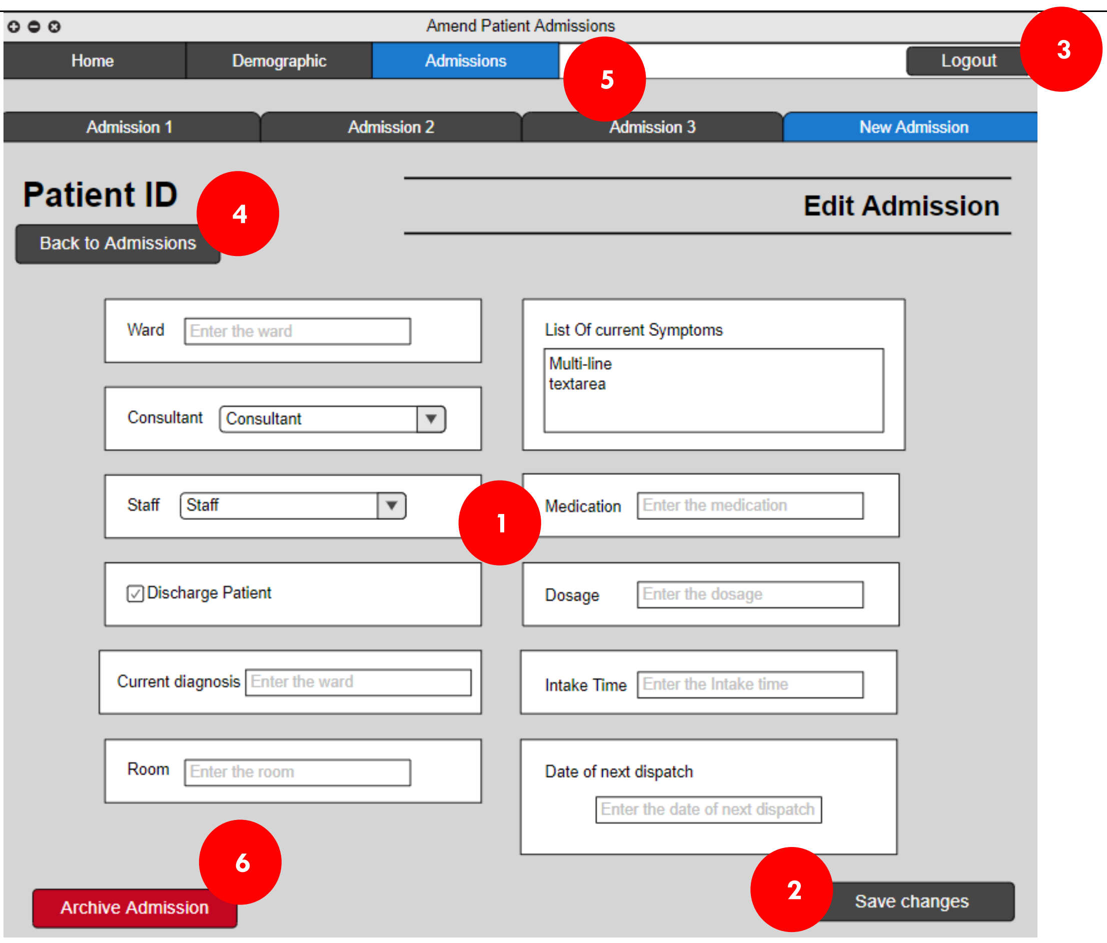


A time clash has occurred with the following date 22/04/12 at 12:30. Please select an available booking

The design above is a predeveloped rendering of what the user interface will be and was created using Moqups. It is not the actual design that will be implemented as it will be subject to changes later on

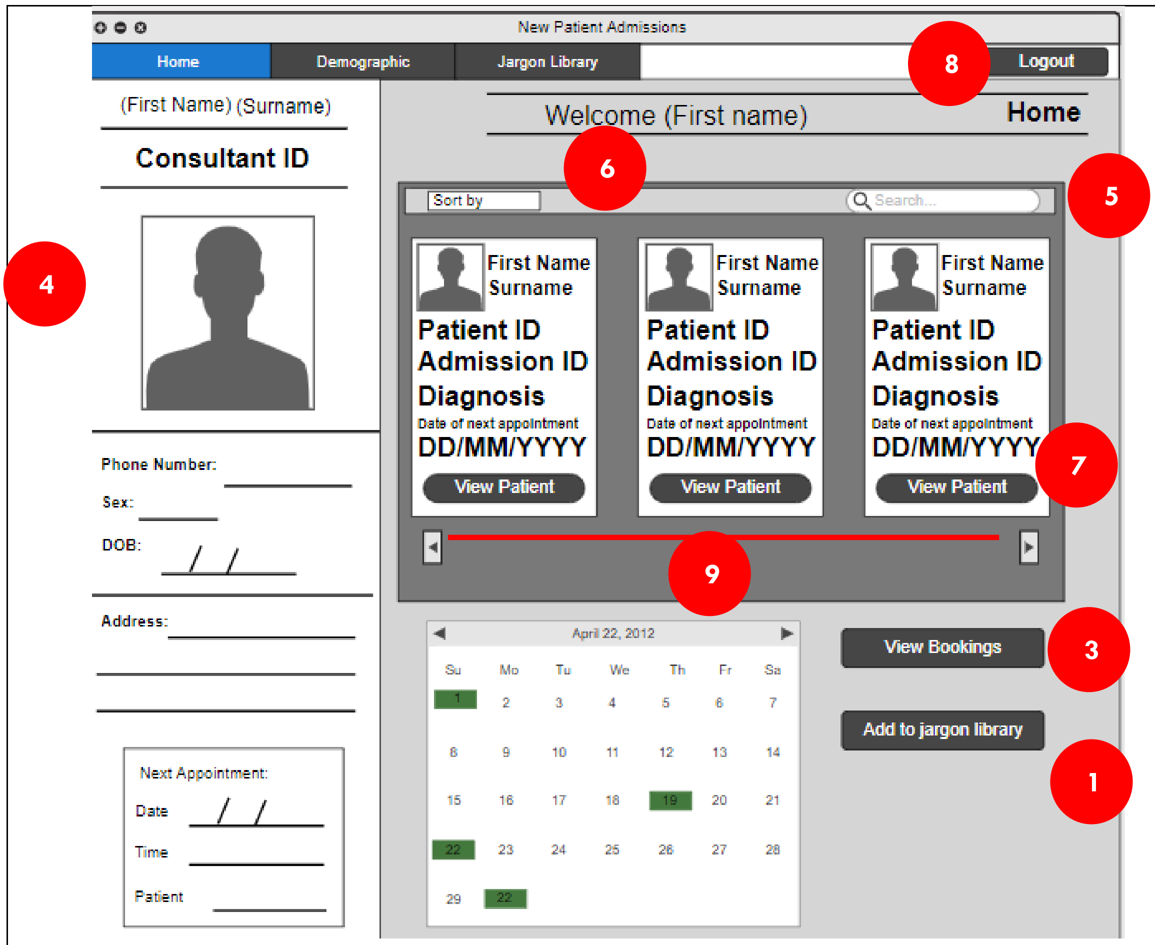
Name of Design	Format	Consideration of Design & Purpose	Features Included
Booking	Screen	The purpose of this screen is to allow the user(either the consultant or patient) to edit or view an appointment respectively. Where possible I have forced the user to reduce incorrect input by confining the available data entries to a drop-down box or calendar. For the design it is clear it still follows the colour scheme and layout used throughout the system. Points of interest would be the automate function radio button. This aspect will work by creating a new appointment and setting the consecutive ones every 6 months. However, it may be possible that an already existing appointment is already in place for this. To resolve this, I have create a notification on the screen informing them that this has occurred and highlighted the date in red to show when on the calendar this occurs. The sharp contrast in colour will show where this error has occurred. Finally, I have included a red button which will inform the user that	**-Open fields to amend them individually** (objective 22) Objects 3 **-Prompt to update a field if an automated one occurs at the same time as another one** (objective 7) **-View basic demographic** (objective 16)

		they can delete the appointment, however I intend to disable it, if it too soon to the appointment. The input for this screen is needed as it will be used to make sure bookings can be processed on the system, without it the ability to read bookings would be pointless. The inputs where included as it is the panel designated for editing documents on the system. The output is more important due the fact the input is needed in order for the instance of that booking to be written to file, without it no bookings would be held on the system	**-Delete the booking** (objective 18) Button 5 **-Logout** Button 2 **-Go back to prior panel** (objective 7) Button 4 **-Quick access to view admissions** (objective 21) Button 1 **-Alter the dates for the admission** (objective 18) Date picker 6 **-update and write over previous data** (objective 21) Button 7



The design above is a predeveloped rendering of what the user interface will be and was created using Moqups. It is not the actual design that will be implemented as it will be subject to changes later on

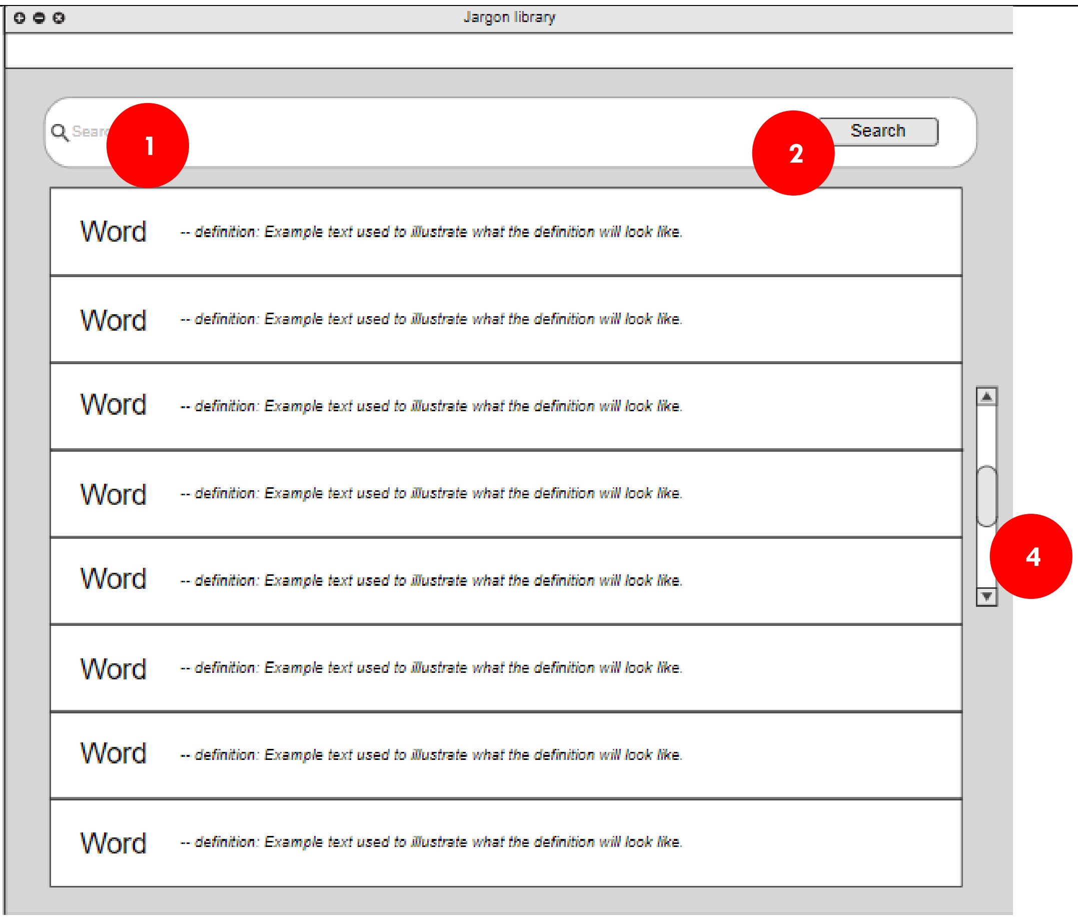
Name of Design	Format	Consideration of Design & Purpose	Features Included
Amending admission	Screen	This page is similar to the edit admission page with respect to functionality it will allow the consultant to enter admissions onto the system. the design also mimics the layout too with small minor fields to the left and leaving the larger data fields like the list of symptoms to be positioned on the right. Also, the colour scheme was also important due to the fact the most important button is highlighted for the consultant. In addition, as of now I have excluded any presence checks on any of the fields for this section, as it will be a new user the consultant may not possess enough information to make an informed decision on what medication they may need. Similar to the booking panel, the need for the fields as input on this panel is very important as the system should have the ability to accept new data from the consultant, to add to this again the outputs are also very important without them the user would be unable to see their data along with the fact it could never be written to file, the purpose of this panel is let these processes occur.	**-Open fields to amend them individually** (objective 18) Objects 1 **-Overwrite old versions of the data once saved** (objective 18) Button 2 **-Logout** Button 3 **-Go back to prior panel** (objective 7) Button 4 **-Quick access to view admissions** (objective 21) Button 5 **-Archive an admission** (objective 34) Button 6



The design above is a predeveloped rendering of what the user interface will be and was created using Moqups. It is not the actual design that will be implemented as it will be subject to changes later on

Name of Design	Format	Consideration of Design & Purpose	Features Included
Consultant homepage	Screen	The purpose of this screen is to welcome the consultant when they log onto the system. When they enter they will see the most recent patient's file they accessed along with any dates they do not have any appointments(the days marked in green). The design of the home-screen is also important, as it will allow for lots of information to be viewed by the consultant without it seeming too cluttered. I think this is achieved with the patient cards. They are compact and display lots of information, but each one is clear and shows enough information to understand which patient it is. Finally, there are there the main headings of homepage demographic and jargon library. These will allow fast access to the respective pages. The design also looks familiar to the patient and staff homepages as it will allow for a familiar feel when moving through the pages, however the slight differences will allow each one to be distinguishable.	**Load the add terms to jargon library panel** (objective 7) Button 1 **Logout** Button 2 **View upcoming appointments** (objective 19) Button 3 **View Basic demographic information** (objective 20) Area 4 **Search patients**

		The input data is needed to allow the user to either search the desired patient on the system or sort them in the desired order, either way the output from these will help the consultant find the desired patient as soon as possible, without these they would have to search an unordered list of all their patients which would take a while.	(objective 15) Text Area 5 **Sort patients** (objective 14) Combo box 6 **Select patients** Button 7 **-Logout user** (objective 6) Button 8 **-Navigate between patients** Button 9



Word -- definition: Example text used to illustrate what the definition will look like.

Word -- definition: Example text used to illustrate what the definition will look like.

Word -- definition: Example text used to illustrate what the definition will look like.

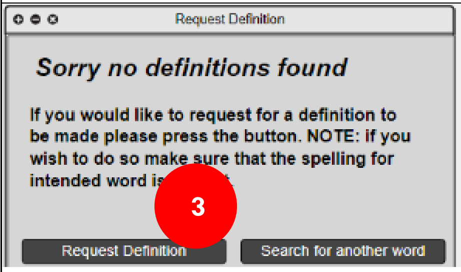
Word -- definition: Example text used to illustrate what the definition will look like.

Word -- definition: Example text used to illustrate what the definition will look like.

Word -- definition: Example text used to illustrate what the definition will look like.

Word -- definition: Example text used to illustrate what the definition will look like.

Word -- definition: Example text used to illustrate what the definition will look like.



Sorry no definitions found

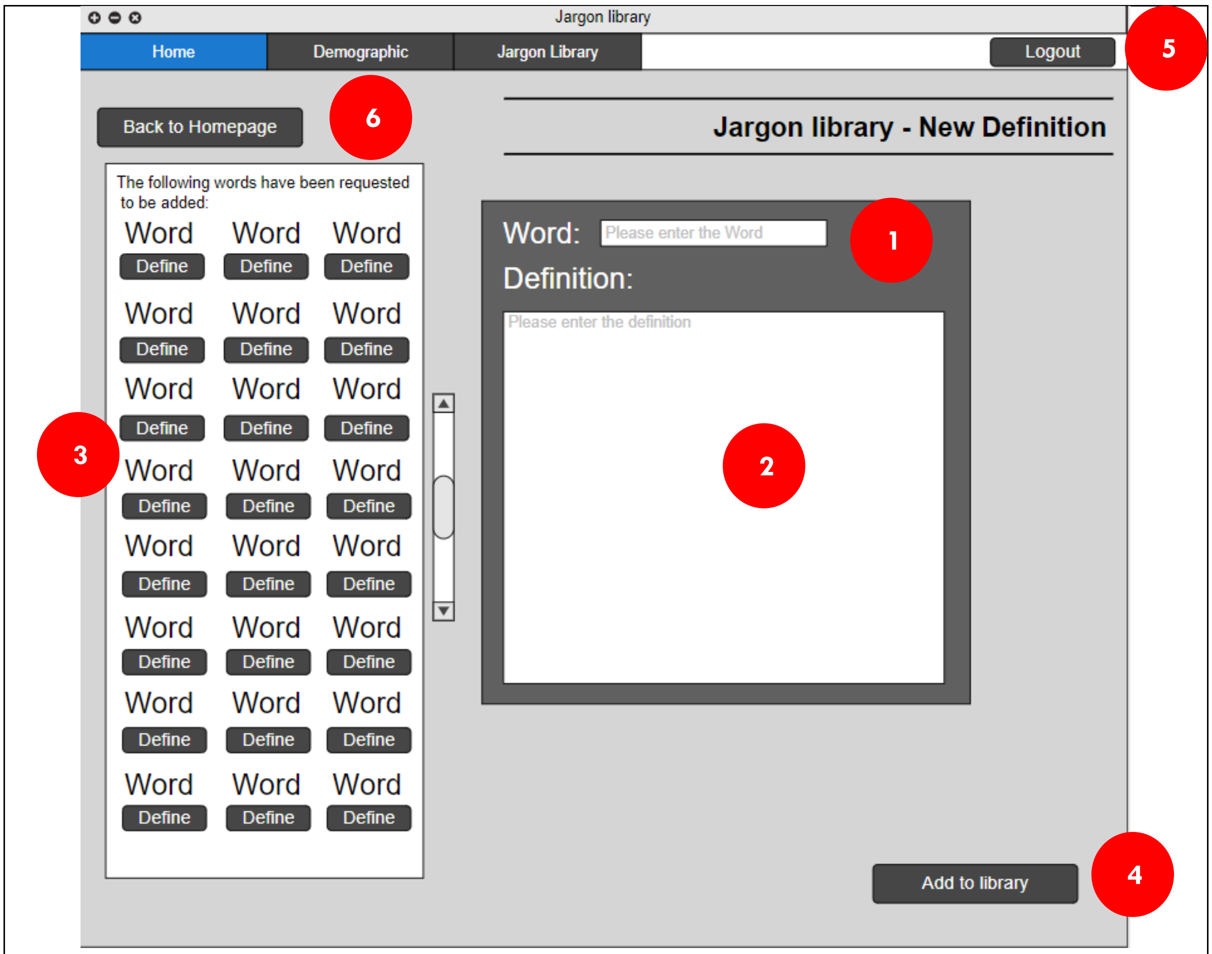
If you would like to request for a definition to be made please press the button. NOTE: if you wish to do so make sure that the spelling for intended word is t.

Popup Menu

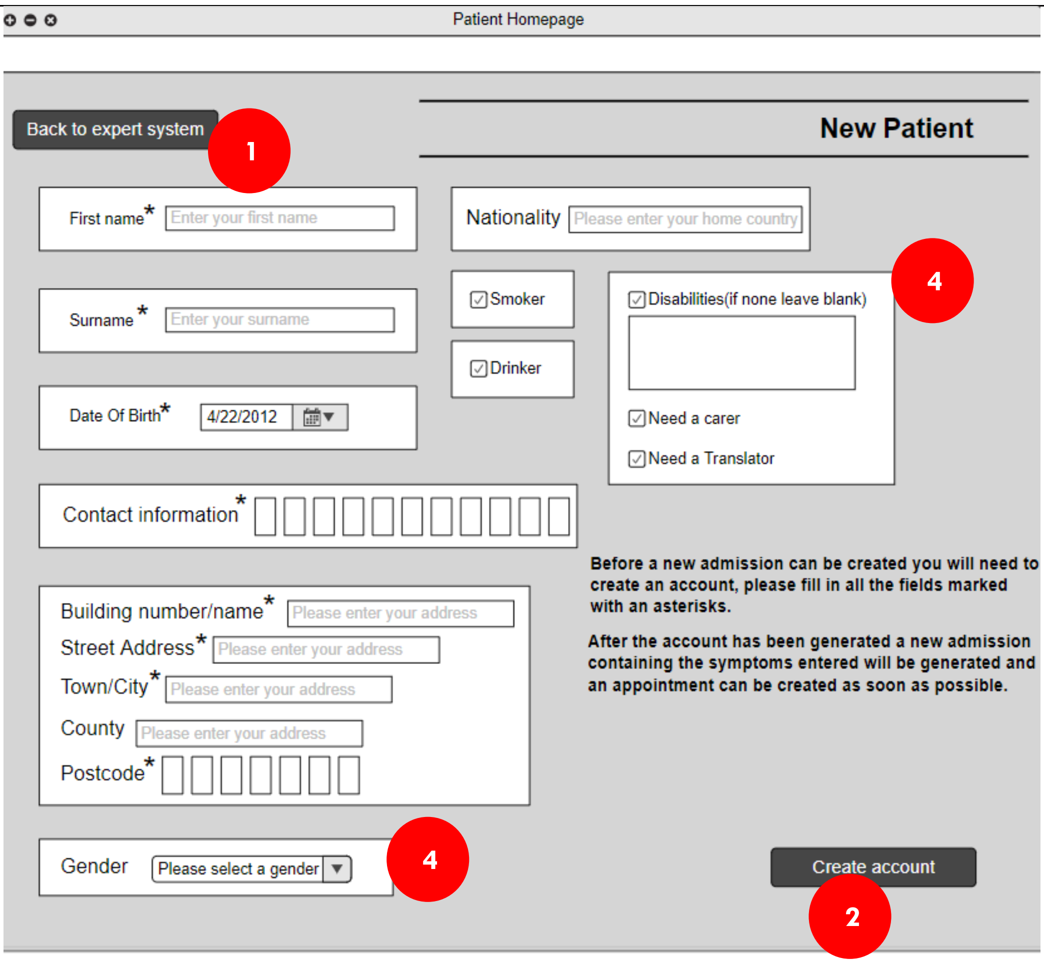
As it is possible that the user may enter a word that is not on the system, to inform the user that no definition exists the popup will be shown to the patient to inform them that no definition was found. However the text underneath the notice is to inform the user that they can request that the word they entered can have a definition written to it,

The design above is a predeveloped rendering of what the user interface will be and was created using Moqups. It is not the actual design that will be implemented as it will be subject to changes later on

Name of Design	Format	Consideration of Design & Purpose	Features Included
Jargon library	Screen	The design is very basic which is intentional, the whole purpose is to display a single word along with a definition. That's it, as this part of the system is subsidiary it is not essential to have it take up the whole screen real estate and so it can have a small window resulting in necessary components being shown The purpose is only to allow the user to enter a word they do not know the meaning of and then receive a definition. The input data is important as it will be used to search the list of items to query whether or not it exists in the database. The output is even more important as it will return the definition this will show the user the meaning of the word. Without it the panel would be pointless.	**-Search for word** (objective 38) Text Area 1 Button 2 **-Request for it to be added** (pop up)** Button 5 **-Navigate the page using a scroll bar (pop up)** Button 4



The design above is a predeveloped rendering of what the user interface will be and was created using Moqups. It is not the actual design that will be implemented as it will be subject to changes later on			
Name of Design	**Format**	**Consideration of Design & Purpose**	**Features Included**
Page to add new definitions	Screen	The purpose of this screen is to allow the consultant to enter a definition to the word and then add it to the library of words to search from. This can be achieved through two ways, either they enter the word they want to define or they select the word from the user wanted words. Unlike the actual library window to search for a definition, this page will be accessed from the actual system from the consultant's homepage, because of this it will follow the format of the rest of the system and will include main aspects such as a main section. To add to this the design is also important with respect to the size of the definition text box, this is because the aim of the jargon library is to provide a simple definition, therefore by constricting the size of the text box a small definition will be needed resulting in a concise definition being included. The input of this data is important as it will allow the consultants the ability to generate new terminology on the system, however the output will be more important as this will be needed to make sure that it is written to file and saved.	**-Enter word** Text field 1 **-Enter definition** Text field 2 **-Select pre requested words** Buttons 3 **-Add word to library** (objective 39) Button 4 **-Logout** Button 5 **-Go back to prior panel** (objective 7) Button 6

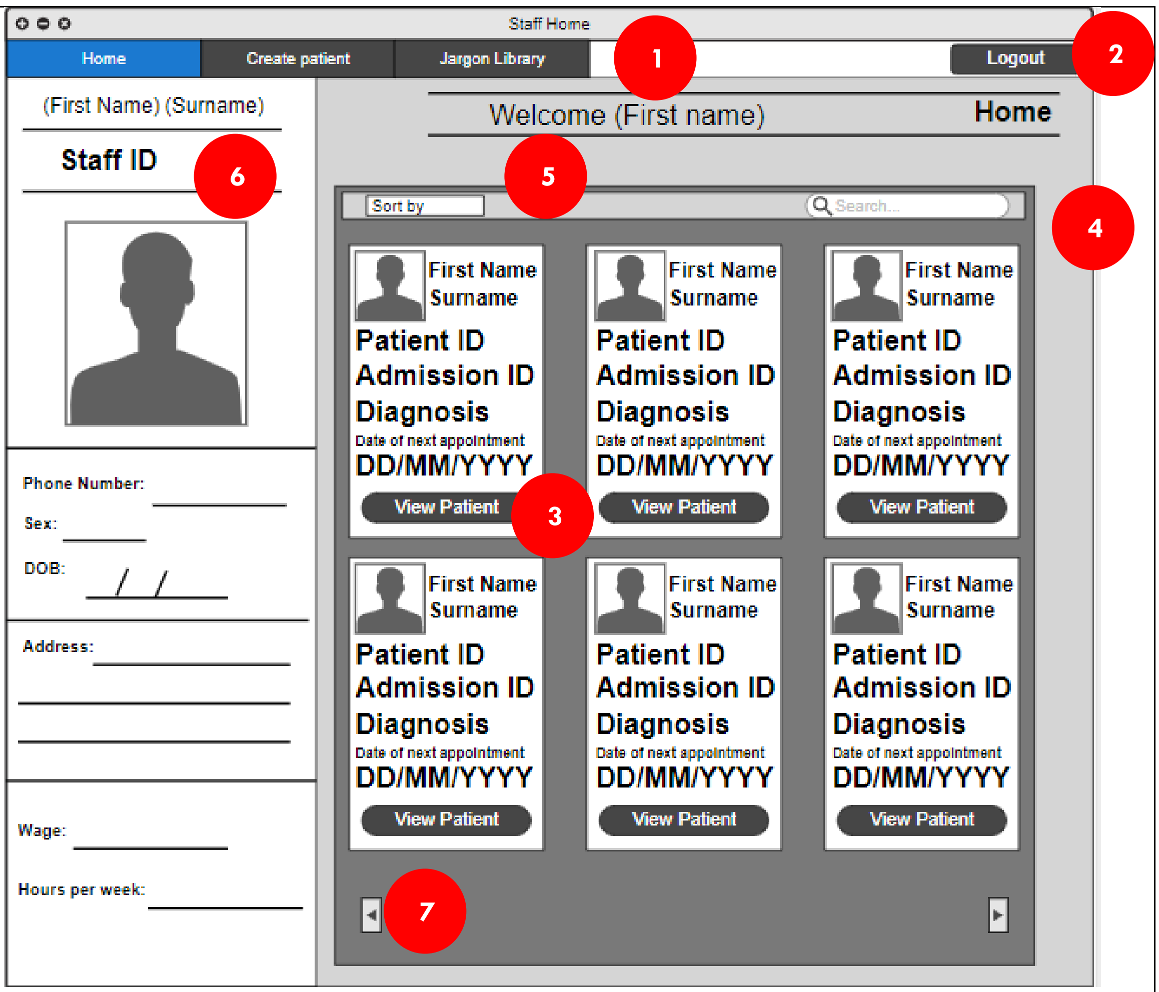


Before a new admission can be created you will need to create an account, please fill in all the fields marked with an asterisks.

