

Deliverable 5
CS3500 Software Engineering:

E-Medical Card and Hospital Bed Allocation System

Team 14:

Louis Sullivan, Abbie Delaney, Cian McDonald, Katie Crowdle, Ben Prizeman

Submitted By:

Ben Prizeman 26/11/2021



University College Cork
Ireland

Abstract

As the Covid-19 pandemic hammers global economies it is Ireland's health service that is left in dismay and expected to experience a further decline. Pre-pandemic Ireland was suffering from a lack of inpatient hospital beds; however, this has only worsened as a result of the virus. While overcrowding is undoubtedly the primary issue in existence here, another issue arises because of this; that of the prioritization of patients. To tackle the ever-looming threat of complete exhaustion of hospital beds, the pre-existing triage scoring system* is utilized to give precedence to vulnerable patients. It is imperative to be knowledgeable of the ranking associated within this system to ensure efficient prioritization of the patients. Thus, the objective of this project is to address this demanding issue of hospital bed allocation through the development of an e-medical card integrated with the triage scoring system.

*A triage scoring system is used to 'assist prehospital personnel determine which patients require trauma center care.' (Marcin, J., Pollock, M., 2002)

1.Introduction

As illustrated in the left column of table 1.1 below, the Organization for Economic Co-operation and Development (OECD) reported that in 2019 there was a total number of 14,213 beds in hospitals in Ireland (Statistica, 2021). Figures relating to hospital beds in 2020 are currently unavailable due to the unprecedented impact the Covid-19 pandemic had on the Irish healthcare system.

According to a report first published in 2018, because of a surge in demand for hospital beds, because of numerous variables including population growth, it was identified that there exists an increased 'need for between 4000 and 6300 beds across public and private hospitals' by 2030 (Keegan, C., 2019). However, between a four-year period of 2015 and 2019, there were only 483 additional beds added. If this increase is to continue at the exact same rate, the number of hospital beds created by 2030 will be $\approx 1,008$, far from what is expected to satisfy demand. This demonstrates that there is a disparity in the number of beds that are in hospitals and the number of beds needed.

Year	Hospital Beds	People Waiting on Trolleys
2015	13730	8684
2016	14073	9345
2017	14279	10365
2018	14475	12395
2019	14213	10350

Table 1.1: Amount of Hospital Bed in Use (Statista, 2021) vs Number of People on Trolleys in Ireland (INMO, 2019).

We believe a system needs to be put in place that implements the triage process efficiently. This is in the hopes that it would not only lessen waiting times for patients in hospitals but simultaneously lessen the pressurizing workload allocated to nurses and admin staff alike.

The inadequacy of availability of hospital beds is an unfortunate reality. As seen in table 1.2, in a stark report released by Eurostat, two years ago, in 2019, Ireland ranked 22nd out of the 27 EU countries regarding the number of curative care beds per 100,000 individuals (EuroStat, 2018). Ireland's figure was 269 beds when the EU average was 387 beds. Considering Ireland's GDP per capita in the same year ranked 2nd out of 27 countries (EuroStat, 2019), coming behind Luxemburg, Ireland's number of hospital beds do not correlate with the country's overall economic successes. These figures demonstrate the urgency that is required to deal with this issue.