After the account has been generated a new admission containing the symptoms entered will be generated and an appointment can be created as soon as possible.

The design above is a predeveloped rendering of what the user interface will be and was created using Moqups. It is not the actual design that will be implemented as it will be subject to changes later on

Name of Design	Format	Consideration of Design & Purpose	Features Included
New patient	Screen	The purpose of this screen is to allow the new user to enter their demographic information onto the system, just after entering their symptoms onto the system. The fields they will enter will be limited to drop down boxes where necessary to try and reduce any error from input. The layout is very familiar to pages where data is edited such as the booking screen. This is because I feel it is a clear design, as the use of individual boxes for fields highlight what needs to be entered onto the system. Also, to help the user a few paragraphs are included to inform the user on what is happening and what needs to occur. Finally, as a point to mention a large space is left for the field to describe disabilities as a patient may have more than one thing they have that needs to be saved. This input data has been included on the panel because it will be needed in order to make sure that all the fields have been covered on the system, without it there would be discrepancies with fields being constantly missing. However the output is more important as this will be used to make sure that the information is actually written to file.	**-Return back to the expert system** Buttons 1 **-Generate a new patient/write account to file/add them to list of patients** (objective 3) Buttons 2 **-Enter the fields required for a user** Buttons 3 **Selecting items through combo boxes** Comb boxes 4

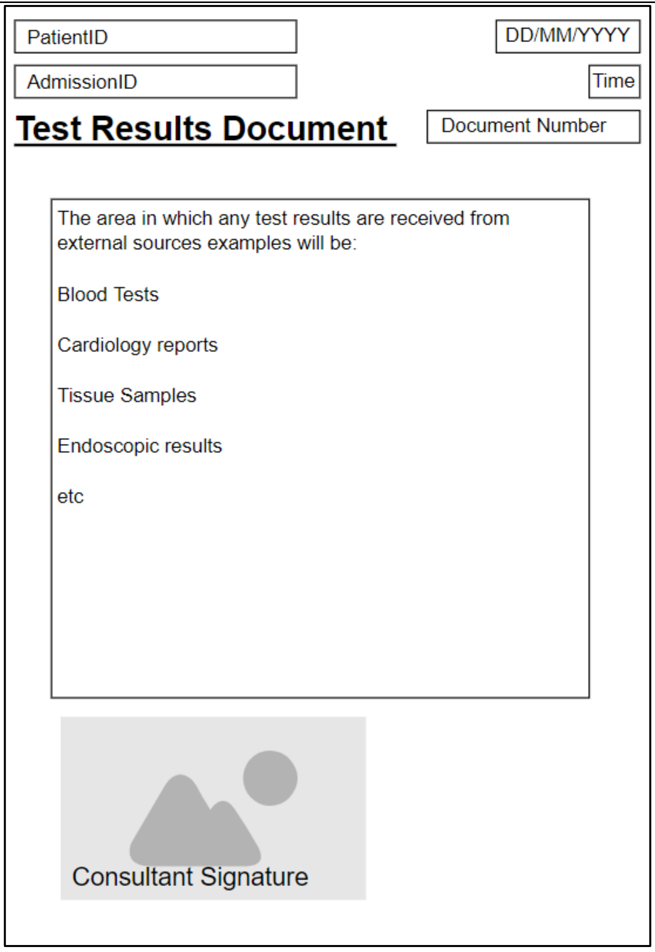


The design above is a predeveloped rendering of what the user interface will be and was created using Moqups. It is not the actual design that will be implemented as it will be subject to changes later on

Name of Design	Format	Consideration of Design & Purpose	Features Included
Staff Homepage	Screen	The purpose of this screen is to greet the staff user when they log onto the system. This is done by displaying their most recent patients they accessed. From here she can also search and sort for the patients they need to access. As their use of the system is limited and focused entirely around individual patients the majority of the screen is focused on accessing each of them. The screens design also mimics that of the patients and consultant's homepage, this creates consistency throughout the system and does not isolate any of the users from design aspects of the system making the general experience seem similar between different types of users. The input for this panel is important due to the fact for the search bar the data will be used to query the database and make sure that if the patient is on the system, they are returned. In addition the input of the desired fields is also important as it will allow the patients to be sorted into the correct order. The outputs are the results of the inputs and are more important as otherwise not retuning the patient would just make the feature pointless similar to the sort.	**use jargon library** (objective 7) Button 1 **Logout** Button 2 **View file** (objective 16) Button 3 **Search patient** (objective 15) Text field 4 **Sort patient** (objective 14) Combo box 5 **View details** Area 6 **Sort patient** (objective 14) Button 7

Paper Documents

While the rest of the systems outputs are virtual n the java environment the outputs that are physical will be the four printable documents. These documents follow the exact formatting of the documents on the screen to allow the user experience consistency between the virtual and physical aspects of the system. This can be done due to the fact while designing the documents for the virtual system the document was specifically formatted to resemble the dimensions of an A4 piece of paper.


PatientID

AdmissionID

DD/MM/YYYY

Time

Test Results Document

Document Number

The area in which any test results are received from external sources examples will be:

Blood Tests

Cardiology reports

Tissue Samples

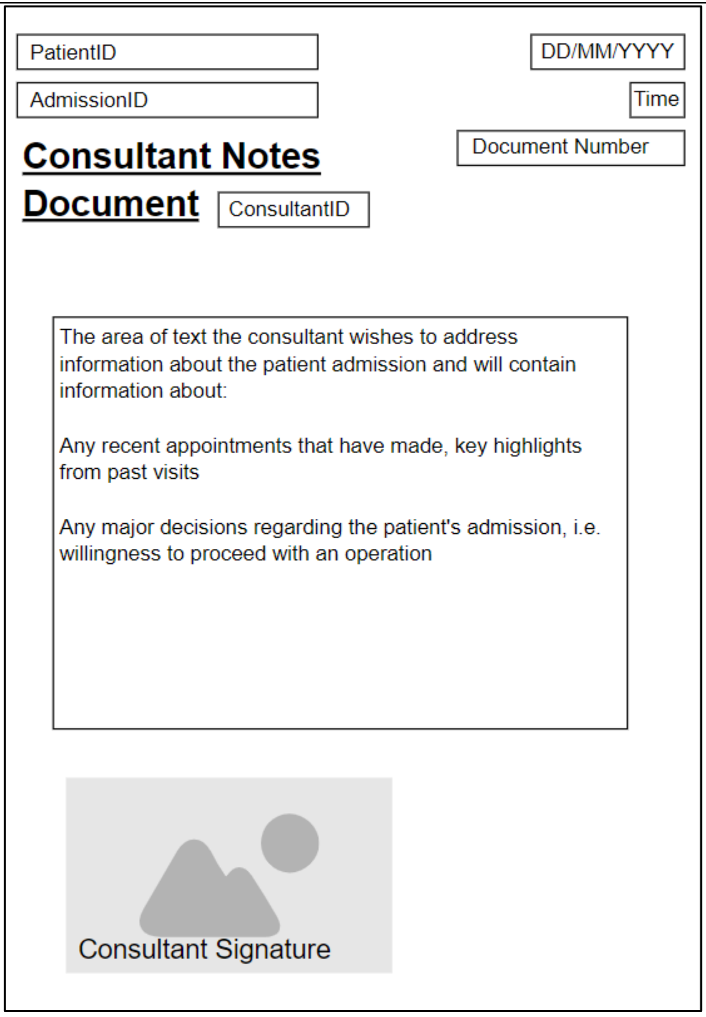
Endoscopic results

etc

Consultant Signature

The design above is a predeveloped rendering of what the user interface will be and was created using Moqups. It is not the actual design that will be implemented as it will be subject to changes later on

Name of Design	Format	Consideration of Design & Purpose	Features Included
Test results Document	Paper	The design of this output is to allow test results to be printed of for archival and to allow physical copies to exist. This will allow the patient to have the ability to either have all their documents virtually on the system, all printed of, or the option to have a select few. While the ability to print of the document can be used an unlimited amount of times it will not remove the document once it is printed. The design of the document itself is straight forward, all the relevant fields are located where you would imagine them to be. When the document is printed what will happen is that also the date and time of printing as well as the original date and time will also be printed to allow for consistency and allow for correct storage.	Represents a physical copy of the virtual document on the system.


PatientID

AdmissionID

DD/MM/YYYY

Time

Document Number

Consultant Notes Document ConsultantID

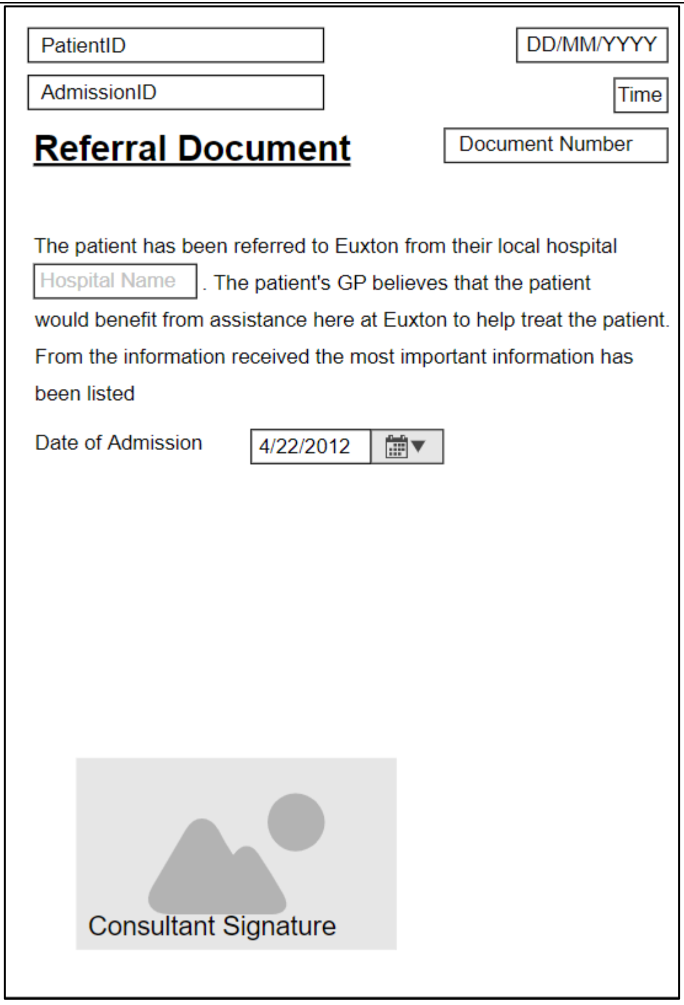
The area of text the consultant wishes to address information about the patient admission and will contain information about:

Any recent appointments that have made, key highlights from past visits

Any major decisions regarding the patient's admission, i.e. willingness to proceed with an operation

The design above is a predeveloped rendering of what the user interface will be and was created using Moqups. It is not the actual design that will be implemented as it will be subject to changes later on

Name of Design	Format	Consideration of Design & Purpose	Features Included
Consultant notes	Paper	Similar to the other documents this is a physical version of the one on the virtual system. Its design and layout was based on the original documents utilised in the original document inspection with the credentials on the top of the document, which also mimics a letter. For this particular document it will be used to store whatever the consultant writes about the admission the only difference is that the tillite will be unique in which the consultants id will be present. The whole point is to allow an offline version for legal and other safekeeping reasons. While the document holds no functionality of the system it will be used to indicate that a certain action has occurred on the system.	Represents a physical copy of the virtual document on the system.


PatientID

AdmissionID

DD/MM/YYYY

Time

Referral Document

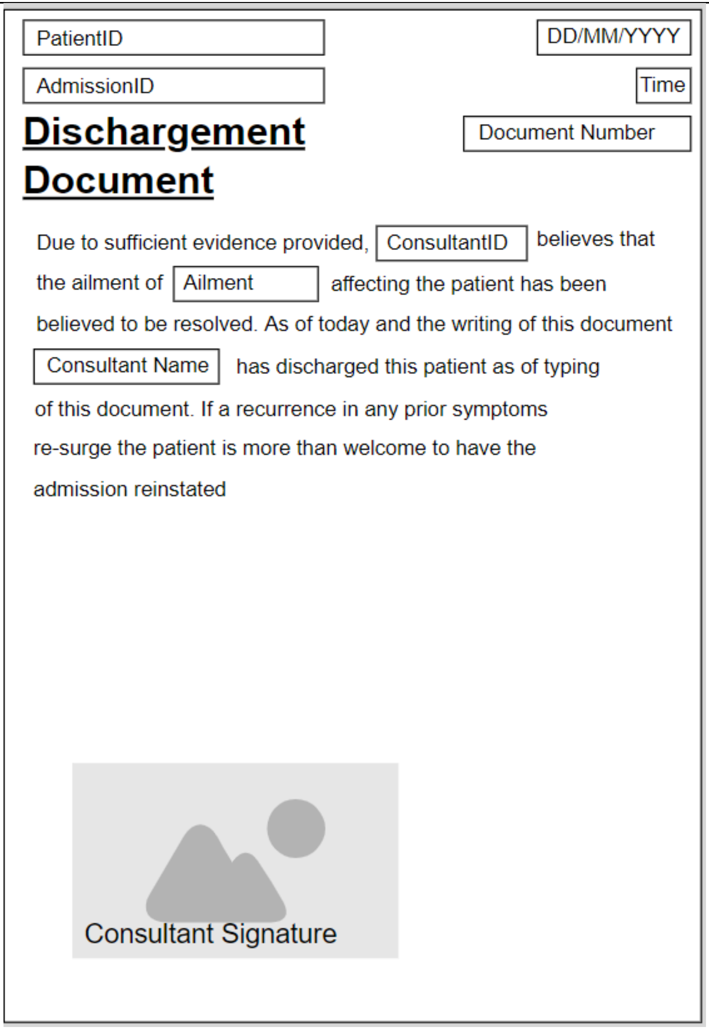
Document Number

The patient has been referred to Euxton from their local hospital Hospital Name . The patient's GP believes that the patient would benefit from assistance here at Euxton to help treat the patient. From the information received the most important information has been listed

Date of Admission 4/22/2012

The design above is a predeveloped rendering of what the user interface will be and was created using Moqups. It is not the actual design that will be implemented as it will be subject to changes later on

Name of Design	Format	Consideration of Design & Purpose	Features Included
Referral Document	Paper	Similar to the other documents this is a physical version of the one on the virtual system. Its design and layout was based on the original documents utilised in the original document inspection with the credentials on the top of the document, which also mimics a letter. The purpose of this document similar to the discharge document is to indicate at a current moment of time this admission was referred to the system. While the document holds no functionality of the system it will be used to indicate that a certain action has occurred on the system. To add to this the document could be utilised for proof of illness like a "doctors note". The point of the signature while not the consultant actually writing it, it would be used to signify that they accept the responsibility for the documents use.	Represents a physical copy of the virtual document on the system.



PatientID

AdmissionID

DD/MM/YYYY

Time

Document Number

Dischargement Document

Due to sufficient evidence provided, ConsultantID believes that the ailment of Ailment affecting the patient has been believed to be resolved. As of today and the writing of this document Consultant Name has discharged this patient as of typing of this document. If a recurrence in any prior symptoms re-surge the patient is more than welcome to have the admission reinstated

The design above is a predeveloped rendering of what the user interface will be and was created using Moqups. It is not the actual design that will be implemented as it will be subject to changes later on

Name of Design	Format	Consideration of Design & Purpose	Features Included
Dischargement Document	Paper	Similar to the other documents this is a physical version of the one on the virtual system. Its design and layout was based on the original documents utilised in the original document inspection with the credentials on the top of the document, which also mimics a letter. With the document the whole point is to allow the patient the ability to print off documents at their own home, this could be used for legal reasons to hold documents for proof of treatment, similar to how bank statements are kept. With the dischargement document as there are only a few fields to fill in I think it would be possible to have the ability to automatically fill these in and discharge the patient in the same process.	Represents a physical copy of the virtual document on the system.

3.3 Files & Data Structures

3.31 Methods of Access

With all the entities on the system the classes involved span from an object hierarchy utilising inheritance to accumulate the majority of their attributes and so the entity actually inherits many more attributes and methods from other classes these will be what the majority of the processes and properties will come from. To indicate this every entity will contain the main class where all entity specific methods will be declared along with any subsequent parent classes.

Entity & Classes	Data structure	Method of Access
Staff *Staff.java* *StaffList.java* **Inherits:** *Employee.java* *User.java*	**Array –** Used to hold them as a collective memory and specifically when accessing the system as a management entity when trying to select a staff entity **Record –** The structure to hold the user's data while logged in as the user, if information needs to be changed about the entity it will be the first to see the change.	Expected Number of records: **50** *There needs to be few staff entities present on the system as they will be in charge of handling old documents and appointments. It will also be up to them to spot discrepancies on the system so by having many users present on the system will reduce the overflow of work that would occur. I believe 50 will be sufficient in controlling the system. However if in development the quantity needs to change it can.* Method of file organisation: **Indexed Sequential** *I feel indexed sequential is the most suited for file organisation for this entity due to the fact that while the number of staff in comparison won't be the largest collection of entities on the system, they will be a dynamic and changing list and as the system will need to search for them constantly: for instance logging in, I only feel that having a direct access to this entity in the data structure is beneficial with no doubt. As the structure will be using indexes the most suitable search will be with a **binary search**. However as a result of the binary search and the fact the order has to be constantly managed it means that a sorting algorithm will need to be put in place to reorder the staff into a desired list. But due to the relatively small size of the list I have no worries in using a* **bubble sort** *to organise the array of data. As while slightly slower than insertion it will only need to take a few passes to get the single data into order, and as the process will likely only be used to add and remove one member of staff occasionally I see no issues arising with speed.* Use: **Permanent** *The use for this entity will be permanent therefore it needs to be ordered. While no actual data will be deleted it will instead being archived. Records or new staff may be added through the management entity and as described added to the correct location in the file through staffID. Therefore it's important that the information held is always correct and up to date.* Programming time required: **Some** *As staff are not the quintessential entities on the system development for these users will not take up most of the time. This is also due to the file organisation used as sequential is very easy and fast to develop. As the number of actions is the second smallest this helps support the idea that development for this entity won't be too long.*

		Big O time evaluation data structure access*:* **O(log$_2$N),** This will be due to the binary chop nature of the algorithm; this should suggest an extremely fast search based on the relative size of the data. This will be the main advantage of the sequential file organisation.
Patients *Patient.java* *PatientList.java* **Inherits:** *User.java*	**Array –** Used hold them in memory to be used as a collective, also used when accessing the system as a staff/consultant entity and trying to find the correct patient **Record –** The structure to hold the user's data while logged in as the user, if information needs to be changed about the entity it will be the first to see the change.	Expected Number of records:**300** *While in reality there will be hundreds upon hundreds of patients it would be a waste of resources to individually create meaningful data in every field. While the use of "Dummy data" could be an option around this the main use for this system is to gain fast access to certain information having every record contain the same data will be pointless. Therefore, to try and put some pressure on the system but not waste my time adding unique information I believe around 300 records will sufficient, it can change if necessary. As in reality it will be forever increasing.* Method of file organisation: **Direct Random-Access File** *Random access file is best suited for this personally as this will be the largest collection of records on the system, while in admissions there will be many admissions the number of patients in reality will be much larger. As for instance only one patient may have no active admissions. The other reason this was chosen was because of the drawbacks associated around serial and sequential such as searching of items. The main benefit being that the access to data will be solely down to the performance of the hashing algorithm, while it may produce collisions I feel that the utilisation of a dynamic link list should relatively fix the problem. However the focus will be to prevent this from happening not getting around the issue.* ***As the hashing algorithm directly produces a path to the desired data it will make searching much easier.*** *While sorting will become much harder this should not become an issue as there will be no need to sort patients directly from file on the system, instead for instance if the consultant wants to view their collection of patients, those accounts available to them will be pulled into a new array and then allowing eventual access to their information, while inconvenient I feel that overall effect will be better this way on the system. As the list of patients may be needed for employees an* ***insertion sort*** *will be employed to correctly order the entities on the system if needed. As the system will also have other lists of patients for instance the consultant's available patients* **a binary search** *will also be utilised to correctly retrieve the index. This will be because the hashing algorithm will only work with the designated hashing table and so another effective search will need to be used however as the list size will be much smaller there is nothing to be slightly concerned about. For these smaller lists an* ***indexed sequential file*** *organisation method will be put in place to allow retrieving of data through associative indexes, but as mentioned before the insertion sort should keep the list of data in order on the system.*

		Use: **Constantly changing** *Unlike the staff entity, patient entities will be able to be deleted this needs to happen in order to follow real life GDPR regulations, which are an important issue at Euxton currently. In addition to this the information about patients held in the fields will always updating this will be needed to prevent data inaccuracy. Because of all of this the file will not be static but instead constantly changing.* Programming time required: **Majority** *As this is the entity that the new system revolves around it is obvious that most of the development time as of now is predicted to be sent on this entity and admissions. This will be because most features found in this entity will be inherited by other classes such as the fields in the patient demographic* *Big O time evaluation data structure access:* ***Subjective to the algorithm for key generation,*** While the time efficiency of the algorithm can't be exactly determined due to the individual nature of the algorithm itself it can however be discussed to declare a base line of expectations. Without this it will not merit the usage of a hashing algorithm.
Admissions *Admission.java* *AdmissionList.java* **Inherits:** *Patient.java* *User.java*	**2D Array –** Used on the consultant side, the rows are for the consultant's patients and then the columns are for the induvial admissions, this will allow for the specific admissions to be pulled from a patient's account **Record –** used to hold the information regarding the admission, if information needs to be changed about the entity it will be the first to see the change. **Array –** while the 2D one will be for the consultant this will be for the patient in order to access their admissions on the system.	Expected Number of records: **4/5 per patient** *As the patient will have numerous ailments over their life they will over time accumulate many admissions on the system. Rather than counting the expected in total I believe a more suitable estimate would be to go off a per patient basis. Using my experience working there, and from my investigation a patient who has been with Euxton has around three to five different reasons to visit Euxton. Because of this I expect there to be a similar number of admissions. However this value is not set in stone and will fluctuate along with any other value like any entity related to the patient.* Method of file organisation: **Serial** *I believe that the file should utilise a serial organizational system due to the fact the patient individually will only poses around 1 to 5 admissions in their use of the system I expect because of this it should be very fast to store them in the order in which they are created, also as they will be stored in the date they are used the more frequent records will be accessed quicker. As they will be personal on a file per patient basis this should cause no latency in retrieving records. And instead will rely on the calling of files using the read method. The other major benefit to the serial approach will be that the latest admission will always be viable to the patient and should help them find what they are looking for on the system. While the number will be only a handful I believe that utilising a* **linear search** *will provide the best results and can therefore also support the use of a serial file organisation. As the list of admissions will also be needed for the users of the system an* **insertion sort** *will be utilised to effectively sort data in the structure.*