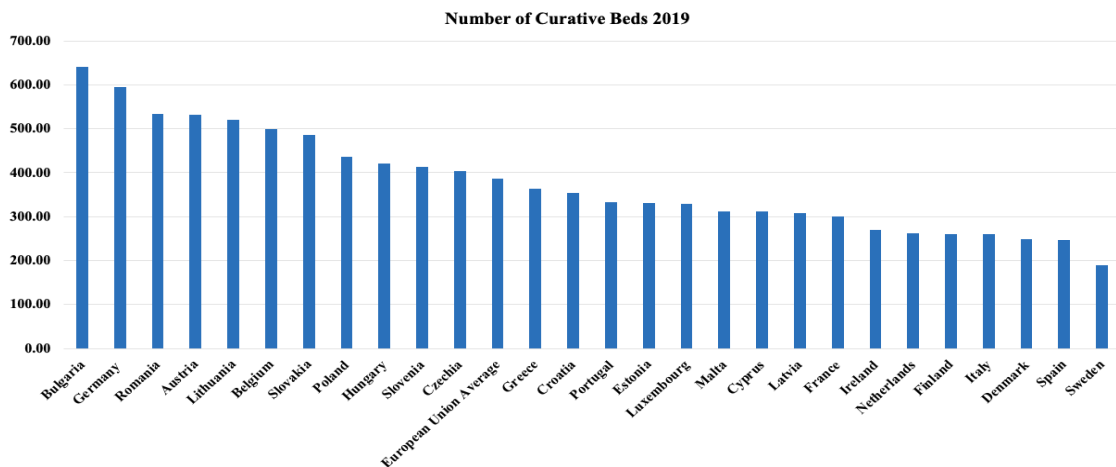


Table 1.2 'Curative Care Beds in Hospital, 2019', EuroStat.

1.1 Background

Hospitals are constantly under pressure with regards to space and time. Hospital beds are limited and so the people that need these beds must have access to them as soon as one becomes available. A system known as a Triage was developed during the Napoleonic Wars to assign different levels of urgency to wounds and illnesses when trying to treat many patients. The people who are ranked at level 1 urgency are seen to first, level 2 is next and so on.

The planning of hospital bed allocation must be strategic, efficient, and ethical to work correctly. Bed capacity is a limited resources in hospitals and if the system is not working correctly, the outcome can be deadly. After speaking to students who study medicine and are on placement in hospitals such as CUH, we discovered that the priority queue of patients is often tracked on physical paper. There are also a multitude of questions to be asked as nurses must try to figure out where each patient is on the triage scale. This seems to us that a lot of crucial time is wasted on tasks that could be completed a lot quicker by dedicated software. We also believe a lot of time is wasted by filling out simple details to retrieve a patient's medical history to make their triage score even more accurate. With almost 3.7 million people in Ireland owning smartphones (Statista, 2021) we believe it's time to create a "E-medical card". A hospital can simply scan a patients assigned QR code and retrieve all the patients personal and medical details. This time save could be crucial to a patient's well-being.

2.Requirements

2.1 Assumptions

It is assumed that:

- There is a mobile signal/WIFI readily available in the hospital to receive a two-factor authentication message.
- There is pre-existing database that contains patient previous medical history.

2.2 Functional Requirements

A functional requirement is one that is necessary for the system to run. Our system's requirements are listed below:

R1. A website that is protected by a login. Once a user logs in with a known email and password stored in a database, their personal QR Code will be displayed.

R2. A QR Code can be scanned by hospital to receive the user's personal details, medical history from a government owned database.

R3. Maintains a database of known symptoms and their associated triage score.

R4. Data entered is used by an algorithm to calculate the person's triage score.

R5. A triage-based priority queue maintained to track the patients waiting for a bed.

2.3 Non-functional Requirements

A non- functional requirement is one that is necessary for operation of the system and not the specific behaviors of the system. Our system's requirements are listed below:

R1: The triage assignment process scales with importance will be longer than two minutes in less severe instances.

R2: The system will be always readily available with 99.99% up time.

R3. Accessible graphical user interface for both the e-medical card and the bed allocation system with clear and intuitive widgets.

3. Use Cases

The diagram below shows the most common use case of the system. The patient logs onto the app if they are verified. Once logged in, their QR code is scanned by the hospital and their medical history is received. We then calculate their triage based upon the data that is received and the current illness entered by the admin. Bed availability is then checked and they may be put in a queue (position based upon their triage score) if there are not any beds available. Once a bed becomes free, the patient at the top of the queue is allocated this bed.