		Use: **Constantly updating** *The records held on this file will be kept and will never be deleted however to offload unneeded records I feel that archiving them onto an external file may be necessary. However, the location of a file will be permanent and will be unlikely to be changed as the physical address of the record will be found from hashing algorithms on the key filed* ***AdmissionID.*** Programming time required: **Most** *In reality the Admission entity is an extension of the patient entity and as previously stated the patient entity will receive the majority of development time because of this, this entity will also see a large amount of time for development spent on it. However, while other entities on the system will not be neglected, I intend to make sure that all the processes and features work as intended.* *Big O time evaluation data structure access:* **O(N)** As the size of the data is small there is no need to utilise an advanced data searching algorithm, by using a linear search an effective algorithm for the data size can be used on the list. This obviously being the main advantage of the serial organisation.
Consultants *Consultant.java* *ConsultantList.java* **Inherits:** *Employee.java* *User.java*	**Array -** Used to hold the consultants on the system as a collective, this will be needed to allow them to be searched for from the management entity. **Record –** Used to hold the information for the entity on the system while they are logged in, if information needs to be changed about the entity it will be the first to see the change	Expected Number of records: **25** *To fully utilise the different medical practises present at Euxton I feel that around 25 consultants reflects the ratio in reality of patients to consultant's present. If too many consultants where on the system the booking side of the system will not reflect how it would work in reality as normally consultants are fully booked rather than having many hours free. However, the number is possible to change as the management has the ability to add new accounts to the system* Method of file organisation: **Indexed Sequential** Indexed sequential will be used on the system for the consultants as it will be the most viable file organisation available to them. *Because of this, the immediate advantage to this is the reduced development time required to develop it compared to other methods. As there will only be a handful of consultants present on the system there is no need in creating an advanced file handling method, therefore negating the use of direct random file access and other more complex techniques. The main benefit of utilising indexed sequential however is the ability to perform* ***binary searches*** *on them, which is what I intend to utilise. To add to this if a field of the consultants needs changing it can simply perform a binary search find the location in the structure and then immediately amend the information. However to maintain a correct order, like for the staff of the system, a sorting algorithm will need to be put in place to manage the order for the data structure. The current most ideal for these users will be an* ***insertion sort,*** *this will allow for the list to be sorted much faster than other sorting algorithms but will not take as much time to develop and test as seen with recursive algorithms like quick sort. As the size of the data does merit using the file organisation method I feel that the entities constant use on the system will validates its need for it more*

		Use: **Permanent** *The entities on the system will be mostly permanent and no fields will be removed from the system. Despite this there is the process of archiving employees from the system, while this will not delete the entity it will prohibit use of the record or interactions with other records.* Programming time required: **Some** *As this entity outside the patient ones is the largest on the system, this entity will have a considerable amount of development time spent on it to ensure that all functionality and process work as intended and are up to standards. This can also be justified as it in reality will have more functionality than the patient due to the patient being unable to produce documents such as test results and dictations.* *Big O time evaluation data structure access:* **O(log$_2$N),** This will be due to the binary chop nature of the algorithm; this will be due to the benefactive nature of the search, while having to have the data in a constant sequential order will bring some annoyance to keep the order correct it will be very necessary when having to search the consultants of the system.
Management *Managment.java* **Inherits:** *Employee.java* *User.java*	**Array –** Will be used to handle the entire collection of the management entity, it will be used to access them if needing to log onto the system **Record –** This will be used to hold the information about the user while active on the system, if they want to edit their information, this will be the first structure to change the fields.	Expected Number of records: **3** This entity is an exception on the system, while many users of the same entity exist for the rest of the user types the one, I don't intend on having multiple records for will be for management. This is because there will be around a handful system managers in reality present at Euxton and due to the lack of fields besides generic data like name there will be no need to have multiple entities of this user present on this system. Method of file organisation: **Serial** *As only one or two entities for management will be on the system it is obvious that serial file organisation will be used to handle records on the system, because of this it will drastically reduce development time and due to the number of records sorting of this entity, while not intended on being used, is available and will not be slowed down by the number of records. This will mean that the entity can use a* **linear search** *in order to effectively access the data. As there is no need to view the list of data in any way* **no sorting algorithm will be needed** Use: **Permanent** *All data held about the management staff will be permanent and will not be updated or changed and will be static. This is due to the fact it will be too far out of scope to allow amending of management information and overall would be useless on the system as it will be ultimately unused.*

		Programming time required: **Least** Finally, as it is the smallest user on the system, in terms of functionality, it is clear that the least amount of time on this user will spent on it being developed. This can also be reduced as I intend on the user interface for the user also using Command line. While I will still aim for the quality for this end of the system to be high, I will not focus any extra resources that are unintended. *Big O time evaluation data structure access:* ***O(N)*** Due to very small data size for the admissions I see no problem in the big O time efficiency as the size will be less than 5 at most the number of items is so small the time required basically is instantaneous and poses no down time on the system.
Document (minor entity) *Document.java* *DocumentList.java* **Inherits:** *Admission.java* *Patient.java* *User.java*	**2D Array –** A two dimensional array will be needed in order to hold every document in the system, every outer index will be used for an admission and will allow them to be grouped, while the inner array will be used for the documents themselves. **Record –** As the object of document is an entity and will have varying datatypes its most suitable to use a record to hold the information.	Expected Number of records: **(on an admission basis: 1<50 on average)** The reason there is no definite number is because there is no true value. This stems from my investigation at Euxton, when performing document inspection as a method of investigation an occurrence that happened was that sometimes I would see patient files that where too heavy to carry and then immediate followed by a different patient with around three documents. This obviously down to the fact the expected number of fields being directly proportional to the time spent being an active patient. while this will put a strain on files sizes as a definite number of documents can't be presumed it means that some files will have many records and some one or two. Method of file organisation: **Indexed Sequential** After coming from the investigation a common issue was accessing documents, with them taking prolonged periods of time to find the most recent one. With that being said indexed sequential will be utilised in a way to prevent these issues occurring on the new system. The immediate ability available is the use of a **binary search** which should vastly improve searching capabilities over older much slower methods of searching like linear. While documents will use an initial serial ordering by date, the option to search by other key fields will likely become a necessity to the system. However due the requirements of the search, the documents will need to retain a constant order on the system and therefore *it will be necessary to incorporate an* ***insertion sort*** *on the system.* The main benefit being that the system will be able to quickly allow the user to isolate the particular document they need; this will be especially important when the number of documents per admission could go theoretically into the hundreds Use: **Permanent** *The obvious reason that these files will remain permanent on the system is due to why the system was concepted in the first place to replace paper documentation and the eventual degrading of data that comes along with it. To add to this while some GDPR regulations allow any data revolving a patient to be able to be removed I will simply archive the admission therefore prevent access to the information but still retain it on the system*

		Programming time required: **Least** *As documents on the system are a priority, compared against the functionality set with the other entities it is clear the range of features and data types for the entity will result in this being the smallest entity on the system and so will require the least time to develop.* *Big O time evaluation data structure access:* **O(log$_2$N)** As the algorithm relies on a divide and conquer approach to searching data this will allow a very fast access to data by portioning the list until the desired item is found.

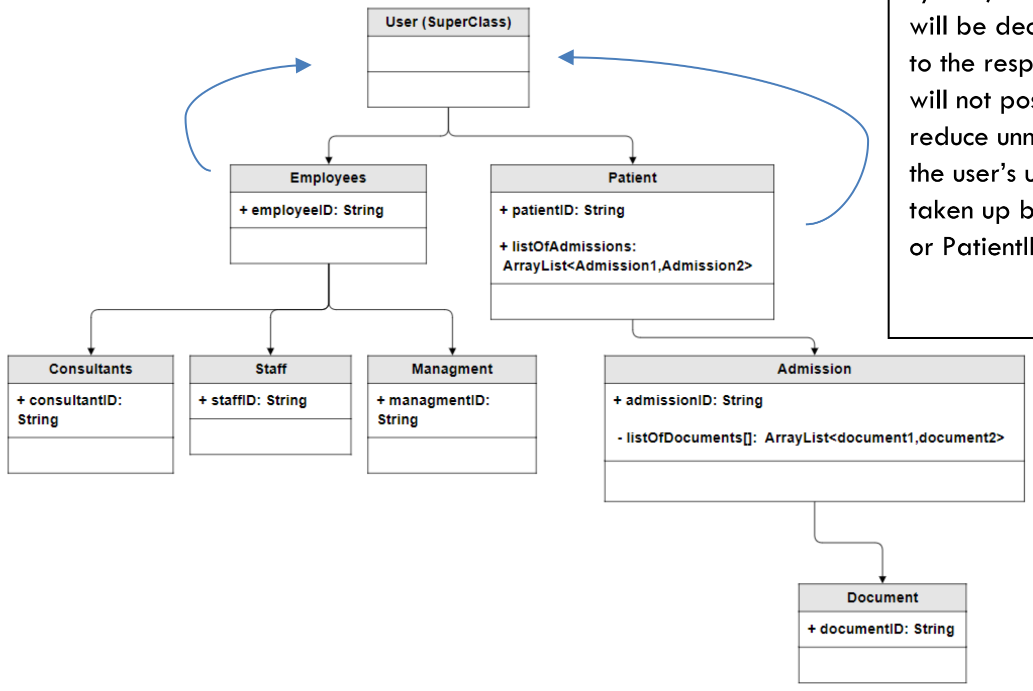
3.33 Data Structure Designs

Here are the main classes, excluding ones to build the GUI, of the system. It is clear that I will definitely utilise inheritance to reduce the declaring of repeating attributes such as first name and surname. In addition to that I have included extra information regarding every field present in the record. For each main entity I have included the classes that are needed along with the list classes for the multiple records to store as a collective unit and can be processed as one. While key field is somewhat important as it will be needed to disclose what variables classify what entity due to the class diagram and the inheritance hierarchy, only primary keys really need to be shown as it's obvious any passed on keys would be foreign for instance employee ID in the admission. However an important reason the lack of foreign keys exist is due the idea of data redundancy.

A point worth noting is the use of primary keys in both objects and lists. In regard to objects, as the entity is a specific instance of a class, the entity will need to retain a primary key as an attribute, so while the object in itself specifically identifies an entity from the class that declared it, the primary key will need to be used in retrieving the entity, for instance in searches where the address held in memory for the instance would be unable to be used once the program is closed due to it never being the same location. To identify all cases where an object utilises a primary key a * will be used.

Primary keys on the system and inheritance

To minimise unnecessary variable declaration through the classes, the system will utilise inheritance and extensions to limit the number of variables and classes needed on the system. The best example of this will be the class user. This will act as a superclass to the entirety of the system providing a basis for inheritance allowing common attributes (like name) and processes (like logging in) to be used by every user on the system and will only need to be declared once, this is in an effort to reduce usernames, unnecessary attributes and common processes and as a result the user class will not have a primary key and instead clients will use the primary keys of its subclasses Employee and Patient for actions on the system. The main idea of this being down to the fact it seems futile having two primary keys for the same entity on the system. The initial benefit of this would be seen through logins as the entirety of the system would not need to be searched and the fact using two separate primary keys seems unnecessary for one user. Even more with the planned primary keys in place entities will be able to be identified as the type of user they are against other users like staff and consultants just from the key itself. As for the extensions these will be used to attach the list classes to the system allowing for multiple instances of the same child class all linked to one account, the initial example that comes into mind is the admission regarding the documents on the system. To help get this idea across the class diagram has been altered to show how these attributes and extensions work. The variables signify the primary keys for the respective user.



In an effort to reduce the duplication of variables which are general to the system, like name and address, they will be declared in user and inherited to the respective classes. But the class will not poses any primary key to reduce unneeded attributes. Instead the user's unique identification will be taken up by either their EmployeeID or PatientID

User.***java***

With the user class this entity just provides the basic foundation to any user as every entity inherits these characteristics its important that they remain nonspecific between patients and employees and so an end result of the precaution is that there are no primary keys for this entity. Obviously the primary keys for the main classes are imposed on other classes.

Field Name	Key Field	Data Type	Length	Example Data	Validate?
firstName	-	*String*	*<10*	*James*	🟩
surName	-	*String*	*<10*	*Nurdin*	🟩
gender	-	Character	1	M	🟥
dob	-	LocalDate	10	28/05/2002	🟩
houseNum	-	*Integer*	3	*167*	🟩
houseStreet	-	String	<25	Town Road	🟩
Town	-	String	<20	Wigan	🟥
Postcode	-	String	7	PR26 9RA	🟩
contactNumber	-	String	11	07484727992	🟩
Passeword	-	String	>0	Dosed123!	🟩

Employee.java

While not all entities need discussing I feel that this entity needs some explanation, all I want to address is that the majority of fields are not validated due to the fact the object will be used in the action log. As the process is automated all the fields will be correct and hence no longer require the need for validation. These values will essentially be hardcoded into the system by pulling variables such as time from the system itself. While it could be appropriate to validate it regardless I believe that it is unnecessary, however if I feel that changes need to come into fruition I will make them.

Field Name	Key Field	Data Type	Length	Example Data	Validate?
employeeID	*PK*	*String*	*11*	*ENUR1234567*	🟩
wagePerHour	-	*Real*	*4 (2 int,2dp)*	*12.80*	🟥
hoursPerWeek	-	Integer	2	42	🟥
currentAction	-	*String*	>0	Created Document	🟥
DateActionPerformed	-	*Date*	8	12/04/2019	🟥
PatientInvloved	FK	*String*	>0	PNUR0000001	🟥
AdmissionInvolved	FK	*String*	>0	ANUR0000001	🟥
newData	-	*String*	>0	*"Test results look good no worries"*	🟥
oldData	-	*String*	>0	n/a	🟥

EmployeeList.java

Field Name	Key Field	Data Type	Length	Example Data	Validate?
allemployees[]	-	*Object*	*100*	*List of all employees*	🟩
tempEmployee	-	*Object*	*10*	*An employee*	🟥
nextPosition	-	int	3	99	🟥

Staff.java

With the staff entity I believe that they don't need to possess many entity specific attributes. While the entity does retain all attributes inherited through the hierarchy there is little entity specific values that could benefit including on the system. However if feedback suggests that there needs to be more relevancy to the entity then changes will occur to see it through.

Field Name	Key Field	Data Type	Length	Example Data	Validate?
staffID	*PK*	*String*	*11*	*SNUR1234567*	🟩
archived	-	Boolean	1	0	🟥

StaffList java

Field Name	Key Field	Data Type	Length	Example Data	Validate?
allStaff[]	-	*Object*	*100*	*List of all staff*	🟩
tempStaff	-	*Object*	*10*	*A staff member*	🟥
nextPosition	-	int	3	99	🟥

Consultant.java

As the consultant is the second largest user entity on the system I felt it was imperative that enough attributes where supplied in order to make the entity feel that it had an important role on the system. Another reason why was due the fact that the staff entity may be imposed to determine a consultant for the patient's admission on the system, without having some information regarding the admission I think the action couldn't occur.

Field Name	Key Field	Data Type	Length	Example Data	Validate
consultantID	*PK*	*String*	*11*	*CNUR1234567*	🟩
ward	-	*String*	*<15*	*Cardiology unit*	🟩
practisesList[]	-	String	>1	Anesthesiology,Dermatology	🟥
nextPatient	PK*	Object		PNUR1234567 James Nurdin ANUR0000001	🟥
timeOfNextApp	-	LocalTime	4	17:18	🟩
dateOfNextApp	-	LocalDate	10	DD/MM/YYYY	🟩
archived	-	Boolean	1	0	🟥
numOfPatients	-	Integer	3-2	99	🟥
listOfAppointments	FK*	Object		PNUR1234567 17:18 DD/MM/YYYY R206	🟥

ConsultantList.java

Field Name	Key Field	Data Type	Length	Example Data	Validate
allConsultants[]	-	*Object*	*100*	*A list of consultants*	🟩
tempConsultant	-	*Object*	*15*	*A consultant*	🟥
nextPosition	-	Integer	3	99	🟥

Management.java

Again in similar respect to the staff entity I believe that the management entity has no reason to withhold data on the system besides basic demographic information. While yes their roles are important, there is no processes that immediately require this users' attributes that are specific to them. Because of this I can therefore only include a primary key in order to locate them on the system individually.

Field Name	Key Field	Data Type	Length	Example Data	Validate
managmentID	*PK*	*String*	*11*	*MNUR1234567*	🟩

ManagmentList.java

Field Name	Key Field	Data Type	Length	Example Data	Validate
allManagment[]	-	*Object*	*100*	*A list of Management staff*	🟩
tempMangement	-	*Object*	*15*	*A management staff*	🟥
nextPosition	-	Integer	3	99	🟥

Patient.java

With these values, the reason that the fields have only a few validation attached is down to the fact that the fields in particular are either binary state or can only be selected as true or false therefore making them pointless to validate. While some may believe that it would be necessary to see these attributes validated as at any time the value will always be an acceptable value verification would be necessary rather than validation.

Field Name	Key Field	Data Type	Length	Example Data	Validate
patientID	*Pk*	*String*	*11*	*PNUR1234567*	🟩
nationality	-	String	<25	English	🟥
bloodtype	-	String	1-2	O+	🟩
smoker	-	Boolean	1	0	🟥
drinker	-	Boolean	1	1	🟥
disability	-	String	1	0	🟥
numOfAdmissions	-	Integer	2	5	🟥
Carer	-	Boolean	1	1	🟥
Translator	-	Boolean	1	0	🟥
numberOfNotifications	-	Int	10	12	🟥
notifications	-	String[]	N/A	Appointment Was Set to 19/05/2019	🟥

PatientList.java

Field Name	Key Field	Data Type	Length	Example Data	Validate
allPatients[]	-	*Object*	*100*	*A list of Patients*	🟩
tempPatient	-	*Object*	*15*	*A Patient*	🟥
nextPosition	-	Integer	3	99	🟥

Admission.java

With admissions as they will be among one of the entities with the most important information on the system I feel that it is important that most values are validated. Because of this I will explain why some of the values don't need validation. With the attribute active as the state is either true or false there can be no state that is invalid on the system while activating may be undesired the process can be undone and therefore doesn't need to be checked. With the number of documents this is a back end value use for data manipulation and so never needs to be entered by the user as it doesn't require human interaction it can therefore be excluded from validation.

Field Name	Key Field	Data Type	Length	Example Data	Validate
admissionId	*PK*	*String*	*11*	*ANUR0000001*	🟩
ward	-	*String*	*<15*	*Cardiology unit*	🟩
consultantID	FK	*String*	*11*	*CNUR1234567*	🟩
staffID	FK	*String*	*11*	*SNUR1234567*	🟩
timeOfNextApp	-	LocalTime	4	17:18	🟩
dateOfNextApp	-	LocalDate	10	22/05/2018	🟩
active	-	Boolean	1	1	🟥
numOfDocuments	-	Integer	2-3	12	🟥
medication	-	String	>20	Azithromycin	🟩
dosage	-	String	5	10.0mg	🟩
intakeTime	-	LocalTime	4	17:18	🟩
dateOfNextDispatch	-	LocalDate	10	22/05/2018	🟩
listOfSymptoms[]	-	Sting	Around 5 indexes	Excessive hunger Weight Gain Fatigue Nausea	🟩
currentDiagnosis	-	String	>15	Diabetes type 2	🟩
room	-	String	4	P001	🟩

AdmissionList.java

Field Name	Key Field	Data Type	Length	Example Data	Validate
allAdmissions[]	-	*Object*	*100*	*A list of Admissions*	🟩
tempAdmission	-	*Object*	*15*	*An admission*	🟥
nextPosition	-	Integer	3	99	🟥

Documents.java

Finally the last critical entity on the system, as most of the system is focused on making sure that documents are correct and not missing any data I am positive in saying that all fields manually entered for this entity will be validated to some degree. Rather than justify each induvial one I would instead suggest why I believe that the ones not validated are for a reason. With the time and date attributes for the documents creation rather than force the document author to manually enter the data and time, the system will automate this process. By doing this it will greatly improve accuracy and could very critical if an error was made. With hospital as stated in the investigation that only Euxton would be considered on the system it means that the attribute gets to be hardcoded and therefore no longer needs t be validated.

Field Name	Key Field	Data Type	Length	Example Data	Validate
documentID	*Pk*	*String*	*11*	*DNUR1234567*	🟩
timeOfDocument	-	Time	4	17:18	🟥
dateOfDocument	-	Date	*10*	22/05/2018	🟥
consultantNotes	-	String	N/A	MR Jones failed to attend the consultation today, I will postpone is surgery to a later date until he visits again to discuss final preparations	🟩
Hospital	-	String	N/A	Euxton Hall	🟥
testResults	-	String	N/A	Ammonia concentration 0.5% etc	🟩
Medication	-	String	N/A	Paracetamol	🟩
dosage	-	String	3/4	5mg	🟩
IntakeTime	-	String	N/A	09:00	🟩
dateOfNextDispatch	-	Date	10	22/05/2018	🟩
employee authorID	FK	String	11	ENUR0000001	🟩

DocumentList.java

Field Name	Key Field	Data Type	Length	Example Data	Validate
allDocuments[]	-	*Object*	*20*	*A list of documents*	🟩
tempDocument	-	*Object*	*15*	*A document*	🟥
nextPosition	-	Integer	3	99	🟥

3.34 File Designs

(PatienttID)_(AdmissionID)_Documentation.txt

As the file will be centred around storing the documents of the patient there will be a varying combination of file types between every patient and so every document file will be different in respect to the individual however the selection available to them will be static and can therefore be defined no. As there are only a few files to work from on the system this section will be small, but rest assured the file size could be large due the amount of records in the file. First the doctype is used to allow indication of the filetype to be immediately identified on the system. Then the more generic data is added to the end of the record. The main 4 types are as shown:

Prescription

(docType),(date),(time),(documentNum),(medication),(dosage),(intakeTime),(dateOfNextDispatch)

Dischargement

(docType),(date),(time),(documentNum),(ailment),(consultant.firstname)

Consultant Notes

(docType),(date),(time),(documentNum),(notes)

Test Results

(docType),(date),(time),(documentNum),(testResults)

e.g.

Prescription,23/06/19,12:04,DNUR000001,Paracetamol,12mg,09:00,30/06/19

Consultant Notes,25/06/19,11:33,DNUR000002, Test came back negative nothing to worry about.

Prescription,30/06/19,12:04,DNUR000003,Paracetamol,4mg,09:00,07/07/19

Consultant Notes,07/07/19,01:32,DNUR000004, James came to see me seems like he no longer needs the medication I will see him in a week to make judgment to continue the medication or not.

Prescription,07/07/19,12:04,DNUR000005,Paracetamol,1mg,09:00,14/07/19

Dischargment,14/07/19,12:04,DNUR000006,Cold,Andy

(PatientID)_oldDocumentation.txt

Just an extension of the prior document this will be used to store the documents from the prior system separately in order to have the best access to new/old documents without the need to sort them into two different arrays by doing this more time should be saved on the system. While it certainly won't be as large as the other file it will allow for a distinct separation of data between time phases of the system, where all digital media on one document and all old paper based converted data on another file, however this file will only be accessed when the account is created and the process of data changeover occurs.

(admissionID),(scanName),(Date)

e.g. ANUR123,Referral1,5/06/2017

(patientID)_Admissions.txt

The purpose of this file will be to store the information regarding the patient's admission on the system. These files will contain all the relative information about the admissions related to the patient. Similar to how the demographic file will hold the information about all the patients on the system, this will hold every admissions information as a record on the file. The whole point will be that the entirety of the admissions can all be read from one coherent location and can immediate be assigned to the admission list class directly from being read from file. The order will be utilise a sequential order with the primary key being the first value but as the ID is generated by finding the last value and incrementing it will be in serial order also.

(admissionID),(ward),(consultantName),timeOfNextApp),(dateOfNextApp),(active),(Medication),

(room),(currentDiagnosis),(listOfSymptoms),,(dateAdmissionCreated),(staffName),(staffID),(consultantID)

e.g.

ANUR0000001, Physiotherapy,Sutton,20:35,22/05/2018,True,Azithromycin,P001, Arthritis , [stiffness, sharp pains, slight swelling],Tomkins,20/05/2018,STOM1234567,,CSUT12234567

ANUR0000002,Physiotherapy,Will,12/12/2099 19:2,true,2,null,P003,Depression,Symptom 1|Symptom 2|Symptom 3|,tom,23/09/2019,STOM0000001,CSUT0000001

Patient_Demographic.txt

Similar to the admission file instead of isolating every patient's demographic information into separate files they are all placed into one file where any patient's information can be directly accessed onto the system without having to also search for patient files. The main benefit of this obviously being that it will take no time to perform a binary search after reading them all from file. Again like I said in 3.31 the main access to the data is sequential this should also allow for the system to use binary searches effectively. The main reason this file will be needed as there will be no general file for the patient created on the accounts generation that will contain enough data to merit its own separate file and so they will be stored as a collective.