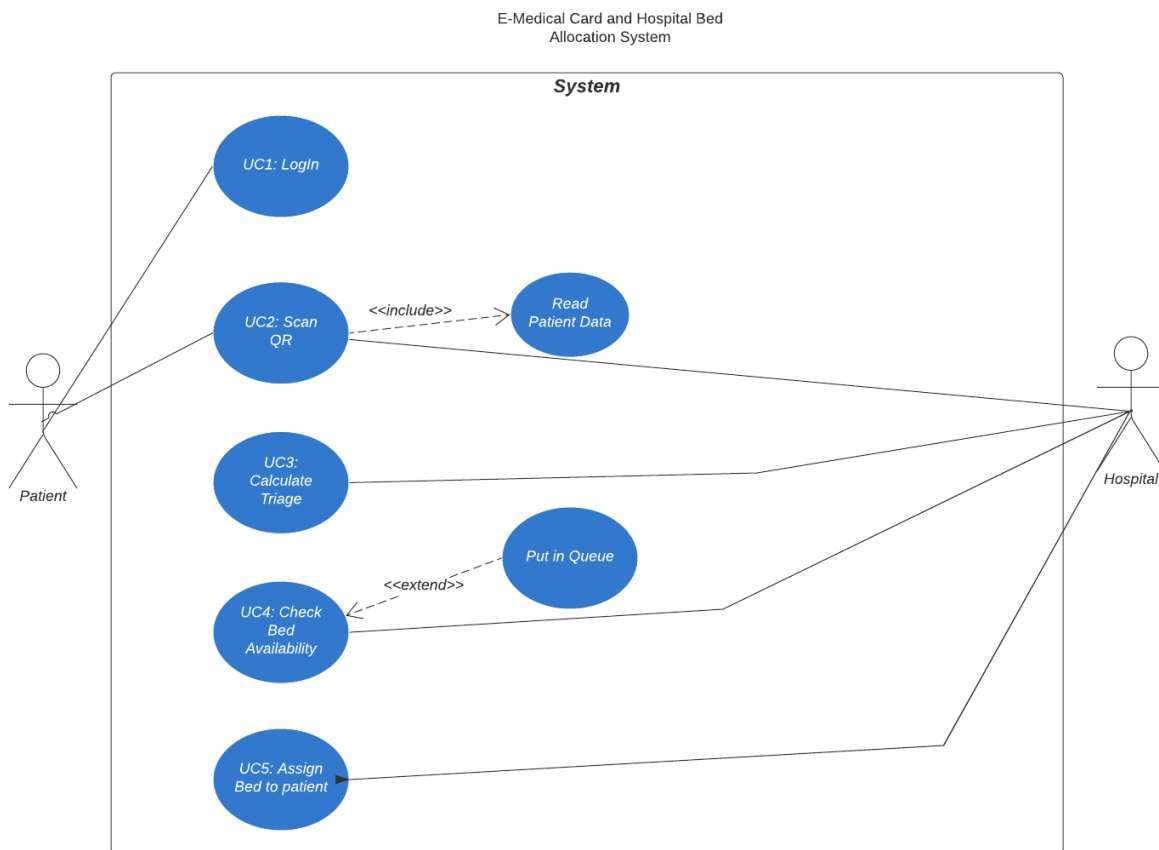


Figure 3.1 Use-case diagram

4. Finite State Machine

The diagram below is a depiction of a finite state machine for the bed allocation system being implemented. A finite state machine is a model for 'any system with a limited number of conditional states of being' (whatif.com, 2019). It is necessary for the machine to transition from one state to another, and so it is usually, as illustrated below, composed of a set of states (the circles) and a set of arrows to each of these circles indicating the corresponding actions. The quality of the system is reliant on each state working efficiently.

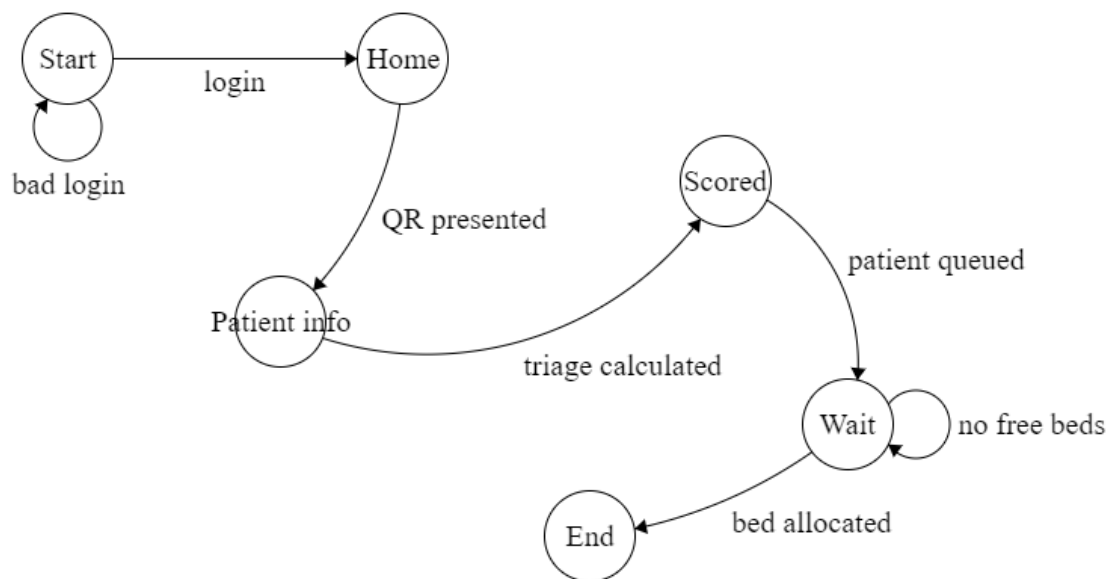


Figure 4.1 Finite State Machine

5. Sequence Diagram

The diagrams below describe how and in what order actors in our system interact in order to carry out the functionality of each scenario. These interactions are modelled around use cases which are given in (3. Use Cases) above. Describes the main scenario for each use case and provides alternative scenarios.

5.1 UC1. Login

Patient login to website.