(patientID),(surname),(firstname),(houseNum),(houseStreet),(town),(Postcode),(contactNumber),(nationality),(bloodtype),(smoker),(drinker),(numOfAdmissions),(D.O.B),(Religion),(Allergies),(Gender),(diablities),(carer),(translator)

e.g.

PMAR0000001,Marr,Jonny,123,a,a,a,1232456789,Manchester,MN6
7LO,true,true,0,12/12/0012,none,none,Male,none,false,false

PMOR0000001,Morrisey,Steven,12,test lane,a,Manchester,123345678898765,,ML3
4RW,false,true,0,12/12/2121,None,a,Male,none,false,false

PNUR0000001,Nurdin,James,55,Spelding Drive, Standish Lower Ground,WN6
8LW,07484727992,Wigan,O+,true,true,5,28/05/2002,none,Pencilinin,P,,false,false

StaffInfo.txt

While this does not differ much to the patient demographic file this will be used for the staff entity specifically to allow direct access to the user's information from logging into the system. Again this uses a sequential file organisation and also uses the primary key to individualise each record. The only difference between other files that will store the demographic information will be down to the fact this will specialise in the staff specific fields also.

(staffID),(employeeID),(surname),(firstname),(wage),(hourlyPerWeek),(archived),(houseNum),(houseStreet),(town),(Postcode),(contactNumber),(dob),(gender)

e.g.

SNUR1234567,ENUR1234567,Nurdin,James,12.80,42,False,167,Town Road,Croston,PR26
9RA,07484727992,28/05/2002,G

ConsultantInfo.txt

The purpose of this file like staffInfo.txt will be to hold demographic information for an employee of the system, while it may have been preferable to some to just combine all the employee demographics onto one system. I feel that it would mean that once they have logged in all their information will then have to be searched again to isolate the correct data associated to the desired employee. To reduce this process as much as possible I have split the file into three separate files. The actual data itself only varies by one or two variables but it means that only one function and data structure will be needed to read the entirety of the file and assign them to an array.

(consultantID),(employeeID),(surname),(firstname),(ward),(practisesList),(wage),(hourlyPerWeek),(archived),(numOfPatients),(list of patients)

e.g.

CNUR1234567,ENUR1234567,Nurdin, James, Podiatry, Anaesthesiology Dermatology ,12.80,42, False,35,PNUR0000001 PTOM0000003

ManagementInfo.txt

Finally the last of the three employee demographic files this last file will also contain the information about the management entity on the system. While the number of records in this file will only be a handful I feel that serial would be sufficient. Anyway the reason the file is needed is to allow direct access to the account as once the login is successful the file will simply need to be read and then a linear search can be used and then the correct account can be returned. Besides the unique attributes there is nothing different to the account.

(manamgentID),(employeeID),(surname),(firstname),(wage),(hourlyPerWeek),(archived)

e.g.

MNUR1234567,ENUR1234567,Nurdin, James,12.80,42, False,

Employee_Actionlog.txt

The purpose of this file will be to store the entirety of the actions the employee performs on the system, the reason an induvial file is utilised is because the number of records will be very high and will be in the tens to hundreds only within a few weeks of working there. Because the file size will be very large for just a single employee on the system, I believe that to prevent stupidly long search times that a file is commissioned to every employee on the system. This will help prevent the long reading and searching times that will occur with objectives 12 and 13.

(EmployeeID),(numberOfActions),(localDate),("action"To"Location"),(patientID),(AdmissionID),(OrginalData),(NewData)

e.g.

02/09/2019 13:28,Created Document Test Results,PNUR0000004,ANUR0000001,n/a,Test 11 Urgent

EmployeeLoginInformaion.txt And PateientLoginInformation.txt

Both of these files serve the same purpose that being holding the login credentials to allow the user to log onto the system. To reduce the time having to search the system for a specific user type then find the exact user, the system will already know what type of user is needed and can then perform a binary search to find the users primary key. While the size of each record will be small the fact that every user will poses a record in either one of these will mean that the size will be large from an early point of the systems use.

Patient

(patientID),(password)

e.g.

PNUR1234567,Dosed12!

Employee

(employeeID),(password)

e.g.

ENUR1234567,Dosed12!

(ConsultantID)_Bookings.txt

This is the last major file of sorts of the system this file will contain all the information regarding the bookings that the consultant will poses. The reason that the file is isolated from the staff demographic is because it will be unnecessary to pull every single appointment of the system every time the employee may wish to log in. Because of this and the fact that the entity doesn't have a dedicated file a new one needed to be generated on the system. While at this point it feels that there may be too many different files I believe that overtime it will prove its inclusion with the fact searching through files should be a lesser worry.

(patientID),(admisisonID),(timeOfNextApp),(dateOfNextApp),(ward),(room),(otherInfo)

e.g.

PNUR0000001,ANUR0000001,10:30,12/11/19,phisotherapy,P002,should be the last appointment if it goes well

Jargon_Library.txt

This is just a minor file that will hold all the detentions to the jargon library on the system, it was made its own file due to the fact that no other file had any other associative data. This was also done as the definitions would take up quite the size of the file so it will ideally only be read from file once when the system starts up. Because of this it would be unideal to include any updating data as the rest of the file will need to be included.

(Word),(Definition)

e.g.

Arthritis, a common condition that causes pain and inflammation in a joint.

Notifications.txt

Finally this last file will be for all the patients on the system. When they have been logged in their notifications on the system will also be pulled up. The size of the record will vary on the account but will always contain the ID so that they have the ability to obtain notifications. Other than this there is no notable aspect about the file besides the fact they utilise a data structure to hold the notifications, so that after the patient is found it is only a matter of copying over the fields into array for them to be seen by the patient

(patientID),(Notification[0]),(Notification[1]),(Notification[2]),(Notification[3]),(Notification[3]),(Notification[4]),(Notification[5]),(Notification[6])

e.g.

PNUR0000001,New document created, SNUR0000001 edited your admission, your appointment for tomorrow was cancelled.

3.4 Validation

While in 3.33 where every attribute was discussed here all the data that needs to be validated is shown. While not every value needs to be changed those that do need to make sure that they are correct. As the whole point in my system is to reduce data accuracy it is important that the data here achieves this.

Class	Field	Val. Type	Val. Rule Description	Justification	Error Message
User.java	*firstName*	***1.Type*** ***2.Presence***	*1.Must be String, no numbers/symbols* *2.Must be present*	*The first name needs to be verified for presence because everyone should have a first name, while some people do use mononyms like Bono, as it is a medical file the full name given to the user at birth should be required.*	*1." Value entered is not a string"* *2." Field must be filled in"*
User.java	*surname*	***1.Type*** ***2.Presence***	*1.Must be String, no numbers* *2.Must be present*	*Similar to the first name another reason for the field to be validated is due to the fact no name will use a non-letter character in their name because of this both the surname and first name should filter out incorrect characters. Also as the field is used in the credentials for a login we can reduce the chances of incorrect entry of data as all usernames follow a standardised format of "Type""3 letters of surname""7 digit number".*	*1." Value entered is not a string"* *2." Field must be filled in"*
User.java	*dob*	***1.Format*** ***2.Range*** ***3.Lookup***	*1.Must Follow correct format of DD/MM/YYYY* *2.Must be <= current date* *3.Must be selected From available dates*	*The date of birth needs to be validated due to the fact age is an important value of data as age is a determining factor on dosage and if a type of procedure should be followed through because of this to try and reduce invalid data it should be validated at least.*	*1. "Data must follow correct format of DD/MM/YYYY"* *2. "DOB can't be set to the future"* *3. "You must select a suitable date"*

User.java	***Password***	*1 Presence* *2 Type* *3.Length*	***1.A password must be entered on the system.*** ***2.A password must contain characters numbers and symbols*** ***3.A password must be at least 7 characters long***	***The reason the password is validated is simple, it is used to protect the user's personal data without it any user can gain unauthorised access to confidential data on the system. To address this a wide range of validation will be incorporated to make sure that the data is safe***	***1."A password must be present on the system"*** ***2. "The password must contain a symbol a number and a capital letter"*** ***3. "The password must be 7 characters long"***
Employee.java **-** **Consultant.java** **-Staff.java** **-** **Management.java**	*employeeID*	***1.Length*** ***2.Format*** ***3.Presence***	*1. Must be 11 characters* *2.must follow E"3 letters of surname" number* *3.Must be here*	*The employee ID should be validated as it is a unique identifier of the entity if it was entered incorrectly it would result in the actual entity being wrongly identified and could result in their file being sorted in the incorrect position.*	*1." Your employeeID is 11 characters long"* *2." employeeID has been entered wrong"* *3." An employeeID is required"*
EmployeeList.java	*allEmployees[]*	***Type***	*Must be an object of employee*	*This array needs to be validated as it will contain all the employee entities when the program is running and will be used to search and sort them. If a wrong entity was to be passed into the list it would also be included, which we don't want included.*	*"Data entered was not an object of employee"*
Staff.java **Admission.java**	*staffID*	***1.Length*** ***2.Format*** ***3.Presence*** ***4.lookup***	*1.Must be 11 characters* *2.must follow S"3 letters of surname" number* *3.Must be here* *4.Must be selected from a list of accounts*	*Similar to the employeeID, this field needs to be validated to make sure when entering information such as log in information it is correct and when the account is being created it will follow a universal standard. In some cases they can be selected from a drop down box so will also utilise a lookup check in areas of the system.*	*1." Your staffID is 11 characters long"* *2." staffID has been entered wrong"* *3." A staffID is required"* *4." Must be selected from the drop down box"*

Class	Field	Val. Type	Val. Rule Description	Justification	Error Message
StaffList java	*allStaff[]*	***Type***	*Must be an object of Staff*	*This array needs to be validated as it will contain all the staff entities when the program is running and will be used to search and sort them. If a wrong entity was to be passed into the list it would also be included, which we don't want included.*	*"Data entered was not an object of Staff"*
Consultant.java Admission.java	*consultantID*	***1.Length 2.Format 3.Presence 4.Lookup***	*1.Must be 11 characters 2.must follow C"3 letters of surname" number 3.Must be required for the entity 4.Must be selected from a list of accounts*	*We want this id to be validated because it will be needed for the entity to access the system and will be used as a key identifier for this entity if it was to be incorrect on the system it would result in having a negative effect on the system as it would not be in the intended position anywhere. In some cases they can be selected from a drop down box so will also utilise a lookup check in areas of the system.*	*1." Your consultantID is 11 characters long" 2." consultantID has been entered wrong" 3." A consultantID is required" 4." Must be selected from the drop down box"*
Consultant.java	*ward*	***1.Presence 2.Lookup***	*1.The entity must be allocated a ward 2. A ward must be selected*	*This field needs to be validated as it is used to indicate where the patient/consultant will be, while data entry will be in the form of a drop-down menu a value will still need to be selected.*	*"The user is missing a ward"*
Consultant.java Admission.java	timeOfNextApp	***1.Lookup 2.Range***	*1.Feild must be data type of LocalTime 2.Time must be real i.e NOT 32:32*	*This field will need to be validated because the time data type is important for bookings if an incorrect time was set it would result in the patient attending at the wrong time to reduce this from occurring.*	*1."The time must be selected from the time selector" 2."That time does not follow the required range of a 24-hour clock"*
Consultant.java Admission.java	dateOfNextApp	***1.Lookup 2.Range***	*1.Feild must be data type of LocalDate DD/MM/YYYY 2.Date must be in the future*	*This field needs to be validated because the date is very important in booking if it is wrong the patient will arrive on the wrong date, which will result in both the*	*1."The date must be selected from the lookup menu" 2."The date needs to be in the future"*

				consultant and patients time being wasted. To reduce this, we will validate it so only appropriate dates will be selectable.	
ConsultantList.java	*allConsultants[]*	***Type***	*Must be an object of a Consultant*	*This array needs to be validated as it will contain all the consultant entities when the program is running and will be used to search and sort them. If a wrong entity was to be passed into the list it would also be included, which we don't want included.*	*"Data entered was not an object of a Consultant"*
Management.java	*managmentID*	***1.Length*** ***2.Format*** ***3.Presence***	*1.Must be 11 characters* *2.must follow M"3 letters of surname" number* *3.Must be required of the object*	*This field will need to be validated because it will contain the most important field for this entity as it will uniquely identify it on the system if it was to be incorrect it would result in the entity being sorted and searched for in the unintended place.*	*1." Your staffID is 11 characters long"* *2." staffID has been entered wrong"* *3." A staffID is required"*
ManagmentList.java	*allManagment[]*	***Type***	*Must be an object of Management*	*This array needs to be validated as it will contain all the management entities when the program is running and will be used to search and sort them. If a wrong entity was to be passed into the list it would also be included, which we don't want included.*	*"Data entered was not an object of Management"*
Patient.java	*pateintID*	***1.Length*** ***2.Format*** ***3.Presence***	*1.Must be 11 characters* *2.must follow P"3 letters of surname" number* *3.Must be present in field*	*This field is important for the entity because it will be the key field to uniquely identify that patient if it was to be in the incorrect format that does not follow the universal standard it could result in it being I the unintended position when being sorted onto the system to minimise from this occurring we will validate it.*	*1." Your pateintID is 11 characters long"* *2." pateintID has been entered wrong"* *3." A pateintID is required"*

Class	Field	Val. Type	Val. Rule Description	Justification	Error Message
Patient.java	*houseNum*	***Presence***	*Must be present in field*	*This field needs to be validated because it might be needed for billing or mailing reasons, if it was incorrect it might get posted to the wrong address, however I have decided to make it a string to allow for house names to be entered as some buildings use names rather than numbers.*	*"The house number was not entered"*
Patient.java	*houseStreet*	***Presence***	*Must be present in field*	*The house street will need to be validated because it will allow for the correct location for where the house is located, while it may not necessarily be need for validation the rest of the patient's address is validated so the complete address might as well be validated to ensure consistency.*	*"The house street was not entered"*
Patient.java	Postcode	***Length***	*Must be 7 characters*	*The length of the postcode needs to be validated because it is set to a universal length of 7 characters consisting of letters and numbers. As it will be used to help locate an address it is important that it is correct.*	*"The postcode is not the correct length"*
Patient.java	contactNumber	***Type***	*Must be String*	*As this will be used to contact the patient it is critical that this information is correct to reduce errors when entering the information, I will only allow strings to be entered. If the number played a more vital role on my system, I would increase the validation but as it is used to just hold the data, I believe a type check is satisfactory.*	*"The contact number is not a string data Type"*

Class	Field	Val. Type	Val. Rule Description	Justification	Error Message
Patient.java	bloodtype	***Presence*** ***Lookup***	*Must be present in field*	*This a very important field to validate, as there are only a discrete number of blood types on the system a lookup check is most suited as the patient will have to select a blood type rather than having to manually enter it, this will greatly reduce in error when inputting data for the field.*	*"The blood type was not entered"*
PatientList.java	*allPatient[]*	***Type***	*Must be an object of patient*	*This array needs to be validated as it will contain all the patient entities when the program is running and will be used to search and sort them. If a wrong entity was to be passed into the list it would also be included, which we don't want included.*	*"Data entered was not an object of Patient"*
Admission.java	*admissionId*	***1.Length*** ***2.Format*** ***3.Presence***	*1.Must be 11 characters* *2.must follow A"3 letters of surname" number* *3.Must be present in field*	*This field is important for the entity because it will be the key field to uniquely identify that admission if it was to be in the incorrect format that does not follow the universal standard it could result in it being I the unintended position when being sorted onto the system to minimise from this occurring we will validate it.*	*1." Your admissionId is 11 characters long"* *2." admissionId has been entered wrong"* *3." An admissionId is required"*
Admission.java	medication	***Presence***	*Must be present in field*	*The prescription, like blood type is another key field to make sure it is correct, one way to do this is to validate the data this can be done through a presence check as if a prescription is not needed it should be stated rather than be left blank, as it would be clear that no medication is needed, rather having the consultant left to assume that.*	*"The Prescription was not entered"*

Class	Field	Val. Type	Val. Rule Description	Justification	Error Message
Admission.java	listOfSymptoms[]	***Type***	*Must be Strings*	*The information in the array will be validated to ensure that the symptoms entered are correct and are the right data type otherwise integers or other data types could be passed into the array.*	*"Data entered was not strings*
Admission.java	currentDiagnosis	***Presence***	*Must be present in field*	*As the admission is centred around a particular ailment it is important that it is validated to make sure that it is present on the system otherwise anyone new to that particular admission will be unaware to what the patient has.*	*"The current diagnosis was not entered"*
AdmissionList. java	*allAdmissions[ ]*	***Type***	*Must be an object of Admission*	*This array needs to be validated as it will contain all the admission entities when the program is running and will be used to search and sort them. If a wrong entity was to be passed into the list it would also be included, which we don't want included.*	*"Data entered was not an object of Admission"*
Admission.java	dosage	***1.Presence*** ***2.Range*** ***3.Lookup***	*Must be present in field* *2.Must be nonlethal amount* *3.Dosage must be reselected from list*	*The field needs to be validated to allow for a correct amount of medication* *The medication needs to be validated to ensure the correct amount was entered and not beyond the most suitable range.* *To limit the room for human error by forcing the user to select the dosage this will prevent lethal amounts from being chosen.*	*"dosage was not entered"* *"A lethal amount was supplied"* *"A dosage was not chosen from the sizes provided."*

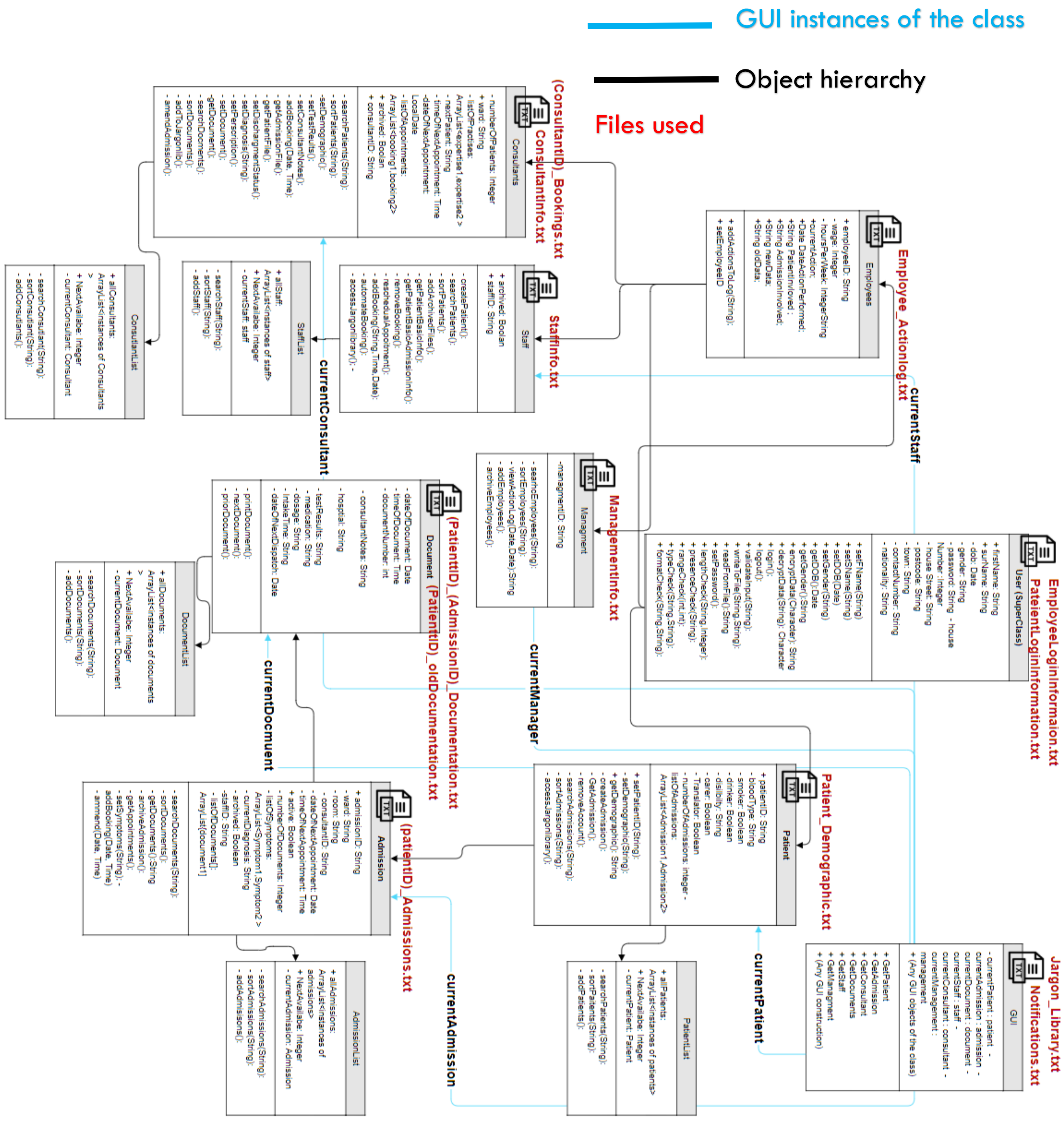
Class	Field	Val. Type	Val. Rule Description	Justification	Error Message
Admission.java	intakeTime	***1.Type*** ***2.Range***	*1.Must be of local time data type* *2.Must be at a present time*	*This needs to be validated to ensure that the field allows for a correct time for the medication to be taken otherwise the patient may consume the medicine at a time which is dangerous i.e. around meals.*	*"The time entered was not of the type time"* *"The time selected does not follow 24-hour clock"*
Admission.java	dateOfNextDispatch	***1.Lookup*** ***2.Range***	*1.must be preselected* *2. must be date in future*	*Finally, the date for next dispatch should be validated to allow the medicine to be retrieved for the patient and gives the patient the next date they should return if they need more.*	*"Data was not selected from available dates"* *"Date chosen was in the past"*
Document.java	*documentID*	***1.Length*** ***2.Format*** ***3.Presence***	*1.Must be 11 characters* *2.must follow D"3 letters of surname" number* *3.Must be present in field*	*This field is important for the entity because it will be the key field to uniquely identify that document if it was to be in the incorrect format that does not follow the universal standard it could result in it being I the unintended position when being sorted onto the system to minimise from this occurring we will validate it.*	*1." Your documentID is 11 characters long"* *2." documentID has been entered wrong"* *3." A documentID is required"*
Document.java	*consultantNotes*	***Presence***	*Field must not be left blank*	*This field needs to be validated as it will be the main field for passing on information between consultant and patient, therefore in order for an individual document to be meaningful there needs to be text included.*	*"For a Document to exist there needs to be text in order for meaning to be construed ."*
Document.java	*testResults*	***Presence***	*Field must not be left blank*	*This field needs to be validated as it will be the main field for passing on information between consultant and patient, therefore in order for an individual document to be meaningful there needs to be text included.*	*"For a Document to exist there needs to be text in order for meaning to be construed ."*

Class	Field	Val. Type	Val. Rule Description	Justification	Error Message
Document.java	*allDocuments[]*	***Type***	*Must be an object of document*	*This array needs to be validated as it will contain all the document entities when the program is running and will be used to search and sort them. If a wrong entity was to be passed into the list it would also be included, which we don't want included.*	*"Data entered was not an object of document"*

3.5 Processes

3.51 Class Diagram

A .png copy of this image will also be attached in the folder to allow for further viewing in detail.