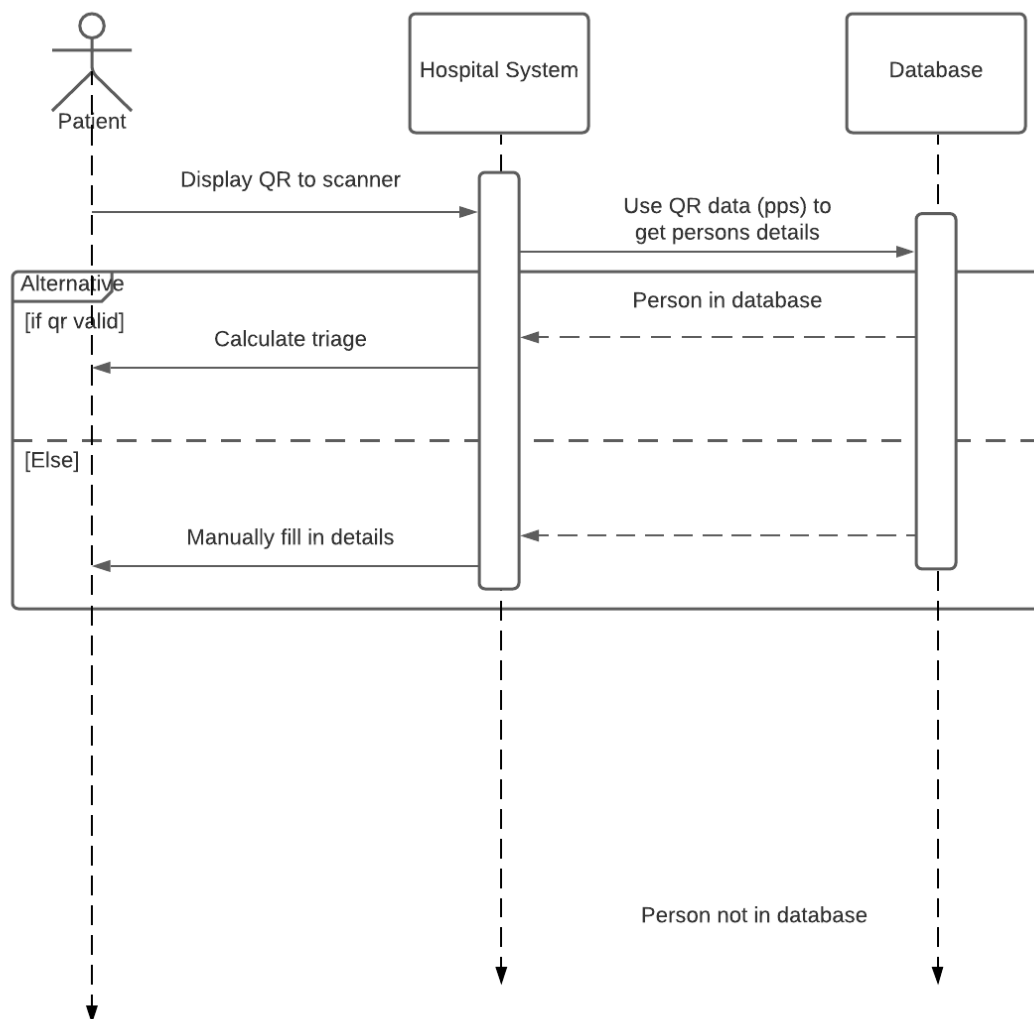


Figure 5.1 Login Use Case

5.2 UC2. Scan QR

Hospital scan patient's QR code.

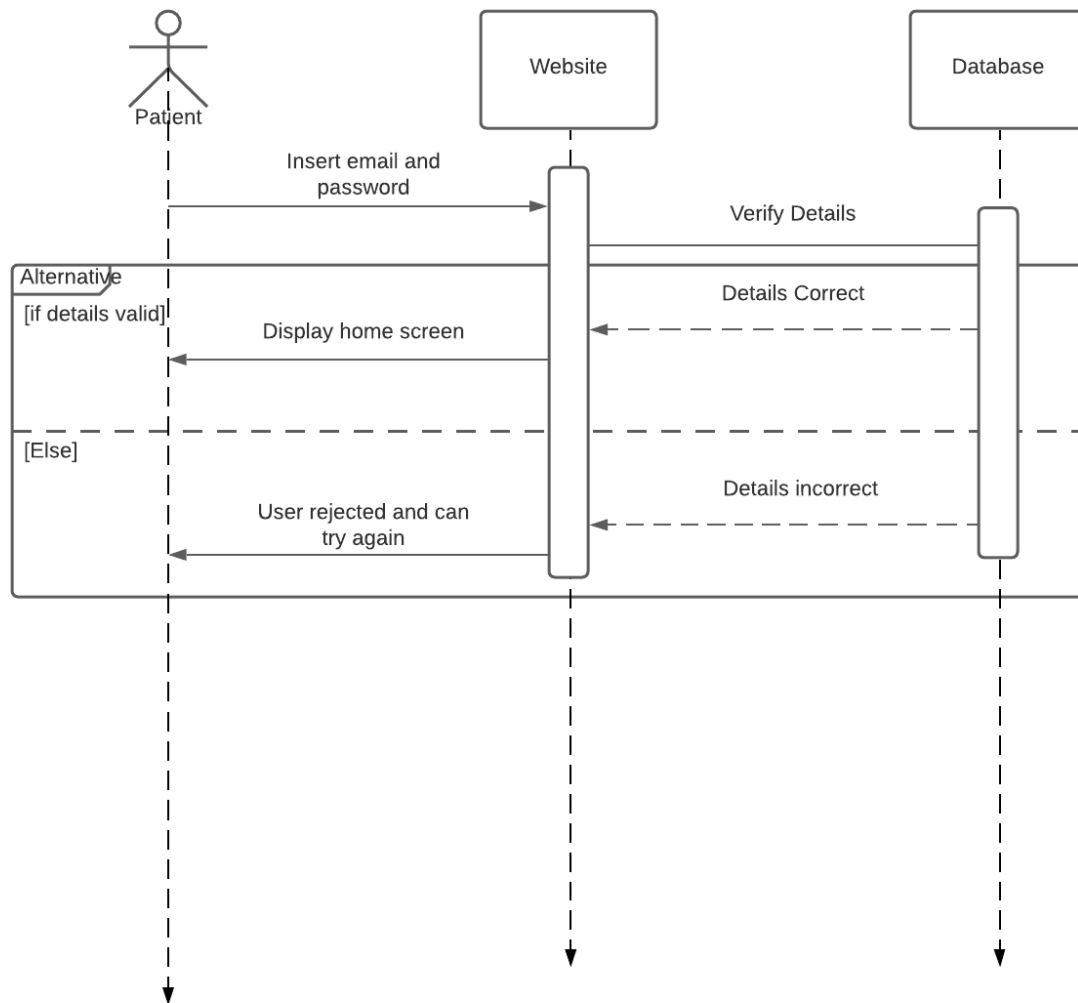


Figure 5.2 Scan QR Use Case

5.3 UC3. Bed Availability

Check for available beds in hospital.

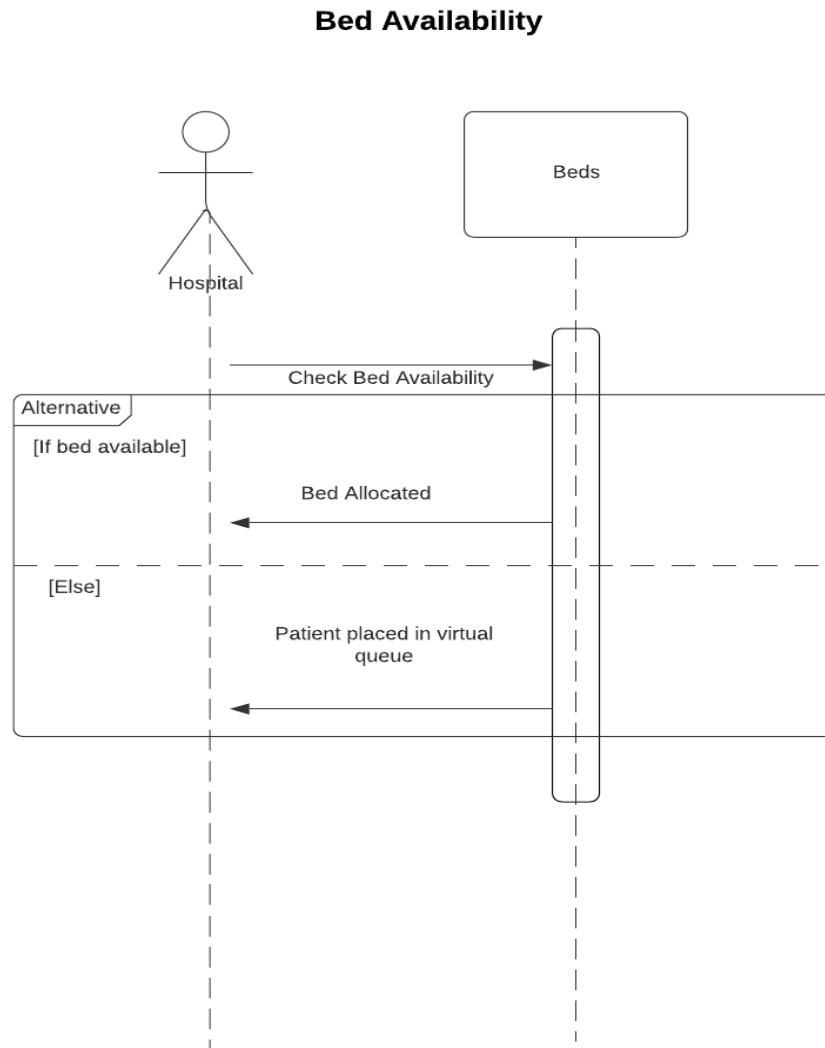


Figure 5.3 Check Bed Availability Use Case

5.4 UC5. Assign Bed to Patient

Assign Bed to patient.