3.52 All process Links with Data/Files

Objective # and Name (Investigation)	Explanation of Process and role within the system	Array / Files Used
1. Input/select symptoms into expert system	The process of this objective is to allow the patient or user the ability to select or enter the symptoms they are experiencing. The data entered will have to be stored in order to be able to be accessed at future dates. The role of the objective is to allow for a diagnosis to be determined from the selected data, by allowing for the user to both enter and select symptoms we can allow for some control for data entry but not prohibit symptoms that I have not considered.	**Symptoms[] –** To hold all the symptoms entered by the user, in one easy to access data structure. This will be compared against the systems symptoms later on.
2. Determine suggestion	The process is to compare symptoms previously entered in the previous objective and send this to the consultant. If a basic suggestion can be created it will otherwise it will be up to the consultant to assume the role. Once the diagnosis it will then have to be saved to the patient's demographic. The role of this will allow the consultant to have an early idea of what the problem will be. It will also allow for the system to suggest appropriate consultants to be used for the admission.	**Symptoms[] –** To hold all the symptoms entered by the user, in one easy to access data structure. This will be compared against the systems symptoms later on. **Diagnosablesymptoms[][] –** holds an array of symptoms per ailment in every array, will be used to compare to the user's symptoms
3. Generating new patients	The process will consist of creating a new object under the patient class and adding the fields currently entered by the staff entity. It will allow for a new patient user to be created containing all the attributes that the patient in real life poses. Its role will be to add new patients onto the system, otherwise only a discrete number of patients will have access when a new patient has been added a new collection of files for that patient will also be created.	**(patientID)_Admissions.txt –** This file will store all the admission information regarding the patient on the system. This will be unique to the patient. **Patient_Demographic.txt –** This file will need to be utilised in order for the demographic, personal, information to be stored on the system. On here the string will be correctly inserted into the location. **AllPatients[] –** a list of all patients on the system they will be stored in an array of the object patient

4. Generating a new admission	The process of this objective is to allow new admissions to be filled in and used. This will allow for multiple admissions to occur at the same time and be separate from each other Once created this will cause a new entry to be written to the patient's admission file and cause a new documentation file to be created containing the new documents that will be created for this new admission. The role of this objective is to allow new documentation and bookings to be created individually for each ailment the patient has, by having an individual documentation file every admission it will allow for separation between ailments and prevent confusion.	**symptoms[] –** To hold all the symptoms entered by the user, in one easy to access data structure **PateintID_Admissions.txt** - This file will store all the admission information regarding the patient on the system. This will be unique to the patient.
5. Booking a new appointment	The process of this objective is to generate a date time and location for the patient to come and visit the consultant at a moment that does not interfere with any other patient visits. Its role is to make sure when reading the file containing the information not to allow for already booked times to be selected this will be done by listing dates that have already been selected and removing them from the time sheet.	**listofAppointments[] –** The array will be used to hold all the instances of the booking instance on the data this will just be the admission object with the details of the appointment only being available.
6. Login users	The process will be using the credentials entered by the user and the file containing login details and comparing the user details with the details in the file. If they match, they are logged in, if they are wrong they are declined. Its role is to determine if the user is allowed onto the system by comparing what the user enters with what is already on the file.	**allPateintLoginDetails[] –** This will be an array of all the patient's login details on the system in it will be two items concatenated in a string containing the username and password, it differs from the employee to reduce the need of searching the entirety of the system **allemployeeLoginDetails[]** - This will be an array of all the employee's login details on the system in it will be two items concatenated in a string containing the username and password, it differs from the patients array to reduce the need of searching the entirety of the system

7. Display menu options	The process of this objective is to visually display all the information to the user in a way that results in them being able to navigate the system with little guidance from anyone else. Its role is to provide a visual display containing the information needed to allow for a correct input and interaction between the UI and user.	No files or arrays are used
8. Add employees	The process of this objective is to generate new employees onto the system this would be done by creating a new user then setting the user as a type of employee and assigning all the respective attributes to it. Once done it would be added to the staff info file containing all the respective information. Its role is to allow new employees onto the system and expand the number of staff working at Euxton. It will also assign them attributes defining each employee as an individual with different attributes such as wage mimicking real life.	**practisesList[] –** an attribute of the consultant this will hold all the expertise of the consultant in one coherent data structure **allConsultants[] –** An array used to hold all the consultants together in one place on the system, they will be stored under the custom object of consultant **StaffInfo.txt –** This text file will be used to hold all the information regarding the staff entity on the system. This will combine the demographic and work based attributes and store them on a line per employee basis **allStaff[] –** An array used to collectively hold every staff entity on the system into one place. The data type will of course be the object of staff allowing for each entity to retain all the correct values on in the system **EmployeeInfo.txt –** This file will be used in order to store the information entered by the management entity
9. Archive employees	The process of this objective is to disable employee interactions with the system by removing the ability for them to log in and access other features. This will be done by setting an attribute by management which results in their login privileges to be revoked by delegating them of the login file Its role is for security reasons to make sure that if an employee leaves Euxton they cannot regain access to the system as their details won't be there they can't log in.	EmployeeLoginInformaion.txt - This will be an array of all the employee's login details on the system in it will be two items concatenated in a string containing the username and password, it differs from the patients array to reduce the need of searching the entirety of the system

Objective # and Name (Investigation)	Explanation of Process and role within the system	Array / Files Used
10. Sort for employees	The process of this objective is to allow the user to sort the list of employees in the array. It does this by retrieving the employees from file, saving them to file then allowing user input to determine the order in which they are sorted. It does this by using the insertion sort and inserting the employee into the correct place. Its role is to allow the array or essentially the file to be put into the correct order determined by the user.	**allEmployees[] –** While other arrays such as AllStaff will only retain the staff class, all employees will contain every employee on the system. However as the array is specific to one data type we will have to use the first instance in the object hierarchy that all the entities collectively inherit the same methods and attributes, this obviously being the employee class.
11. Search for employees	The process of this objective is to use the search value entered by the user and compare it to the list of items in the array, it will then perform a binary search to locate the position the entity is located allowing for further manipulation of the file. The search consists of comparing the search value with the midpoint and disregarding the side that is irrelevant until the entity is found. Its role is to find the user the location in which the entity is stored at and return it to them for further use. If it is not present in the array an error message is shown.	**allEmployees[]** - While other arrays such as AllStaff will only retain the staff class, all employees will contain every employee on the system. However as the array is specific to one data type we will have to use the first instance in the object hierarchy that all the entities collectively inherit the same methods and attributes, this obviously being the employee class.
12. View an employee's transaction log	The process of this objective is to retrieve the actions performed by the individual employee between two set moments of time, the employee desired to be inspected and the two dates are entered as input and are passed through into objective 13A. Its role is to allow the management staff to see the interactions the other employees have with patient documentation.	No files or arrays are used

13. Read transaction log from file	The process of this objective is to read every line from file from the selected start and end points, this is achieved by reading the line and comparing the date the action was performed with the end date the management staff has intended if it lies within the time scale it will be outputted to the user. Its role is to allow the management entity to see all the actions the employee has done regarding the patient's data and see what was done to it.	**employeeID_Actionlog.txt –** This file is unique is such it is instance specific while this may seem initially a problem this file overtime will become **very** large and so it would ultimately be best that every instance of employee receives their own file due to the fact every large action will be recorded on here.
13. Write transaction log to file	The process of this objective is to record the data the employee has changed regarding the patient's information and will write it to file along with other attributes like the date what admission what time and the original date that was included. Its role is to record actions performed by the employee and monitor user interactions with sensitive information to prevent data integrity.	**(employeeID)_Actionlog.txt** - This file is unique is such it is instance specific while this may seem initially a problem this file overtime will become **very** large and so it would ultimately be best that every instance of employee receives their own file due to the fact every large action will be recorded on here.
14. Staff can sort for patients	The process of this objective is to allow the user to sort the list of patients in the array. It does this by retrieving the patients from file, saving them to file then allowing user input to determine the order in which they are sorted. It does this by using the insertion sort and inserting the patient into the correct place. Its role is to allow the array or essentially the file to be put into the correct order determined by the user.	**allPatients**[] - a list of all patients on the system they will be stored in an array of the object patient

Objective # and Name (Investigation)	Explanation of Process and role within the system	Array / Files Used
15. Staff can search for a patient	The process of this objective is to use the search value entered by the user and compare it to the list of items in the array, it will then perform a binary search to locate the position the entity is located allowing for further manipulation of the file. The search consists of comparing the search value with the midpoint and disregarding the side that is irrelevant until the entity is found. Its role is to find the user the location in which the entity is stored at and return it to them for further use. If it is not present in the array an error message is shown.	**allPatients**[] - a list of all patients on the system they will be stored in an array of the object patient
16. View patient details	The process of this objective is to allow the staff entity to read the patient demographic and receive the attributes from the patient this occurs by running the previous objective to find the patient and then accessing the attributes the entity possess. Its role is to allow the staff entity to read of the information the patient has and output the attributes to them.	**allPatients**[] - a list of all patients on the system they will be stored in an array of the object patient
17. Add archived notes from old system	The process of this objective is to read the scan of the documents and assign a filename to the image along with accompanying information like the admission. All of this information is then concatenated and then encrypted and then written to file. Its role is to allow external scans of documentation that are from the old system to be read and viewed. This is done by scanning the document and then creating a suitable filename for it.	**(patientID)_ oldDocumentation.txt –** This file will be utilised to have all the older documents on the system to be read and kept on the new one, a different file is used to handle the different format of the older documents

18. Amend bookings	The process of this objective is to allow the user to retrieve a patient's booking information by reading it from file and then allowing its attributes to be altered. Once this is done it is then saved to the action log and also encrypted and then overwrites the original line. Its role is to allow adjustments to bookings if it needs to occur for whatever reason. It also allows for new information to be added if a field was not originally included.	**(ConsultantID)_Bookings.txt –** This file will be used to store all the bookings on the consultant has on the system, not only will this be utilised as a reference to see upcoming appointments it will also be used as an archive for all prior ones too.
19. View patient bookings	The process of this objective is to run objective 40 which retrieves the desired booking and then outputs all the non-confidential attributes to the user. Its role is to allow the staff entity to view the attributes of the booking but not see the fields that are irrelevant or confidential.	No files or arrays are used
20. View patient admissions	The process of this objective is to retrieve the patient in particular and allow the user to select the admission they want to see information for. Similar to objective 20 they will only show the non-confidential information regarding the admission. Its role is to allow the user to see all the relevant attributes about the admission that concern them. The rest is excluded.	No files or arrays are used
21. Have patients view their Admissions and Demographic information	The process of this objective is retrieve all the attributes regarding the patient's demographic or admission. This occurs through selection and then retrieving the intended attributes from the object and outputting them to be viewed. Its role is to give the patient the option to view either parts of their account fully and see all attributes regarding each section.	**listOfSymptoms[]** - To hold all the symptoms entered by the user, in one easy to access data structure

22. Amend demographic information	The process of this objective is to run objective 41 to find the intended demographic and then retrieve the individual attributes stored on the single line. The user will then be able to select the field they want altering. After this the array containing all the demographic information is then concatenated and encrypted to then overwrite the old information. Its role is to allow changes to be made to the demographic so that the most relevant information can be held as some fields like address are not permanent and change.	**patientDetails[]** – This array will be used to alter information about the demographic, while the record structure of the data type is also another good use of this when reading from file the information needs to be split up form one large string. The most suitable way to store this data would be in the form of an array. Patient_Demographic.txt – This will be the main file of the system this will hold the entirety of all the demographic information of every patient on the system. Where every line will be specific for one patient. This will be used in a way where every attribute is concatenated with every other and then formed into one large string.
23. Validate information	The process of this objective is to receive user input along with the specific condition needs to be met which then returns an output determining if the condition has been met. Its role is to make sure the data is correct and that data redundancy is minimalised. Along with loss of integrity.	No files or arrays are used
23. A) Presence Check	The process of this objective is to make sure the field that has been passed through has some data and is not empty, selection occurs during this to see if the character length is greater than zero and then returns a Boolean value with true being accepted and false being denied. Its role is to make sure any required fields of data are filled in and are not left blank.	No files or arrays are used
23. B) Type Check	The process of this objective is to retrieve both the data by the user along with the desired data type. Once this occurs the user's input is compared to make sure it is the correct data type and will return a Boolean value respective to if the condition has been met or not. Its role is to make sure the user input matches the data type of the intended field such as local time.	No files or arrays are used

23. C) Format Check	The process of this objective is to retrieve the user's input and also the format in which the data should follow, after this it will compare the two values to check that the input follows the condition and will the return a Boolean data type with regards to if the statement was met or not. Its role is to make sure the user's input matches the format desired by the specific field if it is not is then rejected and will wait until a correct input has been entered.	No files or arrays are used
23. D) Range Check	The process of this objective is to retrieve the user's input along with the range of desired values, then selection occurs comparing the input between the start and end of the range to make sure it lies within the values. If the condition is met a Boolean that is true is returned if it does not lie within the range, a Boolean that is false is returned. Its role is to make sure the user's input lies within the range of the desired set of integers and accept the input if it follows this or deny it if it doesn't.	No files or arrays are used
23. E) Lookup Check	The process of this objective is to retrieve the user's input and the list of data that is acceptable. Here iteration occurs going through every index of the array comparing if the user's input matches the data if it does a Boolean that is true is returned, if no values match the input a Boolean that is false is returned Its role is to make sure that the user's input is in the list of acceptable data if it isn't the data is rejected.	**listOfOptions[] –** This will be used as a parameter for the system, here the list of available options will be passed through using this array. This will then be searched through being compared against the users selected value to make sure that it exists in the array.

Objective # and Name (Investigation)	Explanation of Process and role within the system	Array / Files Used
23. F) Length Check	The process of this objective is to retrieve the number of characters of the input and the desired number of characters. Selection then occurs comparing the two values, if the condition is met the Boolean value that is returned is set true. If the condition has not been met the value is set to false Its role is to make sure that user's input follows the number of characters desired of that field.	No files or arrays are used
24. View bookings in entirety	The process of this objective is to retrieve the desired consultant and run objective 40 to find the booking containing the desired patient. Once found the attributes of the booking are then outputted to the patient Its role is to allow the user to see the full booking information regarding a particular patient for the consultant	No files or arrays are used
25. Add bookings	The process of this objective is to allow the user to enter the field they want add to the new booking, they do this until all the fields have been entered then all these attributes are then concatenated and are then written to file Its role is to allow new bookings to be created on the system and then be saved onto the consultants file	**ConsultantID_Bookings.txt –** This file will be used to store all the bookings on the consultant has on the system, not only will this be utilised as a reference to see upcoming appointments it will also be used as an archive for all prior ones too.

Objective # and Name (Investigation)	Explanation of Process and role within the system	Array / Files Used
26. A) Sort documents	The process of this objective is to allow the user to sort the list of documents in the array. It does this by retrieving the documents from file, saving them to file then allowing user input to determine the order in which they are sorted. It does this by using the insertion sort and inserting the document into the correct place Its role is to allow the array or essentially the file to be put into the correct order determined by the user	**allDocuments[] –** This array will be used to store all the documents on the system. The data type in particular will be of the class document allowing for all attributes regardless if they are used or not to be inherited.
26. B) Search documents	The process of this objective is to use the search value entered by the user and compare it to the list of items in the array, it will then perform a binary search to locate the position the entity is located allowing for further manipulation of the file. The search consists of comparing the search value with the midpoint and disregarding the side that is irrelevant until the entity is found. Its role is to find the user the location in which the entity is stored at and return it to them for further use. If it is not present in the array an error message is shown	**allDocuments[]** - This array will be used to store all the documents on the system. The data type in particular will be of the class document allowing for all attributes regardless if they are used or not to be inherited. A it is by object the search field can vary which will be super handy in development when the search order only needs to vary the search field.
27. Print documents	The process of this objective is to retrieve the admission in particular the document is held then the document is retrieved from the admission. After this the document waits until a printer is available before sending it to be printed of. Its role is to allow the user to select the document they want printing off from the admission, once a printer is free it is sent to it	No files or arrays are used

28. Search for patients	The process of this objective is to retrieve the list of patients the consultant has and then retrieve the desired patient. A binary search is then performed on the list removing irrelevant sections of the list until the final value is either the desired one or the end of the list has been reached Its role is to allow the consultant to retrieve the index the patient is located on in the array	**allpatients[]** - a list of all patients on the system they will be stored in an array of the object patient
29. Consultant can view patient files	The process of this objective is to use the prior objective to find the location of the object then to allow further options to be selected to navigate the consultant through the rest of the patient's menu Its role is to allow for further navigation of the patient's data and information on the system	No files or arrays are used
30. View patient Demographic information	The process of this objective is to output all the attributes associated along with the demographic Its role is to show to the user all the fields that are in the demographic	No files or arrays are used
31. Sort admission	The process of this objective is to allow the user to sort the list of documents in the array. It does this by retrieving the documents from file, saving them to file then allowing user input to determine the order in which they are sorted. It does this by using the insertion sort and inserting the document into the correct place Its role is to allow the array or essentially the file to be put into the correct order determined by the user	**allAdmissions[] –** This will be used to store all the admissions the patient has on the system onto one coherent place. This will allow for fast access over a wide range of data. The data type in particular will be the object of admission and should allow for a wide range of fields to sort by.

Objective # and Name (Investigation)	Explanation of Process and role within the system	Array / Files Used
32. Edit Prescriptions	The process of this objective is to retrieve the correct location of where the prescription is located on the patients document text file. From this the consultant will be able to then individually amend each individual field. Finally, the new data is saved to the action log and is also encrypted and overwrites the old data in the documentation log Its role is to allow the consultant the ability to edit the prescription if changes need to occur	**allPatients**[] - a list of all patients on the system they will be stored in an array of the object patient **allAdmisisons**[] - This will be used to store all the admissions the patient has on the system onto one coherent place. This will allow for fast access over a wide range of data. The data type in particular will be the object of admission so should be useful when making direct changes to individual fields. **listOFPrescrition**[] – The list of prescriptions will be needed to collectively store these entities together. This will be utilised as an attribute of the admission entity after being used in memory. **listOFDates**[] – This will be subjective to what the consultants time schedule is like, however it will contain all the available dates the patient can choose from on the system. **(patientID)_(admissionID) _Documentation.txt** – This text file will be used to store all the induvial documents for a particular admission. It is on a per admission basis due the investigation where some admissions had over 100 documents. While inconvenient it should greatly reduce search times having no longer to consider other admissions.
33. Add Admission information	The process of this objective is to retrieve the desired patient and find the desired admission from this any new fields are brought up for the consultant to enter the data into those fields. The action is written to the consultant's action log and the new data is added to the file after it has been encrypted Its role is to allow the user to add information to the admission, fields may be left due to human error or could be left until the data in the field is eventually known like diagnosis etc.	**allPatients**[] - a list of all patients on the system they will be stored in an array of the object patient **patientID_Admisison.txt** - This file will store all the admission information regarding the patient on the system. This will be unique to the patient. **allAdmisisons**[] - This will be used to store all the admissions the patient has on the system onto one coherent place. This will allow for fast access over a wide range of data. The data type in particular will be the object of admission so should be useful when amending information.

34. Edit Admission information	The process of this objective is to retrieve the desired admission and bring up all the information regarding the admission. All the fields are decrypted and then saved into an array the user is then allowed to enter the new data in the fields. The new data is then saved into the action log and then saved over in the array. Finally, the admission is then concatenated encrypted and written to file Its role is to allow updates to occur to the admission, this should allow in case fields that are not permanent like address change	**allAdmisisons[]** - This will be used to store all the admissions the patient has on the system onto one coherent place. This will allow for fast access over a wide range of data. The data type in particular will be the object of admission so should be useful when amending information. **allPatients[]** - a list of all patients on the system they will be stored in an array of the object patient **patientID_Admisison.txt** - This file will store all the admission information regarding the patient on the system. This will be unique to the patient.
35. Add notes	The process of this objective is to allow the document type for the user to be declared and then accordingly set the individual fields to what the user inputs, this data is then validated and encrypted and then saved to the employee's action log. Its role is to allow the user to add documentation onto the patient's admission and allow new information to be saved.	**(patientID) _(AdmissionID)_ Documentation.txt** - This text file will be used to store all the induvial documents for a particular admission. It is on a per admission basis due the investigation where some admissions had over 100 documents. While inconvenient it should greatly reduce search times having no longer to consider other admissions. **listOFDosages[] –** This will be used when adding a prescription in particular but as the attribute is common to all will be used here. This will show the employee the quantities available to choose from. **listOFDates[]** - This will be subjective to what the consultants time schedule is like, however it will contain all the available dates the patient can choose from on the system.
36. Encrypting data before being written to file	The process of this objective is to pass the string through the objective then the ASCII value for each index is then moved up 5 values. The word is then concatenated and then saved as a whole string after this it is then returned to the desired field. Its role is to allow any data to be written to file to be encrypted to make sure it is secure and allows sensitive data to be protected.	No files or arrays are used

37. Decrypting that has been read from file	The process of this objective is to retrieve the encrypted data from this the index of the individual string are then converted back into the original data, this is done by reducing the ascii value down by 5 after this it is then concatenated again and then returned to the user Its role is to allow the user to decrypt any data that is read from file, without it the user would be unable to understand any data returned from file	No files or arrays are used
38. Using the Jargon library	The process of this objective is to retrieve a desired word the list of definitions is also retrieved then a binary search is performed to find the location of the word. After it has been found the world has been split the word is then returned Its role is to allow the definition to be retrieved and then showed to the user	**Jargon_Library.txt –** The text file will be used as a permanent store on the system to hold all the meanings to the words. **allDefinitions[]** - The array will contain all the definitions on the system, it will be a string consisting of a concatenation of the word then followed by the definition. The string will be split in the front end of the system as it only job is to be displayed to the user
39. Adding to the Jargon library	The process of this objective is to allow the consultant to enter the word they want to use and the definition is then concatenated. Finally, a binary search is used to find the location, afterwards it is then written to the correct location Its role is to allow new words to be added by the consultant otherwise no new definitions will be included besides the initial declaration	**Jargon_Library.txt** - The text file will be used as a permanent store on the system to hold all the meanings to the words. This will be used to write any definitions to file. **allDefinitions[] –** The array will contain all the definitions on the system, it will be a string consisting of a concatenation of the word then followed by the definition. The string will be split in the front end of the system as it only job is to be displayed to the user

Objective # and Name (Investigation)	Explanation of Process and role within the system	Array / Files Used
40. Search through Bookings	The process of this objective is to use the search value entered by the user and compare it to the list of items in the array, it will then perform a binary search to locate the position the entity is located allowing for further manipulation of the file. The search consists of comparing the search value with the midpoint and disregarding the side that is irrelevant until the entity is found. Its role is to find the user the location in which the entity is stored at and return it to them for further use. If it is not present in the array an error message is shown	**allBookings[]** - The array will be used to hold all the instances of the booking instance on the data this will just be the admission object with the details of the appointment only being available. **(consultantID)_Bookings.txt –** This file will be used to store all the bookings on the consultant has on the system, not only will this be utilised as a reference to see upcoming appointments it will also be used as an archive for all prior ones too. **allEmployees[]** - While other arrays such as AllStaff will only retain the staff class, all employees will contain every employee on the system. However as the array is specific to one data type we will have to use the first instance in the object hierarchy that all the entities collectively inherit the same methods and attributes, this obviously being the employee class.
41. Search through demographic information	The process of this objective is to use the search value entered by the user and compare it to the list of items in the array, it will then perform a binary search to locate the position the entity is located allowing for further manipulation of the file. The search consists of comparing the search value with the midpoint and disregarding the side that is irrelevant until the entity is found. Its role is to find the user the location in which the entity is stored at and return it to them for further use. If it is not present in the array an error message is shown	**allDemographic[] –** Here the list of every line from the file will be read from file and inserted here. As the first set of characters is the username there is no issue with finding the correct values to sort the data. **Patient_Demographic.txt** - This will be the main file of the system this will hold the entirety of all the demographic information of every patient on the system. Where every line will be specific for one patient. This will be used in a way where every attribute is concatenated with every other and then formed into one large string.

3.52 Processing Routines

Before the code is shown, due to the modular nature of the system and the environment I developed the system will usually utilise one main instance of an entity and will keep it in memory for the entire duration of the program, nulling the contents to make way for a new instance if necessary. This is important to note as the pseudocode will include instructions such as ***retrieve*** this will be down to two different reasons. The first one is that the process called utilises parameters and these will be stated using the retrieve command. To add to this the other instance is when on the set up of the program, and the current main instance of the class is used it will be retrieved instead of having to search an array to locate an address for it. For instance objective 21 where the entity is already chosen but isn't initially clear.

Pseudocode for Objectives 1 to 10

1. Entering symptoms

```
START program
    SET Array symptoms[4]//creates an empty array for the patient to enter the symptoms
    SET counter AS 0//sets the counter for the symptoms array as zero
    WHILE input {Text field} IS PRESENT//repetition statement determining if the input is valid
    using a presence check
        THEN OUTPUT "Please enter symptoms you are experiencing"//asks patient for the
        symptoms they are experiencing
        SET Symptoms[counter] AS RUN OBJECTIVE 23A(INPUT)//input symptoms into the
        array before this it passed through the validation method of objective 23.A
        SET listOfSympotms AS Symptoms[counter]//sets the attribute to the input
        Counter ++//increments
    END While//ends while loop
    SET admission.ListOfSymptoms AS listOfSympotms//saves the admission attribute of the list of
    symptoms to the entity
END program
```

2. Suggesting basic diagnosis

```
START program
    SET diagnosis//This will be used to hold the patient's diagnosis
    SET diagnosislength//Number of ailments the system has with symptoms
    SET userSymptomLength//Number of symptoms patient has
    SET innerCounter//for the array of user symptoms
    SET counter//For the array with system symptoms
    SET similarities//when comparing symptoms this will count the number of similarities there are
    between lists
    SET max//declares the maximum value which has similarities between the user's symptoms and
    what the system has
    RETREIVE symptoms[]//imports the patient's symptoms
    RETREIVE diagnosablesymptoms[Diagnosis][symptom]//imports systems symptoms this will be
    the database of values and will be hardcoded as they won't change to prevent data
    inconsistency when comparing symptoms
```

```
        FOR counter FOR RANGE diagnosislength//goes through all system symptoms
            FOR innerCounterFOR RANGE userSymptomLength//goes through all patients'
            symptoms
                IF symptoms[innerCounter] IS EQUAL TO diagnosablesymptoms[counter][innerCounter]
                //compares to see if patients are similar to the induvial symptom
                    THEN similarities ++//increments the number of symptoms that are
                    the same for the diagnosis
                ENDIF
            NEXT FOR//moves onto next symptom
            IF buffer IS GREATER THAN max//compares the number of the same symptoms the
            patient has with the system if they have more this is ran
                THEN similarities IS max//updates the number of the same symptoms
                diagnosis IS diagnosablesymptoms[counter][]//updates diagnosis
            ENDIF
        NEXT FOR//moves on to the next possible diagnosis
        IF max IS GREATER THAN 3//checks to see if the number of same symptoms is greater than 3
        to make sure diagnosis has at least some accuracy
            RUN OBJECTIVE 4 (symptoms[],diagnosis)//moves onto admission page to generate a
            new admission
        ENDIF
        ELSE IF IS LESS THAN 3//checks to see if the number of same symptoms is less than 3 to make
        sure diagnosis has at least some accuracy
            SET Diagnosis AS "inconclusive"//clears diagnosis as not enough evidence
            RUN OBJECTIVE 4 (symptoms[],diagnosis)//moves onto admission page to generate a
            new admission

        ENDIF
    END program
```

3. Generating New Patients

```
START program
    SET NEW user
    SET user.firstname AS RUN OBJECTIVE 23A (INPUT {Text field})//Sets the attribute of the
    new object to the input, it is also validate for the user's presence on input
    SET user.surname AS RUN OBJECTIVE 23B (INPUT {Text field})//Sets the attribute of the
    new object to the input, it is also validate for the fields data type
    SET user.DOB AS RUN OBJECTIVE 23C (INPUT {Date Picker})//Sets the attribute of the new
    object to the input, it is also validate for the fields format of input
    SET user.gender AS RUN OBJECTIVE 23E (INPUT {Combo box})//Sets the attribute of the
    new object to the input, it is also validate for the fields input through checking it is an available
    option
    SET User AS patient//Sets this new object of user as a patient as well
    SET randomNUM (0,9999999)//generates a random number
    SET patient.patientID AS P + user.surname(0,2) + randomNUM//creates a new ID for the
    patient using their surname and a random number
    SET patient.houseNum AS RUN OBJECTIVE 23A (INPUT {Text field})//Sets the attribute of the
    new object to the input, it is also validate for the user's presence on input
    SET patient.houseStreet AS RUN OBJECTIVE 23A (INPUT {Text field})//Sets the attribute of
    the new object to the input, it is also validate for the user's presence on input
    SET patient.Postcode AS RUN OBJECTIVE 23F (INPUT {Text field})//Sets the attribute of the new object to the input, it is also validate for the field's length being suitable
    SET patient.contactNumber AS RUN OBJECTIVE 23F (INPUT {Text field})//Sets the attribute of the new object to the input, it is also validate for the field's length being suitable
    SET patient.nationality AS RUN OBJECTIVE 23B (INPUT {Combo box})//Sets the attribute of the new object to the input, it is also validate for the fields data type
    SET patient.smoker AS INPUT {Radio button}//Sets the attribute of the new object to the input
    SET patient.drinker AS INPUT {Radio button}//Sets the attribute of the new object to the input
    GENERATE NEW text file (patientID)_Admissions.txt//creates a new text file for the patient's admissions
    SET address AS RUN HASHING ALGORITHM (patientID)//runs a hashing algorithm to return the address of the patient in the table
    ADD New Patient TO AllPatients[address]//Adds the patient to the list of patients
    SET text AS CONCATENATE patient//concatenates all the fields together
    SET encryptedtext AS (RUN OBJECTIVE 38(text))//encrypts the text
    RUN WRITETOFILE(Patient_Demographic.txt, encryptedtext)//writes the patient to file
    If user IS employee//The system checks if the user was an employee
        THEN RUN OBJECTIVE 13B(employeeID, actionToLocation, patientID, AdmissionID, OrginalData, NewData)//this passes the action just performed into the objective to add it to their action log
END program
```

4. Generating New Admissions

```
START program
    RETREIVE patient//Finds the relevant patient to access the file
    RETRIEVE symptoms[]//Finds the symptoms that they have entered
    RETRIEVE diagnosis//Finds the corresponding diagnosis for the symptoms
    SET filename AS patient.patientID + _Admissions//creates a suitable filename to find the actual file
    RETREIVE AS filename.txt//Uses the filename to find the text file and returns all the admissions
    SET randomNUM (0,9999999)//Generates a random number
    SET admission.admissionID AS A + user.surname(0,2) + randomNUM creates a new ID for the patient's admission using their surname and a random number
    SET admission.ward AS INPUT {Text field}//Sets the attribute of the new object to the input
    SET admission.consultantID AS RUN OBJECTIVE 23E(INPUT {Combo box}, allConsultants[]) //Sets the consultant of the new admission to the input, performs a lookup check so the desired consultant is selected from the drop down box.
    SET admission.staffID AS RUN OBJECTIVE 23E(INPUT {Combo box}, allConsultants)//Sets the staff of the new admission to the input, performs a lookup check so the desired consultant is selected from the drop down box.
    SET admission.active AS TRUE//Sets the attribute of the new object to a hardcoded value so it therefore doesn't need validating or inputting.
    SET admission.listOfSymptoms AS symptoms[]//uses the imported list of symptoms and saves it to the variable, no need to validate it as it already has been.
    SET admission.currentDiagnosis AS diagnosis//uses the imported diagnosis and saves it to the variable, no need to validate it as it already has been.
```

```
        SET text AS CONCATENATE Admission//concatenates all the fields together
        SET encryptedtext AS (RUN OBJECTIVE 38(text))//encrypts the text
        RUN WRITETOFILE(filename.txt, encryptedtext )//Writes the new object to file, serial so it doesn't matter which order
        RUN OBJECTIVE 13B(employeeID, actionToLocation, patientID, AdmissionID, OrginalData, NewData)//this passes the action just performed into the objective to add it to their action log
END program
```

5. Booking a new appointment

```
START program
        RETRIEVE availableDates(consultantID)//retrieves all the available dates from the consultant attribute
        SET datepicker AS availableDates//using the retrieved dates they are saved to the visual object
        SET NEW booking//creates a new object of booking
        SET booking.dateOfNextAppointment AS INPUT {Date picker}//saves the date selected as that date, no need to validate only available days are available to be selected
        RETRIEVE availableTimes(consultantID,date)//retrieves all the available times from the selected date
        SET timepicker AS availableTimes//using the retrieved times they are saved to the visual object
        SET booking.timeOfNextAppointment AS INPUT {Time picker}//saves the time selected from the user's choice
        SET room AS INPUT{Text field}//saves the room inputted
        ADD NEW booking TO Conssultant.listofAppointments[] AT index nextPosition//Adds the booking to the consultants list
        nextPosition ++//increments the pointer to the appointments
        SET text AS CONCATENATE appointment //concatenates all the fields together
        SET encryptedtext AS (RUN OBJECTIVE 38(text))//encrypts the text
        RUN WRITETOFILE(filename.txt, encryptedtext  )//Writes the new object to file
        RUN OBJECTIVE 13B(employeeID, actionToLocation, patientID, AdmissionID, OrginalData, NewData)//this passes the action just performed into the objective to add it to their action log

END program
```

6. login users

```
START program
        SET Loggedin AS FALSE//Sets the login credentials as false to loop the initial login screen
        WHILE loggedin IS FALSE//loops the screen so allows for multiple attempts
                THEN SET useType AS INPUT{button}//user selects the type of user they want to login as
                IF useType IS patient//selection to determine if this type of user was selected
        OPEN file(pateintLogincredentials.txt)//Opens the file containing all the
        patient's credentials
        //Below pulls all the credentials from file to the array
        WHILE end IS NOT REACHED//A while loop checking the end hasn't been
        reached
                THEN SET allPateintLoginDetails[counter] AS READ line//Reads the
                line and saves it to the array of details
                Counter ++//increments the counter
        END WHILE
        SET username AS INPUT{Text field}//saves the users input to the variable
        SET password AS INPUT{Text field}//saves the users input to the variable
        CHECK username AND password AGAINST allPateintLoginDetails[]//performs
        a binary search on the list with the name
        IF ALLOWED//checks to see if the credentials are correct
                THEN LOGIN//allows the user onto the system
                LOAD HOMEPAGE//generates the user homepage
        IF NOT ALLOWED OUTPUT ERROR MESSAGE// Outputs an error message if
        login credentials are wrong
ENDIF
ELSE
        //Below pulls all the credentials from file to the array
        THEN OPEN file(employeeLogincredentials.txt)//Opens the file containing all
        the patient's credentials
        WHILE end IS NOT REACHED//A while loop checking the end hasn't been
        reached
                THEN SET allEmployeeLoginDetails[counter] AS READ line//Reads
                the line and saves it to the array of details
                Counter ++//increments the counter
        END WHILE

        IF useType IS consultant// selection to determine if this type of user was
        selected
                SET username AS INPUT{Text field}//saves the users input to the
                variable
                SET password AS INPUT{Text field}//saves the users input to the
                variable
                CHECK username AND password AGAINST
                allEmployeeLoginDetails[]//performs a binary search on the list with
                the name
                IF ALLOWED//checks to see if the credentials are correct
                        THEN LOGIN//allows the user onto the system
                        LOAD HOMEPAGE//generates the user homepage
                IF NOT ALLOWED OUTPUT ERROR MESSAGE// Outputs an error
                message if login credentials are wrong
        ENDIF
        IF useType IS admin// selection to determine if this type of user was selected
                SET username AS INPUT{Text field}//saves the users input to the
                variable
                SET password AS INPUT{Text field}//saves the users input to the
                variable
                CHECK username AND password AGAINST
                allEmployeeLoginDetails[]//performs a binary search on the list with
                the name
                IF ALLOWED//checks to see if the credentials are correct
                    THEN LOGIN//allows the user onto the system
                    LOAD HOMEPAGE//generates the user homepage
                IF NOT ALLOWED OUTPUT ERROR MESSAGE// Outputs an error
                message if login credentials are wrong
            ENDIF
            IF useType IS management// selection to determine if this type of user was
            selected
                SET username AS INPUT{Text field}//saves the users input to the
                variable
                SET password AS INPUT{Text field}//saves the users input to the
                variable
                CHECK username AND password AGAINST
                allEmployeeLoginDetails[]//performs a binary search on the list with
                the name
                IF ALLOWED//checks to see if the credentials are correct
                    THEN LOGIN//allows the user onto the system
                    LOAD HOMEPAGE//generates the user homepage
                IF NOT ALLOWED OUTPUT ERROR MESSAGE//Outputs an error
                message if login credentials are wrong
        ENDIF
    ENDWHILE
END program
```

7. Display Menu Options

NO Pseudocode, objective is summarises the process of projecting a visual interface to the user, pseudocode cant be generated for this.

8. Generating New Employees

```
START program
    RETRIEVE NEW user//Once a new user is generated, they will be retrieved
    SET type AS INPUT{command line}//The user when determine if this new user is a consultant or
    staff
    SET user AS Employee//The system will set the new user as an employee
    SET randomNUM (0,9999999)//generates a random number
    SET employee.employeeID AS E + user.surname(0,2) + randomNUM//System assigns the user
    a employeeID
    SET employee.wagePerWeek AS RUN OBJECTIVE 23A (INPUT{command line})//Sets the
    attributes for the wage as the input
```

```
        SET employee.hoursPerWeek AS RUN OBJECTIVE 23D (INPUT{command line})//Saves the
        employees hourly work rate from the input, validates it to make sure not working for excessive
        periods of time
        SET index//declares the index the employee will be written to in the file.
        IF type IS consultant//If management decides to save the employee as a consultant the
        following will set the rest of the attributes
            THEN SET employee AS consultant//updates the entity to obtain the new attributes
            SET randomNUM (0,9999999)//generates a random number
            SET consultant.staffID AS C + user.surname(0,2) +randomNUM//generates the id for
            the entity
            SET Ward AS RUN OBJECTIVE 23A  INPUT{command line}//saves the attribute as
            the input they have entered validated to check it is correct
            SET archived AS FALSE//saves that the archived field as false and they are currently
            working
            SET counter AS 0//adds an empty counter for array indexes in the iterative section
            DO//Will immediately loop until a condition has been satisfied
                SET practices AS INPUT{command line}//saves the attribute as the input they
                have entered
                SET practisesList[counter] AS practices//will save the input to the array
                Counter++//Will increment and move onto the next index
            UNTILL INPUT IS NULL//termination condition until no more input is left to be entered
            SET index AS RUN OBJECTIVE 11(allStaff[],EmployeeID)//calls a method which finds
            the index of the new employee
            ADD NEW consultant TO allConsultants[index]//adds the new entity to the array of
                entities
            SET text AS CONCATENATE Consultant //concatenates all the fields together
            SET encryptedtext AS (RUN OBJECTIVE 38(text))//encrypts the text
            RUN WRITETOFILE(StaffInfo.txt, text )//Writes the entity to staff file also
        END IF
        ELSE IF type Is staff// If management decides to save the employee as staff the following will
        set the rest of the attributes
            THEN SET user AS staff// updates the entity to obtain the new attributes
            SET randomNUM (0,9999999)// generates a random number
            SET staff.staffID AS S + user.surname(0,2) + randomNUM// generates the id for the
            entity
            SET archived AS FALSE//saves that the archived field as  false and they are currently
            working
            SET index AS RUN OBJECTIVE 11(allStaff[],EmployeeID)//calls a method which finds
            the index of the new employee
            ADD NEW staff TO allStaff[index]//adds the new entity to the array of entities at
                the correct index
            SET text AS CONCATENATE staff //concatenates all the fields together
            SET encryptedtext AS (RUN OBJECTIVE 38(text))//encrypts the text
            RUN WRITETOFILE(EmployeeInfo.txt, encryptedtext )// Writes the entity to staff file
                also
        RUN OBJECTIVE 13B(employeeID, actionToLocation, patientID, AdmissionID, OrginalData,
        NewData)
        //this passes the action just performed into the objective to add it to their action log
        ENDIF
END program
```

9. Archiving employees

```
START program
        SET employeeID AS INPUT{Text field}//allows the management entity to retrieve an employee
        needing to be archived
        SET index AS RUN OBJECTIVE 11(allStaff[],EmployeeID)//calls a method which finds the
        index of the new employee
        SET employee AS allStaff[index]//calls upon the index to declare the correct instance of
        the employee to use
        SET employee.archived AS TRUE//updates the attribute on the entity
        REVOKE ACCESS TO system//removes the ability for the entity to use the system in any way
        REMOVE employee.loginDetails FROM EmployeeLoginInformaion.txt//Removes the file login
        details from the text file preventing the employee from logging in
END program
```

10. Sort employees

```
START program
        RETRIEVE allEmployees[]//Brings up all the employees in the array
        RETRIEVE sortOrder//retrieves the sort order required to sort the array of entities
        SET arrayLength AS length.allEmployees[]//Finds the length of the array or number of entities
        SET outerLoop//Declares the outer loop counter of the array
        FOR outerloop FOR (arrayLength-1)//Declares a for loop running for the length of the array -
        1
        SET currentEmployee AS allEmployees[outerloop]// Sets the entity to be the one in the array
        this indexes contents will be overwritten so this preserves the data
        SET inserted AS FALSE//sets the inserted value as false allowing
                SET inserted AS FALSE//sets the inserted value as false allowing for iterations to occur
                SET innerLoop AS (outerloop – 1)//sets the inner loop counter as an index behind
                DO
                        IF currentEmployee IS sortOrder THAN allEmployees[innerLoop]//Uses the
                        retrieved sort order to have selection on the two adjacent list
                                THEN allEmployees[innerLoop +1] IS allEmployees[innerLoop]// sets
                                the new index above as the index behind
                                SET innerLoop AS innerLoop – 1]//De-increments the inner counter,
                                essentially moves down an index
                                SET allEmployees[innerLoop +1] AS currentEmployee//Sets the old
                                value to the current employee being swapped over
                        ENDIF
                        ELSE IF currentEmployee IS sortOrder THAN allEmployees[innerLoop]//
                        selection determining if the employee is in the correct position
                THEN SET inserted AS TRUE//updates the variable to allow the loop
                to break
            ENDIF
        UNTIL innerLoop IS sortOrder TO 0 AND inserted IS FALSE//while loop condition
        determining  if the employee in the array is in order
    NEXT outerloop// increments on the next index or employee to order
END program
```

Pseudocode for Objectives 11 to 20

11. Search employees

```
START program
    RETRIEVE allEmployees[]//retrieves the entire array of all employees
    RETRIEVE searchEmployee//Returns the employee needing to be searched
    SET lengthOfArray AS length.allEmployees[]//sets the length of the array
    SET startPosition//Declares the start position index variable
    SET middlePosition//Declares the middle position index variable
    SET endPosition AS lengthOfArray// Declares the end position index variable
    Initially setting it as the last index of the employee list
    SET found AS FASLE//sets the value to be false as nothing is found as of yet
    SET Position//sets the found position of the employee if found within the array
    DO
        SET middlePosition AS (startPosition + endPosition) DIV 2//Finds the middle position of
        the array by averaging the start and end points
        IF searchEmployee IS allEmployees[middlePosition]//Selection determining if the search
        value is the same as the employee at the current index
            THEN SET found AS TRUE//updates the found attribute
            SET position AS middlePosition//sets the position of the index where it was
            found
            OUTPUT employee found at index (position)//outputs that the employee was
            found
            RETRIEVE employee//retrieves all the data regarding the employee
        ENDIF
        ELSE IF searchEmployee IS LESS THAN allEmployees[middlePosition]// Selection
        determining if the search value is less than the employee at the current index
            THEN SET endPosition AS middlePosition – 1//correctly adjusts the new end
            position removing any irrelevant employees
        ENDIF
        ELSE IF searchEmployee IS GREATER THAN allEmployees[middlePosition]//
        Selection determining if the search value is greater than the employee at the current
        index
            THEN SET startPosition AS middlePosition + 1// correctly adjusts the new start
            position removing any irrelevant employees
        ENDIF
    UNTIL found IS TRUE OR endPosition IS LESS THAN startPosition//iteration condition of
    termination until found or that the start index exceeds the end point indicating that no other
    values exist and the search value is not there
    IF found IS FASLE//selection if seeing that the employee was not found
    THEN OUTPUT no employee found//informs that the employee was not found
END program
```

12. View an employee's transaction log

```
START program
        SET employeeID AS INPUT{Text field}//allows the management entity to retrieve an employee
        needing to be archived
        SET index AS RUN OBJECTIVE 11(allStaff[],EmployeeID)//calls a method which finds the
        index of the new employee
        SET employee AS allStaff[index]//calls upon the index to declare the correct instance of
        the employee to use
        SET startPostision AS INPUT//the management entity will enter the date they want to start with
        SET endPostision AS INPUT// the management entity will enter the date they want to end at
        SET employee AS INPUT//They will finally enter the employee they wish to view
        RUN OBJECTIVE 13A(employee,startPostision,employee)//This will run the objective which
        passes all three variables through when ran it will output the correct information
END program
```

13A. Read transaction log from file

```
START program
        RETRIEVE employee//this retrieves the patient the management entity wishes to view
        RETRIEVE startPostision//This retrieves the start position of the transaction log the management
        entity wants to start at
        RETRIEVE endPostision// This retrieves the start position of the transaction log the management
        entity wants to end at
        SET filename AS employee.employeeID//this sets the filename to the correct one associated
        with the patient
        OPEN filename
        FOR line FROM startPostision TO endPostision//Declaration of the for loop running from the
                start point to the end point for every line. This will go through every action performed
                by the employee between the selected date
                THEN READ filename_Actionlog.txt//This will then read that line
                SET text AS Line//The line read is then saved to the text variable
                SET decryptedtext AS (RUN OBJECTIVE 38(text))//Encrypts the data allowing it to be
                securely written to file
                OUTPUT decryptedtext//the text is then outputted to the management entity
        NEXT line//The line now increments to the next one
```

```
    CLOSE filename
END program
```

13B. write transaction log to file

```
START program
    RETRIEVE employeeID//retrieves the employee ID to correctly write to the file
    SET localDate//sets the date the action was performed
    SET localTime//sets the time the action occurred
    RETRIEVE actionToLocation//retrieves the action and where it was applied to
    RETRIEVE patientID//retrieves the patient ID to allow for tracing of who's information was
    affected
    RETRIEVE AdmissionID//retrieves the admission ID to trace what admission was affected
    RETRIEVE OrginalData//retrieves the data in the field before it was amended
    RETRIEVE NewData//retrieves the new data that was added
    SET text AS localDate + "," + localTime + "," + actionToLocation + "," + patientID + "," +
    AdmissionID + "," + OrginalData + "," + NewData//Concatenates all the data retrieved and
    saves it as a single string
    SET filename AS employeeID +"_Actionlog.txt"//Sets the correct filename to that of the
    employeeID
    RETRIEVE filename//retrieves the action log for the individual employee
    OPEN filename//opens the file allowing for amending of data
    SET encryptedtext AS (RUN OBJECTIVE 38(text))//encrypts the text
    WRITE encryptedtext TO filename//Writes the concatenated data to file, as it is ordered by
    date then time no ordering issues occur so can be added to the bottom of the file
    CLOSE filename//
END program
```

14. Staff entity can sort for patients

```
START program
    RETRIEVE allPatients[]//Brings up all the patients in the array
    RETRIEVE sortOrder//retrieves the sort order required to sort the array of entities
    SET arrayLength AS length.allPatients[]//Finds the length of the array or number of entities
    SET outerLoop//Declares the outer loop counter of the array
    FOR outerloop FOR (arrayLength-1)//Declares a for loop running for the length of the array -
    1
    SET currentPatient AS allPatients [outerloop]//Sets the entity to be the one in the array this
    indexes contents will be overwritten so this preserves the data
        SET inserted AS FALSE//sets the inserted value as false allowing for iterations to occur
        SET innerLoop AS (outerloop – 1)//sets the inner loop counter as an index behind
        DO
            IF currentPatient IS sortOrder THAN allPatients[innerLoop]//Uses the retrieved
            sort order to have selection on the two adjacent list
                THEN allPatients[innerLoop +1] IS allPatients[innerLoop]// sets the new
                index above as the index behind
                SET innerLoop AS innerLoop – 1]//De-increments the inner counter,
                essentially moves down an index
                SET allPatients[innerLoop +1] AS currentPatient//Sets the old value to
                the current patient being swapped over
            ENDIF
            ELSE IF currentPatient IS sortOrder THAN allPatients[innerLoop]// selection
            determining if the patient is in the correct position
                THEN SET inserted AS TRUE//updates the status of the variable
            ENDIF
        UNTIL innerLoop IS sortOrder TO 0 AND inserted IS FALSE//while loop condition
        determining  if the patient in the array is in order
    NEXT outerloop// increments on the next index or patient to order
END program
```

15. Staff entity can search for a patient

```
START program
    RETRIEVE allPatients[]//retrieves the entire array of all patients available
    RETRIEVE searchPatient//Returns the patient needing to be searched
    SET lengthOfArray AS length.allPatients[]//sets the length of the array
    SET startPosition//Declares the start position index variable
    SET middlePosition//Declares the middle position index variable
    SET endPosition AS lengthOfArray// Declares the end position index variable
    Initially setting it as the last index of the employee list
    SET found AS FASLE//sets the value to be false as nothing is found as of yet
    SET Position//sets the found position of the patient if found within the array
    DO
```

```
        SET middlePosition AS (startPosition + endPosition) DIV 2//Finds the middle position of the array by averaging the start and end points
        IF searchPatient IS allPatients[middlePosition]//Selection determining if the search value is the same as the patient at the current index
            THEN SET found AS TRUE//updates the found attribute
            SET position AS middlePosition//sets the position of the index where it was found
            OUTPUT patient found at index (position)//outputs that the patient was found
            RETRIEVE patient//retrieves all the data regarding the patient
        ENDIF
        ELSE IF searchPatient IS LESS THAN allPatients[middlePosition]// Selection determining if the search value is less than the patient at the current index
            THEN SET endPosition AS middlePosition – 1//correctly adjusts the new end position removing any irrelevant employees
        ENDIF
        ELSE IF searchPatient IS GREATER THAN allPatients[middlePosition]// Selection determining if the search value is greater than the patient at the current index
            THEN SET startPosition AS middlePosition + 1// correctly adjusts the new start position removing any irrelevant patient
        ENDIF
    UNTIL found IS TRUE OR endPosition IS LESS THAN startPosition//iteration condition of termination until found or that the start index exceeds the end point indicating that no other values exist and the search value is not there
    IF found IS FASLE//selection if seeing that the employee was not found
        THEN OUTPUT no employee found//informs that the employee was not found
END program
```

16. View non confidential patient details

```
START program
    SET patientID AS INPUT//Uses the desired patient to search for and find by retrieving their associated data
    SET patient AS (RUN OBJECTIVE 15(allPatients[],patientID))//Creates a new object of patient using the patient found from the array
    OUTPUT patient.firstname//outputs attribute
    OUTPUT patient.surname// outputs attribute
    OUTPUT patient.houseNum// outputs attribute
    OUTPUT patient.houseStreet// outputs attribute user
    OUTPUT patient.Postcode// outputs attribute
    OUTPUT patient.contactNumber// outputs attribute
    OUTPUT patient.numOfAdmissions// outputs attribute
END program
```

17. Add archived notes from old system

```
START program
    RETRIEVE patient//retrieves the patient desired
    SET filename AS patient.patientID + "_" + oldDocumentation.txt//sets up the filename of the text file where all other scans are saved
    SET admisisonID AS INPUT{Text field}//Saves the admission ID so it can be used to distinguish scan names between admissions
    SET scanName AS INPUT{Text field}//Sets the scan name of the scan so it can be used to identify its contents
    SET date AS INPUT{date picker}//declares the date of the document to distinguish documents in the same admission that are of the same type
    RETRIEVE SCAN//this will retrieve the scan of the document
    SET scan.filename AS admisisonID + "_" + scanName + "_" + date//concatenates the three attributes and saves them as a string
    OPEN filename//opens the file to allow for it to be used
    SET encryptedtext AS (RUN OBJECTIVE 38(scan.filename))
    WRITE encryptedtext TO filename//Writes the concatenated string to file
    SET employeeID//sets the employeeID to the employee using the system
    SET actionTOloaction AS adding old document to admission//saves the action the employee is doing to adding old documentation
    SET originalData AS NULL//AS no original data was amended this is set to null
    SET NewData AS scan.filename//sets the new data to the scan's filename
    CLOSE filename//closes the file to prevent it from corrupting
    RUN OBJECTIVE 13B(employeeID, actionToLocation, patientID, AdmissionID, OrginalData, NewData)//this passes the action just performed into the objective to add it to their action log
END program
```

18. Amend bookings

```
START program
    RETRIEVE patient//finds the patient who's booking needs amending
    SET field AS INPUT{Text field}//the field wanting to be edited is chosen
    IF field IS timeOfNextApp//selection determining if this field needs to be edited
        THEN SET orignialData AS Admisson.timeOfNextApp//updates the original data variable to add to the action log
        SET Admisson.timeOfNextApp AS RUN OBJECTIVE 23E (INPUT{Time picker})//updates the field to contain the desired data
        SET newData AS Admisson.timeOfNextApp//uses the new data and saves it to an external variable
    END IF
    ELSE IF field IS dateOfNextApp//selection determining if this field needs to be edited
        THEN SET orignialData AS Admisson.dateOfNextApp//updates the original data variable to add to the action log
        SET Admisson.dateOfNextApp AS RUN OBJECTIVE 23E (INPUT {date
        picker})//updates the field to contain the desired data
        SET newData AS Admisson.dateOfNextApp//uses the new data and saves it to an
        external variable
    END IF
    ELSE IF field IS room//selection determining if this field needs to be edited
        THEN SET orignialData AS Admisson.dateOfNextApp//updates the original data
        variable to add to the action log
        SET Admisson.room AS INPUT {Text field}//updates the field to contain the desired
        data
        SET newData AS AS Admisson.dateOfNextApp//uses the new data and saves it to an
        external variable
    END IF
    ELSE IF field IS ward//selection determining if this field needs to be edited
        THEN SET orignialData AS Admisson.ward//updates the original data variable to add
        to the action log
        SET Admisson.ward AS RUN OBJECTIVE 23A (INPUT {Text field})//updates the field
        to contain the desired data
        SET newData AS Admisson.ward//uses the new data and saves it to an external
        variable
    END IF
    ELSE IF field IS consultant//selection determining if this field needs to be edited
        THEN SET orignialData AS Admisson.consultant//updates the original data variable to
        add to the action log
        SET Admisson.consultant AS RUN OBJECTIVE 23E+A (INPUT {Text field})//updates
        the field to contain the desired data
        SET newData AS Admisson.consultant//uses the new data and saves it to an external
        variable
    END IF
    SET actionToLocation AS amendment to booking field//sets the action to amending the booking
    field
    SET newText AS patientID + "," + admissionID + "," + timeOfNextApp + "," +
    dateOfNextApp + "," + ward + "," + room//concatenates all the fields together onto one
    string
    SET filename AS patient.ConsultantID + "_Bookings.txt"//sets the correct filename
    OPEN filename//opens the file to allow amendments
    SET line AS RUN OBJECTIVE 40(patientID)//finds the booking in the file running the objective
    SET text AS CONCATENATE staff //concatenates all the fields together
    SET encryptedtext AS (RUN OBJECTIVE 38(text))//encrypts the text
    RUN WRITETOFILE(EmployeeInfo.txt, encryptedtext,Line)// Writes the entity to staff file  at
    the correct line
    CLOSE filename//closes the file to prevent corruption
    RUN OBJECTIVE 13B(employeeID, localDate, localTime, actionToLocation, patientID,
    AdmissionID, OrginalData, NewData)//this passes the action just performed into the objective
    to add it to their action log
END program
```

19. View patient bookings

```
START program
    RETRIEVE  consultantID//retrieves the consultant who holds the information about the booking
    SET admission AS admissionList[(RUN OBJECTIVE 40(patientID))]//finds the admission in the
    list we are looking for along with them we receive all their attributes
    OUTPUT admission.dateOfNextApp//outputs to the user the attribute
    OUTPUT admission.ward//outputs to the user the attribute
    OUTPUT admission.room//outputs to the user the attribute
    OUTPUT admission.consultant//outputs to the user the attribute
END program
```

20. View patient's non restricted information about admissions

```
START program
    RETRIEVE patient//retrieves the patient we are interested in; admission doesn't need to be
    retrieved as it is linked to the patients account
    SET admission AS INPUT{button}//uses the input to determine the admission set by the user
    OUPUT admission.admissionID//outputs to the user the attribute
    OUPUT admission.ward//outputs to the user the attribute
    OUPUT admission.consultantID//outputs to the user the attribute
    OUPUT admission.dateOfNextApp//outputs to the user the attribute
    OUPUT admission.active//outputs to the user the attribute
    OUPUT admission.currentDiagnosis//outputs to the user the attribute
END program
```

Pseudocode for Objectives 21 to 30

21. Have patients view their Admissions and Demographic information

```
START program
    RETRIEVE patient//retrieves the patient we are interested in; admission doesn't need to be
    retrieved as it is linked to the patients account. It doesn't need to be searched as it is already
    in memory.
    SET choice AS INPUT{button}//users selects if they want to see the admission or the
    demographic
    IF choice IS Admission//if the user wants to see the admission this will run
        THEN SET AdmissionChoice AS INPUT{button}//The user selects which admission they
        want
        SET admission AS admissionChoice//The system retrieves the correct admission
        OUTPUT admission.admissionID//outputs to the user the attribute
        OUTPUT admission.ward//outputs to the user the attribute
        OUTPUT admission.consultantID//outputs to the user the attribute
        OUTPUT admission.staffID//outputs to the user the attribute
        OUTPUT admission.timeOfNextApp//outputs to the user the attribute
        OUTPUT admission.dateOfNextApp//outputs to the user the attribute
        OUTPUT admission.active//outputs to the user the attribute
        OUTPUT admission.Perscription//outputs to the user the attribute
        OUTPUT admission.room//outputs to the user the attribute
        OUTPUT admission.currentDiagnosis//outputs to the user the attribute
        OUTPUT admission.listOfSymptoms[]//outputs to the user the attribute
    ENDIF
    ELSE IF choice IS Demographic{button}//if the user wants to see the demographic this will run
        OUTPUT patient.patientID//outputs to the user the attribute
        OUTPUT patient.surname//outputs to the user the attribute
        OUTPUT patient.firstname//outputs to the user the attribute
        OUTPUT patient.houseNum//outputs to the user the attribute
        OUTPUT patient.houseStreet//outputs to the user the attribute
        OUTPUT patient.Postcode//outputs to the user the attribute
        OUTPUT patient.contactNumber//outputs to the user the attribute
        OUTPUT patient.nationality//outputs to the user the attribute
        OUTPUT patient.bloodtype//outputs to the user the attribute
        OUTPUT patient.smoker//outputs to the user the attribute
        OUTPUT patient.drinker//outputs to the user the attribute
        OUTPUT patient.disability//outputs to the user the attribute
        OUTPUT patient.numOfAdmissions//outputs to the user the attribute
    ENDIF
END program
```

22. Amend demographic information

```
START program
    SET patientID AS INPUT{Text field}//Declares the patientID from the input
    SET patient AS (RUN OBJECTIVE 41(patientID))//Runs the objective to retrieve the
    demographic
    RETURN LINE//returns the line the patient was in on the file
    SET patientDetails[] AS (split.patientDetails[])//Splits up the line to hold the individual data in
    each index
    SET detailToChange AS INPUT//User sets the field they wish to change
    IF detailToChange is patientID//selection to see if they want to amend this field
        THEN SET patientDetails[0] AS INPUT//sets the correct field in the array to the new
        data
    ENDIF
    ELSE IF detailToChange is surname//selection to see if they want to amend this field
        THEN SET patientDetails[1] AS INPUT//sets the correct field in the array to the new
        data
    ENDIF
```

```
        ELSE IF detailToChange is firstname//selection to see if they want to amend this field
                THEN SET patientDetails[2] AS RUN OBJECTIVE 23A(INPUT)//sets the correct field in
                the array to the new data
        ENDIF
        ELSE IF detailToChange is houseNum//selection to see if they want to amend this field
                THEN SET patientDetails[3] AS RUN OBJECTIVE 23A(INPUT)//sets the correct field in
                the array to the new data
        ENDIF
        ELSE IF detailToChange is houseStreet//selection to see if they want to amend this field
                THEN SET patientDetails[4] AS RUN OBJECTIVE 23A(INPUT)//sets the correct field in
                the array to the new data
        ENDIF
        ELSE IF detailToChange is Postcode//selection to see if they want to amend this field
                THEN SET patientDetails[5] AS RUN OBJECTIVE 23F(INPUT)//sets the correct field in
                the array to the new data
        ENDIF
        ELSE IF detailToChange is contactNumber//selection to see if they want to amend this field
                THEN SET patientDetails[6] AS RUN OBJECTIVE 23B(INPUT)//sets the correct field in
                the array to the new data
        ENDIF
        ELSE IF detailToChange is nationality//selection to see if they want to amend this field
                THEN SET patientDetails[7] AS INPUT//sets the correct field in the array to the new
                data
        ENDIF
        ELSE IF detailToChange is bloodtype//selection to see if they want to amend this field
                THEN SET patientDetails[8] AS INPUT//sets the correct field in the array to the new
                data
        ENDIF
        ELSE IF detailToChange is smoker//selection to see if they want to amend this field
                THEN SET patientDetails[9] AS INPUT//sets the correct field in the array to the new
                data
        ENDIF
        ELSE IF detailToChange is drinker//selection to see if they want to amend this field
                THEN SET patientDetails[10] AS INPUT//sets the correct field in the array to the new
                data
        ENDIF
        ELSE IF detailToChange is disability//selection to see if they want to amend this field
                THEN SET patientDetails[11] AS INPUT//sets the correct field in the array to the new
                data
        ENDIF
        FOR counter FROM 0 TO 11//creates a loop that will iterate 12 times for a total of 13 runs
                THEN SET text AS text + ”,” + patientDetails[counter]//merges the current text
                variable with the next data in the array, essentially re-joins the array back together
                just replaces the old data
        END FOR
        SET encryptedtext AS RUN OBJECTIVE 37(text)//encrypts the data
        OPEN Patient_Demographic.txt//opens the text file containing all demographic information
        WRITE encryptedtext TO Patient_Demographic.txt AT line//writes the encrypted data to file
        CLOSE Patient_Demographic.txt//closes file to prevent corruption
END program
```

23.A) Presence Check

```
START program
    RETRIEVE dataToValidate//the data intended to be checked is passed through
    SET Validated//initialises the validated variable
    IF dataToValidate length IS NOT 0//Checks to see if the variable is not empty
        THEN SET Validated AS TRUE//verifies that the input is allowed
    ENDIF
    ELSE IF dataToValidate length IS 0//Checks to see if the variable is empty
        THEN SET Validated AS FALSE//verifies that the input is allowed
    ENDIF
    RETURN(Validated)//returns the result from the check
END program
```

23.B) Type Check

```
START program
    RETRIEVE dataToValidate//the data intended to be checked is passed through
    RETRIEVE datatype//the data type is passed through
    SET Validated//initialises the validated variable
    IF dataToValidate IS datatype//Checks to see if the variable is matches the data type
        THEN SET Validated AS TRUE//verifies that the input is allowed
    ENDIF
    ELSE IF dataToValidate IS NOT datatype//Checks to see if the variable is not of the same
    data type
        THEN SET Validated AS FALSE//verifies that the input is allowed
    ENDIF
    RETURN(Validated)//returns the result from the check
END program
```

23.C) Format Check

```
START program
    RETRIEVE dataToValidate//the data intended to be checked is passed through
    RETRIEVE dataformat//the data format is passed through
    SET Validated//initialises the validated variable
    IF dataToValidate IS dataformat//Checks to see if the variable is matches the data format
        THEN SET Validated AS TRUE//verifies that the input is allowed
    ENDIF
```

```
        ELSE IF dataToValidate IS NOT dataformat//Checks to see if the variable is not of the same
        data format
            THEN SET Validated AS FALSE//verifies that the input is allowed
        ENDIF
        RETURN(Validated)//returns the result from the check
END program
```

23.D) Range Check

```
START program
        RETRIEVE dataToValidate//the data intended to be checked is passed through
        RETRIEVE start//the start point is passed through
        RETRIEVE end//the end point is passed through
        SET Validated//initialises the validated variable
        IF dataToValidate IS LESS THAN end AND GREATER THAN start//Checks to see if the
        variable lies within the range
            THEN SET Validated AS TRUE//verifies that the input is allowed
        ENDIF
        ELSE IF dataToValidate IS LESS THAN start OR GREATER THAN end//Checks to see if the
        does not variable lies within the range
            THEN SET Validated AS FALSE//verifies that the input is allowed
        ENDIF
        RETURN(Validated)//returns the result from the check
END program
```

23.E) Lookup Check

```
START program
        RETRIEVE dataToValidate//the data intended to be checked is passed through
        RETRIEVE listOfOptions[]//passes through all the available options to choose from
        SET Validated//initialises the validated variable
        SET length AS length.listOfOptions[]//finds the length of array
        SET index//declares the index counter for loop
        SET validated AS FALSE//initially sets validated as false if not found it will remain like this
        otherwise it will be updated
        FOR index IN RANGE OF length//declares the for loop that runs through entire array
            THEN IF dataToValidate IS listOfOptions[index]//selection comparing current value
            with user's data
                THEN Validated IS TRUE//if it matches must be in the array and is then
                validated
            ENDIF
        NEXT index//moves onto next index
        RETURN(Validated)//returns the result from the check
END program
```

23.F) Length Check

```
START program
    RETRIEVE dataToValidate//the data intended to be checked is passed through
    RETRIEVE minLength//retrieves the minimum length required
    SET Validated//initialises the validated variable
    IF dataToValidate Length IS LESS THAN minLength//comparison to see if the user input is less
    than the required amount
        THEN SET Validated AS FALSE//If this is true it sets validated as false and rejects
        input
    ENDF
    ELSE IF dataToValidate Length IS GREATER THAN minLength//comparison to see if the user
    input is greater than the required amount
        THEN SET validated AS TRUE//Accepts the variable if the selection condition is met
    ENDIF
    RETURN(Validated)//returns the result from the check
END program
```

24. View bookings in its entirety

```
START program
    RETRIEVE  consultantID//retrieves the consultant who holds the information about the booking
    SET patient AS patientList[(RUN OBJECTIVE 40(patientID))]//finds the patient in the file we
    are looking for along with them we receive all their attributes
    THEN SET AdmissionChoice AS INPUT{button}//The user selects which admission they want
    SET admission AS patient.admissionChoice//The system retrieves the correct admission
    OUPTUT admission.timeOfNextApp//outputs to the user the attribute
    OUTPUT admission.dateOfNextApp//outputs to the user the attribute
    OUTPUT admission.ward//outputs to the user the attribute
    OUTPUT admission.room//outputs to the user the attribute
    OUTPUT admission.consultantID//outputs to the user the attribute
    OUTPUT admission.admissionID//outputs to the user the attribute
    OUTPUT admission.otherInfo//outputs to the user the attribute
END program
```

25. Add bookings

```
START program
    SET field AS INPUT{Text field}//the field wanting to be edited is chosen
    IF field IS timeOfNextApp//selection determining if this field needs to be edited
```

```
        THEN SET Admisson.timeOfNextApp AS INPUT{Time picker}//updates the field to
        contain the desired data
    END IF
    ELSE IF field IS dateOfNextApp//selection determining if this field needs to be edited
        THEN SET Admisson.dateOfNextApp AS RUN OBJECTIVE 23E(INPUT){Date
        picker}//updates the field to contain the desired data
    END IF
    ELSE IF field IS room//selection determining if this field needs to be edited
        THEN SET Admisson.room AS RUN OBJECTIVE 23E(INPUT){Text field}//updates the
        field to contain the desired data
    END IF
    ELSE IF field IS ward//selection determining if this field needs to be edited
        THEN SET Admisson.ward AS RUN OBJECTIVE 23A(INPUT){Text field}//updates the
        field to contain the desired data
    ELSE IF field IS consultant//selection determining if this field needs to be edited
        THEN SET Admisson.consultant AS RUN OBJECTIVE 23E(INPUT){Text field}//updates
        the field to contain the desired data
    END IF
    SET newText AS patientID + "," + admissionID + "," + timeOfNextApp + "," +
    dateOfNextApp + "," + ward + "," + room//concatenates all the fields together onto one
    string
    SET encryptedtext AS RUN OBJECTIVE 37(newText )//encrypts the data
    SET filename AS patient.ConsultantID + "_Bookings.txt"//sets the correct filename
    OPEN filename//opens the file to allow amendments
    WRITE encryptedtext TO filename(Line)//writes the new booking to file
    CLOSE filename//closes the file to prevent corruption
END program
```

26A. Sort documents

```
START program
    RETRIEVE allDocuments[]//Brings up all the documents in the array
    RETRIEVE sortOrder//retrieves the sort order required to sort the array of entities
    SET arrayLength AS length.allDocuments[]//Finds the length of the array or number of entities
    SET outerLoop//Declares the outer loop counter of the array
    FOR outerloop FOR (arrayLength-1)//Declares a for loop running for the length of the array -
    1
    SET currentDocument AS allDocuments[outerloop]// Sets the entity to be the one in the array
    this indexes contents will be overwritten so this preserves the data
    SET inserted AS FALSE//sets the inserted value as false allowing
        SET inserted AS FALSE//sets the inserted value as false allowing for iterations to occur
        SET innerLoop AS (outerloop – 1)//sets the inner loop counter as an index behind
        DO
            IF currentDocument IS sortOrder THAN allDocuments[innerLoop]//Uses the
            retrieved sort order to have selection on the two adjacent list
                THEN allDocuments[innerLoop +1] IS allDocuments[innerLoop]// sets
                the new index above as the index behind
                SET innerLoop AS innerLoop – 1]//De-increments the inner counter,
                essentially moves down an index
                SET allDocuments[innerLoop +1] AS currentDocument//Sets the old
                value to the current document being swapped over
            ENDIF
            ELSE IF currentDocument IS sortOrder THAN allDocuments[innerLoop]//
            selection determining if the document is in the correct position
                THEN SET inserted AS TRUE//updates the variable to allow the loop
                to break
            ENDIF
        UNTIL innerLoop IS sortOrder TO 0 AND inserted IS FALSE//while loop condition
        determining  if the document in the array is in order
    NEXT outerloop// increments on the next index or document to order
END program
```

26B. Search documents

```
START program
    RETRIEVE allDocuments[]//retrieves the entire array of all documents available
    RETRIEVE searchDocument//Returns the document needing to be searched
    SET lengthOfArray AS length.allDocuments[]//sets the length of the array
    SET startPosition//Declares the start position index variable
    SET middlePosition//Declares the middle position index variable
    SET endPosition AS lengthOfArray//Declares the end position index variable
    Initially setting it as the last index of the document list
    SET found AS FASLE//sets the value to be false as nothing is found as of yet
    SET Position//sets the found position of the document if found within the array
    DO
        SET middlePosition AS (startPosition + endPosition) DIV 2//Finds the middle position of
        the array by averaging the start and end points
        IF searchDocument IS allDocuments[middlePosition]//Selection determining if the search
        value is the same as the document at the current index
            THEN SET found AS TRUE//updates the found attribute
            SET position AS middlePosition//sets the position of the index where it was
            found
            OUTPUT patient found at index (position)//outputs that the patient was found
            RETRIEVE patient//retrieves all the data regarding the document
        ENDIF
        ELSE IF searchDocument IS LESS THAN allDocuments[middlePosition]// Selection
        determining if the search value is less than the document at the current index
            THEN SET endPosition AS middlePosition – 1//correctly adjusts the new end
            position removing any irrelevant documents
        ENDIF
        ELSE IF searchDocument IS GREATER THAN allDocuments[middlePosition]//
        Selection determining if the search value is greater than the document at the current
        index
```

```
                THEN SET startPosition AS middlePosition + 1// correctly adjusts the new start
                position removing any irrelevant patient
            ENDIF
        UNTIL found IS TRUE OR endPosition IS LESS THAN startPosition//iteration condition of
        termination until found or that the start index exceeds the end point indicating that no other
        values exist and the search value is not there
        IF found IS FASLE//selection if seeing that the document was not found
            THEN OUTPUT no document found//informs that the document was not found
END program
```

27. Print documents

```
START program
    RETRIEVE admission//the system will retrieve the desired admission
    SET document AS INPUT{button}//the user will select the document they want printed
    RETRIEVE document//The document is found on the system and returned
    SEND document TO PRINTER//this document is then set to the printer
    THEN ADD document TO spool//AS documents may currently be printing to save on resources
    it is sent to the spool where it will wait until ready
    DO//indicates the iterative process of waiting until the condition is satisfied
        THEN WAIT UNTIL AVAILABLE//allows for other documents to be printed
    UNTIL PRINTER IS FREE//termination condition is to see if printer is available for use
    PRINT document//the document is then printed
END program
```

28. Consultant can Search for patients

```
START program
    RETRIEVE allpatients[]//retrieves the entire array of all patients available that they look after
    RETRIEVE searchPatient//Returns the patient needing to be searched
    SET lengthOfArray AS length.allpatients[]//sets the length of the array
    SET startPosition//Declares the start position index variable
    SET middlePosition//Declares the middle position index variable
    SET endPosition AS lengthOfArray//Declares the end position index variable
    Initially setting it as the last index of the patient list
    SET found AS FASLE//sets the value to be false as nothing is found as of yet
    SET Position//sets the found position of the patient if found within the array
    DO
        SET middlePosition AS (startPosition + endPosition) DIV 2//Finds the middle position of
        the array by averaging the start and end points
        IF searchPatient IS allpatients[middlePosition]//Selection determining if the search
        value is the same as the patient at the current index
            THEN SET found AS TRUE//updates the found attribute
```

```
                SET position AS middlePosition//sets the position of the index where it was found
                OUTPUT patient found at index (position)//outputs that the patient was found
                RETRIEVE patient//retrieves all the data regarding the patient
            ENDIF
            ELSE IF searchPatient IS LESS THAN allpatients[middlePosition]// Selection determining if the search value is less than the patient at the current index
                THEN SET endPosition AS middlePosition – 1//correctly adjusts the new end position removing any irrelevant documents
            ENDIF
            ELSE IF searchPatient IS GREATER THAN allpatients[middlePosition]// Selection determining if the search value is greater than the document at the current index
                THEN SET startPosition AS middlePosition + 1// correctly adjusts the new start position removing any irrelevant patient
            ENDIF
        UNTIL found IS TRUE OR endPosition IS LESS THAN startPosition//iteration condition of termination until found or that the start index exceeds the end point indicating that no other values exist and the search value is not there
        IF found IS FASLE//selection if seeing that the patient was not found
            THEN OUTPUT no patient found//informs that the patient was not found

END program
```

29. Consultant can view patient files

```
START program
    SET patientID AS INPUT{Text field}//the consultant will be able to enter the patient they want to see
    SET patient AS RUN OBJECTIVE 28(patientID, listofConsulatns'patients)//runs the objective and finds the intended patient from the consultants own list
    SET menuOption AS INPUT{Button}//the consultant will decide the menu they want to see
    IF menuOption IS demographic//selection determining if they want to see this screen
        THEN RUN OBJECTIVE 30()//If they want to see the demographic of the patient's version of it will be loaded
    ENDIF
    ELSE IF menuOption IS admissions//selection determining if they want to see this screen
        THEN OUTPUT patient admissionMenu//If they want to see the admission of the patient's version of it will be loaded
    ENDIF
END program
```

30. View patient Demographic information

```
START program
    OUTPUT patient.patientID//outputs to the user the attribute
    OUTPUT patient.surname//outputs to the user the attribute
    OUTPUT patient.firstname//outputs to the user the attribute
    OUTPUT patient.houseNum//outputs to the user the attribute
    OUTPUT patient.houseStreet//outputs to the user the attribute
    OUTPUT patient.Postcode//outputs to the user the attribute
    OUTPUT patient.contactNumber//outputs to the user the attribute
    OUTPUT patient.nationality//outputs to the user the attribute
    OUTPUT patient.bloodtype//outputs to the user the attribute
    OUTPUT patient.smoker//outputs to the user the attribute
    OUTPUT patient.drinker//outputs to the user the attribute
    OUTPUT patient.disability//outputs to the user the attribute
    OUTPUT patient.numOfAdmissions//outputs to the user the
END program
```

Pseudocode for Objectives 31 to 40

31. Sort admission

```
START program
    RETRIEVE allAdmissions[]//Brings up all the admissions in the array
    RETRIEVE sortOrder//retrieves the sort order required to sort the array of entities
    SET arrayLength AS length.allAdmissions[]//Finds the length of the array or number of entities
    SET outerLoop//Declares the outer loop counter of the array
    FOR outerloop FOR (arrayLength-1)//Declares a for loop running for the length of the array -
    1
    SET currentAdmission AS allAdmissions[outerloop]// Sets the entity to be the one in the array
    this indexes contents will be overwritten so this preserves the data
    SET inserted AS FALSE//sets the inserted value as false allowing
        SET inserted AS FALSE//sets the inserted value as false allowing for iterations to occur
        SET innerLoop AS (outerloop – 1)//sets the inner loop counter as an index behind
        DO
            IF currentAdmission IS sortOrder THAN allAdmissions[innerLoop]//Uses the
            retrieved sort order to have selection on the two adjacent list
                THEN allAdmissions[innerLoop +1] IS allAdmissions[innerLoop]// sets
                the new index above as the index behind
                SET innerLoop AS innerLoop – 1]//De-increments the inner counter,
                essentially moves down an index
                SET allAdmissions[innerLoop +1] AS currentAdmission //Sets the old
                value to the current admission being swapped over
            ENDIF
            ELSE IF currentAdmission IS sortOrder THAN
            allAdmissions[innerLoop]//selection determining if the admission is in the
            correct position
```

```
                    THEN SET inserted AS TRUE//updates the variable to allow the loop
                    to break
                ENDIF
            UNTIL innerLoop IS sortOrder TO 0 AND inserted IS FALSE//while loop condition
            determining  if the admission in the array is in order
        NEXT outerloop// increments on the next index or admission to order

END program
```

32. Edit Prescriptions

```
START program
        RETRIEVE patient FROM allPatients[]//retrieves the patient from the array containing the
        consultant's patients
        RETRIEVE admission FROM allAdmisisons[]//retrieves the admission from the
        SET filename AS patient.patientID + "_" + admission.admissionID + "_" +
        Documentation//concatenates the admission information to form a suitable filename
        OPEN filename//opens the correct file
        SET line AS READ prescription//reads the line containing the prescription
        SET ARRAY listOFPrescrition[] AS line.SPLIT//splits the line-up storing the contents to an array
        SET choiceOfPerscription AS INPUT{Text field}//the user will set the prescription field they
        want to edit
        IF choiceOfPerscription IS medication//if the user sets the field choice to this, it will allow for
        this field to be amended
            THEN SET origninalData AS listOFPrescrition[0]//sets the original data to the one held
            in the array
            SET Data AS INPUT{Text field}//the data the consultant wants to use is entered
            SET validated AS RUN OBJECITVE 23.A(Data)//a Boolean variable is set up which
            has the validated variable returned
            IF validated IS TRUE//if validated is true the user input is accepted and the condition
            is satisfied
                THEN SET newData AS data//sets the new data attribute to the Initial data
                entered by the user
                SET listOFPrescrition[0] AS newData//adds the new data to the array
            ENDIF
        ENDIF
        ELSE IF choiceOfPerscription IS dosage//if the user sets the field choice to this, it will allow for
        this field to be amended
            THEN SET origninalData AS listOFPrescrition[1]//sets the original data to the one held
            in the array
            SET Data AS INPUT{Text field}//the data the consultant wants to use is entered
            SET validated AS RUN OBJECITVE 23.A(Data) AND RUN OBJECTIVE
            23.D(Data,start,end) AND RUN OBJJECTIVE// 23.E(Data listOFDosages[])//this
            selection statement passes through out the new data through all three of the validation
            functions making sure that they are correct if they all return true or accept the
            condition is met
            IF validated IS TRUE//if validated is true the user input is accepted and the condition is satisfied
                THEN SET newData AS data//sets the new data attribute to the Initial data entered by the user
                SET listOFPrescrition[1] AS newData//adds the new data to the array
            ENDIF
        ENDIF
        ELSE IF choiceOfPerscription IS intakeTIme//if the user sets the field choice to this, it will allow for this field to be amended
            THEN SET origninalData AS listOFPrescrition[2]//sets the original data to the one held in the array
            SET Data AS INPUT{Text field}//the data the consultant wants to use is entered
            SET validated AS RUN OBJJECTIVE 23.E(Data listOFDates[]) AND RUN OBJECTIVE 23.D(Data,start,end)//this selection statement passes through out the new data through all two of the validation functions making sure that they are correct if they all return true or accept the condition is met
            IF validated IS TRUE//if validated is true the user input is accepted and the condition is satisfied and the new saved
                THEN SET newData AS data//sets the new data attribute to the Initial data entered by the user
                SET listOFPrescrition[2] AS newData//adds the new data to the array
            ENDIF
        ENDIF
        ELSE IF choiceOfPerscription IS dateOfNextDispatch//if the user sets the field choice to this, it will allow for this field to be amended
            THEN SET origninalData AS listOFPrescrition[3]//sets the original data to the one held in the array
            SET Data AS INPUT{Text field}//the data the consultant wants to use is entered
            SET validated AS RUN OBJECITVE 23.B(Data, string) AND RUN//OBJECTIVE 23.D(Data,start,end)//this selection statement passes through out the new data through all two of the validation functions making sure that they are correct if they all return true or accept the condition is met
            IF validated IS TRUE//if validated is true the user input is accepted and the condition is satisfied and the new saved
                THEN SET newData AS data//sets the new data attribute to the Initial data entered by the user
                SET listOFPrescrition[3] AS newData//adds the new data to the array
            ENDIF
        ENDIF
        SET text AS listOFPrescrition[0]+","+listOFPrescrition[1]+","+listOFPrescrition[2]+","+listOFPrescrition[3]//concatenates the prescription setting it as one single sting
        SET encryptedtext AS RUN OBJECTIVE 37(text)//the text is then passed through the encryption objective which returns it to an encrypted text
        WRITE encryptedtext TO filename AT LINE//the text is written to file at the point it originally was
        SET actionToLocation AS "amending prescription"//saves the action to amendments to prescription
        RUN OBJECTIVE 13B(employeeID, actionToLocation, patientID, AdmissionID, OrginalData, NewData)//this passes the action just performed into the objective to add it to their action log
```

```
        CLOSE filename//the file is closed to prevent is corruption
END program
```

33. Add Admission information

```
START program
        RETRIEVE patient FROM allPatients[]//the desired patient is retrieved from the array
        containing all patients
        SET allAdmisisons[]//initialises the array containing all the admissions
        SET filename AS patient.patientID + "_" + Admisison.txt//concatenates the information
        creating the text filename
        SET desiredAdmission AS INPUT{Button}//the user enters the desired admission
        OPEN filename//opens the file to allow it to be amended
        DO
                THEN READ LINE//the line is read
                SET decryptedtext AS RUN OBJECTIVE 38(line)//the line is decrypted
                SET allAdmisisons[lineCounter] AS decryptedtext//the line is then added to the array
                SET lineCounter AS lineCounter++//the index counter increments
                IF LINE IS desiredAdmission//selection determining if the admission the user enters is in
                the array
                        THEN SET index AS lineCounter//the line containing the desired admission is
                        saved to the variable index
                ENDIF
        UNTIL LINE IS NULL//this loop keeps iterating until the file contains no more text
        THEN SET admission AS alladmissions[index]//the admission is then set to the correct one
        containing all the relevant information
        SET arrayLength AS LENGHTH.alladmissions[]//the length of the array is found
        FOR counter IN RANGE arrayLength//declaration of a for loop running for the length of the
        array
                THEN IF alladmissions[counter] IS NULL//selection of the admission array seeing if its
                contents are empty, this indicating that the field in the file is empty
                        THEN SET alladmissions[counter] AS RUN OBJECTIVE 23A-F(INPUT){Text
                        field}//At this point the new data is filled in and correctly validated
                        corresponding to the field
                        SET NewData AS alladmissions[counter]//the new data is saved to the
                        attribute for the transaction log
                        SET originalData AS NULL//As no original data existed the attribute for the
                        transaction log is set to null
                        SET actionToLocation AS "Adding to admission"//the attribute for the attribute
                        is saved to contain  the correct information
                ENDIF
        NEXT counter//the next counter is incremented
        FOR counter IN RANGE 10//declares a counter that runs for an iteration of 10 times
                SET text AS alladmissions[counter]+ ","//the text for all the array is concatenated to
                add new information to the string
        NEXT counter//the counter increments
        SET encryptedtext AS RUN OBJECTIVE 37(text)//the concatenated string is then encrypted using the objective
        WRITE encryptedText TO filename AT LINE//this newly encrypted text is then wrote to file
        CLOSE filename//the file is closed to prevent corruption
        RUN OBJECTIVE 13B(employeeID, actionToLocation, patientID, AdmissionID, OrginalData, NewData)//this passes the action just performed into the objective to add it to their action log
END program
```

34. Edit Admission information

```
START program
        RETRIEVE patient FROM allPatients[]//the desired patient is retrieved from the array containing all patients
        SET allAdmisisons[]//initialises the array containing all the admissions
        SET filename AS patient.patientID + "_" + Admision.txt//concatenates the information creating the text filename
        SET desiredAdmission AS INPUT{Text field}//the user enters the desired admission
        OPEN filename//opens the file to allow it to be amended
        DO
                THEN READ LINE//the line is read
                SET decryptedtext AS RUN OBJECTIVE 38(line)//the line is decrypted
                SET allAdmisisons[lineCounter] AS decryptedtext//the line is then added to the array
                SET lineCounter AS lineCounter++//the index counter increments
                IF LINE IS desiredAdmission//selection determining if the admission the user enters is in the array
                        THEN SET index AS lineCounter//the line containing the desired admission is saved to the variable index
                ENDIF
        UNTIL LINE IS NULL//this loop keeps iterating until the file contains no more text
        THEN SET admission AS alladmissions[index]//the admission is then set to the correct one containing all the relevant information
        SET field AS INPUT//the field wanting to be edited is set as input
        SET unvalidatedData AS INPUT{Text field}//the new data to be added is set as input
        SET validated AS RUN OBJECTIVE 23(unvalidatedData)//The Boolean attribute is set to true or false if correctly or incorrectly validates respectively
        IF validated IS TRUE//if the attribute is accepted the selection condition is satisfied
                THEN SET oldData AS admission.feild//before updating the new field the old data is saved for the action log
                SET newData AS unvalidatedData//Once the new data has been validated it has been saved to the transaction log attribute
                SET allAdmissions[index] AS newData//the new data now overwrites the old data allowing for an update to occur
                SET  actionToLocation AS "Adding to Admision"//the action to location attribute is updated to be used in the action log
        ENDIF
        FOR counter IN RANGE 10//a for counter runs for the number of attributes on the line normally
                SET text AS alladmissions[counter]+ ","//the new text to be added is concatenated to
                be written to the line
        NEXT counter//the next attribute is then used by the index pointer
        SET encryptedtext AS RUN OBJECTIVE 37(text)//the line is then encrypted using the objective
        and using the text as a parameter
        WRITE encryptedText TO filename AT LINE//the new text now overwrites the information
        previously found at the beginning
        CLOSE filename//the file is closed to prevent data corruption
        RUN OBJECTIVE 13B(employeeID, actionToLocation, patientID, AdmissionID, OrginalData,
        NewData)//this passes the action just performed into the objective to add it to their action log
END program
```

35. Add notes

```
START program
    SET docType AS INPUT//the user decides what document they want to use
    SET filename AS "patient.patientID + "_" + admission.AdmissionID +
    "_Documentation.txt"//information regarding the patient is retrieved and the respective
    filename is created
    OPEN filename//the file is opened to allow new documents to be added
    SET oldData AS NULL//as the document is new no old data is available and so field is set as
    null
    THEN SET date AS CURRENT DATE//system sets start date for when the document was written
    SET time AS INPUT{time picker}//user sets time for when the document was written
    SET documentNUM AS INPUT{Text field}//a unique document id is generated
    SET consultantID AS consutltant.consultantID//the consultants id is retrieved from the user's
    attribute
    SET patientID AS patient.patientID//the patient's patient id is also retrieved from the patient's
    attribute
    IF doctype IS prescription//if the user wants this document then this will run
        SET dataMedication AS INPUT{date picker}//the intended input is entered by the
        consultant
        SET validated AS RUN OBJECTIVE 23A(dataMedication )//it is then validated too
        make sure it is acceptable
        IF validated IS TRUE//If it is allowed to be used then this is ran
            THEN SET medication AS dataMedication//the new field in the document is
            now ran
        ENDIF
        SET dataDosage AS INPUT{Text field}//the intended input is entered by the consultant
        SET validated AS RUN OBJECITVE 23.A(Data) AND RUN OBJECTIVE
        23.D(Data,start,end) AND RUN OBJJECTIVE// 23.E(dataDosage
        listOFDosages[])//this selection statement passes through out the new data through all
        three of the validation functions making sure that they are correct if they all return true
        or accept the condition is met
        IF validation IS TRUE//If it is allowed to be used then this is ran
            THEN SET dosage AS dataDosage//once validated the actual field is
            updated to contain the user's input
        ENDIF
        SET dataIntaketime AS INPUT{time picker}//the intended input is entered by the
        consultant
        SET validated AS RUN OBJJECTIVE 23.E(Data listOFDates[]) AND RUN OBJECTIVE
        23.D(dataIntaketime ,start,end)//this selection statement passes through out the new
        data through all two of the validation functions making sure that they are correct if
        they all return true or accept the condition is met
        IF validated IS TRUE//If it is allowed to be used then this is ran
            THEN SET Intaketime AS dataIntaketime//once validated the actual field is
            updated to contain the user's input
        ENDIF
        SET datadateOfNextDispatch AS INPUT{date picker}//the intended input is entered
        by the consultant
        SET validated AS RUN OBJECITVE 23.B(Data, string) AND RUN OBJECTIVE
        23.D(Data,start,end)//the data is entered and checked to see if acceptable
        IF validated IS TRUE//If it is allowed to be used then this is ran
            THEN SET dateOfNextDispatch AS datadateOfNextDispatch //once
            validated the actual field is updated to contain the user's input
        ENDIF
        SET newdata AS doctype+","+date+","+ time +","+documentNum+","+ consultantID
        +","+PatientID +","+medicationdosage//saves the new data field for the action log
        as a concatenation of all the fields
        SET ActionToLocation AS adding to prescription//updates the action log field to
        indicate what action has occurred
    ENDIF
    IF doctype IS Consultant Notes//if the user wants this document then this will run
        THEN SET notes AS INPUT{Text field}//the intended input is entered by the consultant
        SET newdata AS doctype+","+date+","+ time +","+documentNum+","+ consultantID
        +","+ notes//saves the new data field for the action log as a concatenation of all the
        fields
        SET ActionToLocation AS adding to Consultant Notes//updates the action log field to
        indicate what action has occurred
    ENDIF
    ELSE IF doctype IS Test Results//if the user wants this document then this will run
        THEN SET testResults AS INPUT{Text field}//the intended input is entered by the
        consultant
        SET newdata AS doctype+","+date+","+ time +","+documentNum+","+ consultantID
        +","+ testResults//saves the new data field for the action log as a concatenation of all
        the fields
        SET ActionToLocation AS adding to Test Results//updates the action log field to
        indicate what action has occurred
    ENDIF
    RUN OBJECTIVE 13B(employeeID, actionToLocation, patientID, AdmissionID, OrginalData,
    NewData)//this passes the action just performed into the objective to add it to their action log
    SET encryptedData AS RUN OBJECTIVE 37(newData)//encrypts the data using the objective
    WRITE encryptedData TO filename//writes the encrypted data to file, as it is the latest entry it
    is automatically in order
END program
```

36. Encrypting data before being written to file

```
START program
    RETRIEVE dataToEncrypt//the value intending to be encrypted is passed through
    SET data//initialising the encrypted value
    SET length AS dataToEncrypt.length//the length of the attribute is found
    FOR index FROM 0 TO length //declaration of a for loop that will cycle through every
    character in the string
        THEN SET data AS data + dataToEncrypt{index-5}// the new encrypted value is
        formed by adjusting index locations of the value
    NEXT index//the next index is then used
    RETURN data//the newly encrypted value is returned
END program
```

37. Decrypting that has been read from file

```
START program
    RETRIEVE dataToDecrypt//the value intending to be decrypted is passed through
    SET data//initialising the decrypted value
    SET length AS dataToDecrypt.length//the length of the attribute is found
    FOR index FROM 0 TO length //declaration of a for loop that will cycle through every
    character in the string
        THEN SET data AS data + dataToDecrypt{index+5}// the new decrypted value is
        found returning the original indexes to the intended locations
    NEXT index//the next index is then used
    RETURN data//the newly decrypted value is returned
END program
```

38. Using the Jargon library

```
START program
    RETRIEVE desiredWord//system retrieves the desired word they want to find the definition of
    SET filename AS "Jargon_Library.txt"//system sets the filename to the correct path
    SET found AS FASLE//As the word is not found yet it is set as false
    SET start//initialises the starting point
    SET end//initialises the ending point
    OPEN filename//opens the file correctly
    SET length AS NUMBEROFLINES//finds the number of lines currently in the file
    SET allDefinitions[]//initialises an array to hold all the lines
    FOR 0 TO length//declares a for loop running for every line in the file
        THEN READ LINE//the current line is read
        SET allDefinitions[length] AS LINE//this line is then saved to the respective point in the
        array
    NEXT LINE//the line is then moved on
    DO
        SET middlePosition AS (start + end) DIV 2//Finds the middle position of the array by
        averaging the start and end points
        IF desiredWord IS allDefinitions[middlePosition]//Selection determining if the search
        value is the same as the definition at the current index
            THEN SET found AS TRUE//updates the found attribute
            SET position AS middlePosition//sets the position of the index where it was
            found
            RETRIEVE definition//retrieves all the data regarding the definition
        ENDIF
        ELSE IF desiredWord IS LESS THAN allDefinitions[middlePosition]// Selection
        determining if the search value is less than the definition at the current index
            THEN SET end AS middlePosition – 1//correctly adjusts the new end position
            removing any irrelevant definitions
        ENDIF
        ELSE IF desiredWord IS GREATER THAN allDefinitions[middlePosition]//
        Selection determining if the search value is greater than the definition at the current
        index
            THEN SET startPosition AS middlePosition + 1// correctly adjusts the new start
            position removing any irrelevant definitions
        ENDIF
    UNTIL end > start OR found IS TRUE//iteration condition checking if the array has finished or
    the point found
    CLOSE filename//closes the file to prevent corruption
END program
```

39.Adding to the Jargon library

```
START program
    SET word AS INPUT{Text field}//the consultant will enter the word they want to apply a
    definition for
    SET definition AS RUN OBJECTIVE 23A(INPUT){Text field}//they will now give the definition
    of it
    SET oringialData AS NULL// they  will set the original data for the action log as null as it has
    not been amended
    SET newData AS  word +”,”+ definition//the system will save the new data as a
    concatenation of both the word and definition
    SET filename AS “Jargon_Library.txt”//Correctly declares the name of the file that contains
    the definitions
    SET start//declares the start point of the array
    SET end//declares the end point of the array
    OPEN filename//correctly opens the file
    SET index//index where everything is copied up to
    SET length AS NUMBEROFLINES//the system finds the number of lines in the file
    SET found AS FALSE//sets the attribute as false as it has not been found yet
    SET allDefinitions[]//declares the array holding all the definitions
    FOR 0 TO length//declares a for loop that runs for the entire length of the file
              THEN READ LINE//reads the current line
              SET allDefinitions[length] AS LINE//saves the current line to the array at the respective
              line
    NEXT LINE
    DO
              SET middlePosition AS (start + end) DIV 2//Finds the middle position of the array by
              averaging the start and end points
              ELSE IF word IS LESS THAN allDefinitions[middlePosition]// Selection determining if
              the search value is less than the definitionat the current index
                        THEN SET end AS middlePosition – 1//correctly adjusts the new end position
                        removing any irrelevant definitions
              ENDIF
              ELSE IF desiredWord IS GREATER THAN allDefinitions[middlePosition]//
              Selection determining if the search value is greater than the definition at the current
              index
                        THEN SET startPosition AS middlePosition + 1// correctly adjusts the new start
                        position removing any irrelevant definitions
              ELSE IF word IS GREATER THAN allDefinitions[middlePosition-1] AND LESS THAN
              allDefinitions[middlePosition]//selection determining if the search has found the correct
              index where the definition should be saved to
                        THEN SET index AS middlePosition-1//saves the index r line where the new
                        definition will be saved
                        SET FOUND AS TRUE//escapes the search but meeting the condition for the
                        iteration
              ENDIF
    UNTIL found IS TRUE//iteration condition waiting until  position has been found

    RUN OBJECTIVE 13B(employeeID, localDate, localTime, actionToLocation, patientID,
    AdmissionID, OrginalData, NewData)//this passes the action just performed into the objective
    to add it to their action log
    CREATE TEMP FILE//a temporary file is created to copy over the contents of the array
    FOR counter 0 to index//copies over all the definitions up to the location before the new
    definition
              WRITE allDefinitions[counter]//writes the definition to the line
    NEXT counter//iterates the index moving onto the next definition
    WRITE newData//the system will now write the new desired data to the file
    FOR counter index+2 to length//declares a for loop that will iterate from the position after
    the new definition to the last definition
              WRITE allDefinitions[counter]//writes the definition to file
    NEXT counter//iterates to the next definition
    SET filename AS TEMPFILE//overwrites the actual file with the file containing the new definition
    CLOSE TEMP FILE//closes the file to prevent corruption
    CLOSE filename//corruption
END program
```

40.Searching through booking

```
START program
    SET allBookings[]//initialises the array containing all the respective booking information
    SET filename AS consultant.consultanID+"_Bookings.txt"//declares the file name by using the
    correct attributes of the user
        OPEN filename//opens the file
        DO
            READ LINE//reads the file at the line
            SET allBookings[counter] AS LINE//saves the contents of the line to the index of
            the array
            SET COUNTER ++//increments counter
        UNITL LINE IS NULL//iteration condition seeing if the line read is null if so, loop ends
    RETRIEVE desiredBooking //Returns the booking needing to be searched
    SET lengthOfArray AS length.allBookings[]//sets the length of the array
    SET startPosition//Declares the start position index variable
    SET middlePosition//Declares the middle position index variable
    SET endPosition AS lengthOfArray// Declares the end position index variable
    Initially setting it as the last index of the booking list
    SET found AS FASLE//sets the value to be false as nothing is found as of yet
    SET Position//sets the found position of the booking if found within the array
    DO
        SET middlePosition AS (startPosition + endPosition) DIV 2//Finds the middle position of
        the array by averaging the start and end points
        IF desiredBooking IS allBookings[middlePosition]//Selection determining if the search
        value is the same as the booking at the current index
            THEN SET found AS TRUE//updates the found attribute
            SET position AS middlePosition//sets the position of the index where it was found
            SET index AS position//once the position has been found the index containing
            the position is returned
        ENDIF
        ELSE IF desiredBooking IS LESS THAN allBookings[middlePosition]// Selection
        determining if the search value is less than the booking at the current index
            THEN SET endPosition AS middlePosition – 1//correctly adjusts the new end
            position removing any irrelevant bookings
        ENDIF
        ELSE IF desiredBooking IS GREATER THAN allBookings[middlePosition]//
        Selection determining if the search value is greater than the booking at the current
        index
            THEN SET startPosition AS middlePosition + 1// correctly adjusts the new start
            position removing any irrelevant bookings
        ENDIF
    UNTIL found IS TRUE OR endPosition IS LESS THAN startPosition//iteration condition of
    termination until found or that the start index exceeds the end point indicating that no other
    values exist and the search value is not there
    RETURN index//returns the position to the user
    CLOSE filename//closes the file to prevent corruption
END program
```

41.Searching through demographic

```
START program
    SET allDemographic[]//declares the array containing all the demographic information
    SET filename AS "Patient_Demographic.txt"//creates the filename for the correct file
        OPEN filename//opens the file
        DO
            READ LINE//reads the current line
            SET allDemographic[counter] AS LINE//saves the line to the correct index on
            the array
            SET COUNTER ++//increments counter to allow the line to move on
        UNITL LINE IS NULL//iteration condition until the line contains no information
    RETRIEVE desiredPatient//Returns the patient needing to be searched
    SET lengthOfArray AS length.allDemographic[]//sets the length of the array
    SET startPosition//Declares the start position index variable
    SET middlePosition//Declares the middle position index variable
    SET endPosition AS lengthOfArray// Declares the end position index variable
    Initially setting it as the last index of the patient list
    SET found AS FASLE//sets the value to be false as nothing is found as of yet
    SET Position//sets the found position of the booking if found within the array
    DO
        SET middlePosition AS (startPosition + endPosition) DIV 2//Finds the middle position of
        the array by averaging the start and end points
        IF desiredPatientIS allDemographic[middlePosition]//Selection determining if the
        search value is the same as the patient at the current index
            THEN SET found AS TRUE//updates the found attribute
            SET position AS middlePosition//sets the position of the index where it was
            found
            SET index AS position//saves the position the demographic was found
        ENDIF
        ELSE IF desiredPatient IS LESS THAN allDemographic[middlePosition]// Selection
        determining if the search value is less than the patient at the current index
            THEN SET endPosition AS middlePosition – 1//correctly adjusts the new end
            position removing any irrelevant patients
        ENDIF
        ELSE IF desiredPatient IS GREATER THAN allDemographic[middlePosition]//
        Selection determining if the search value is greater than the patient at the current
        index
            THEN SET startPosition AS middlePosition + 1// correctly adjusts the new start
            position removing any irrelevant patients
        ENDIF
    UNTIL found IS TRUE OR endPosition IS LESS THAN startPosition//iteration condition of
    termination until found or that the start index exceeds the end point indicating that no other
    values exist and the search value is not there
    RETURN index//returns the position the demographic as found
    CLOSE filename//closes the file to prevent corruption
END program
```