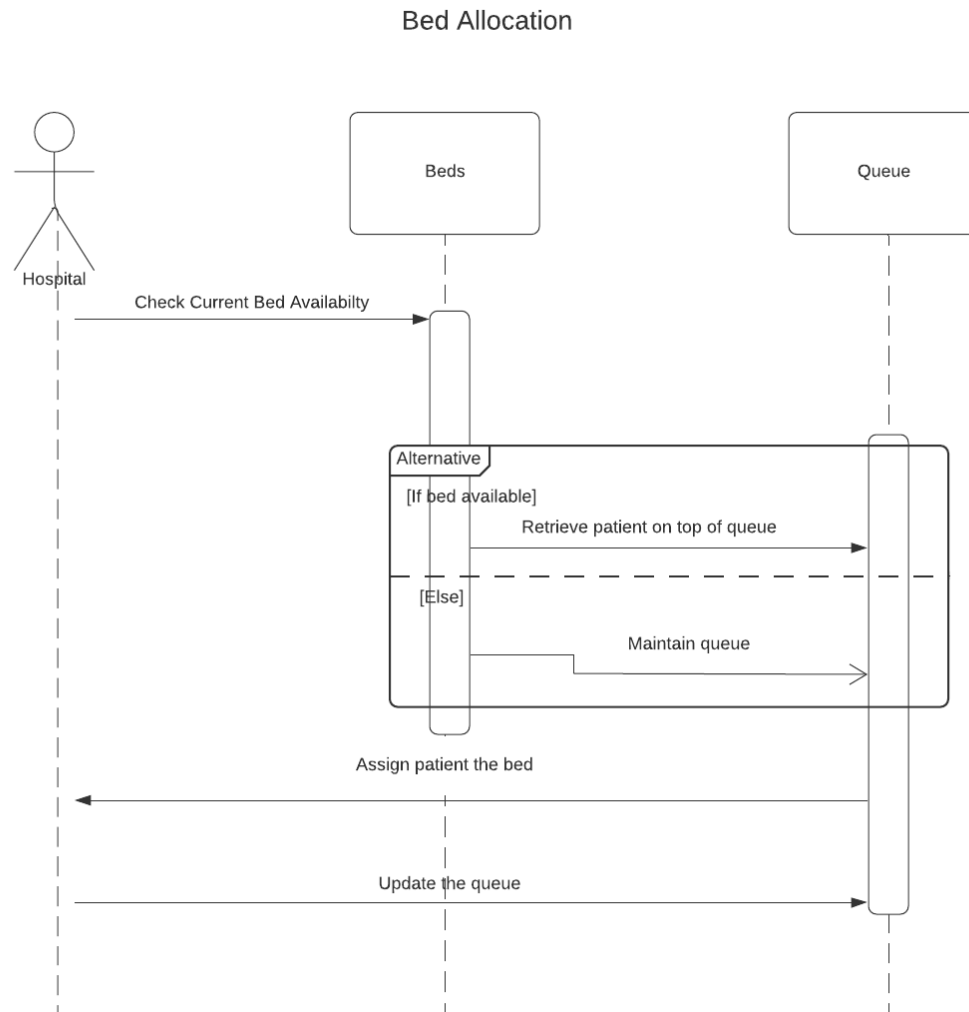


Figure 5.4 Assign Bed to Patient Use Case

6. Data Flow Diagram

The below diagram describes the flow of information through the system. It also provides information on each entity's inputs and outputs. There are two levels to the diagram level 0 providing the most abstracted view of the system. Level 1 a greater level of detail, the specific entities data is drawn from and end points for data.

Level 0:

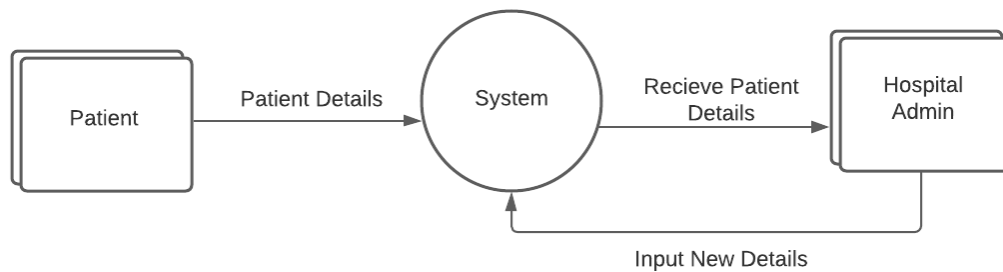


Figure 6.1 Level 0 Data Flow Diagram

Level 1:

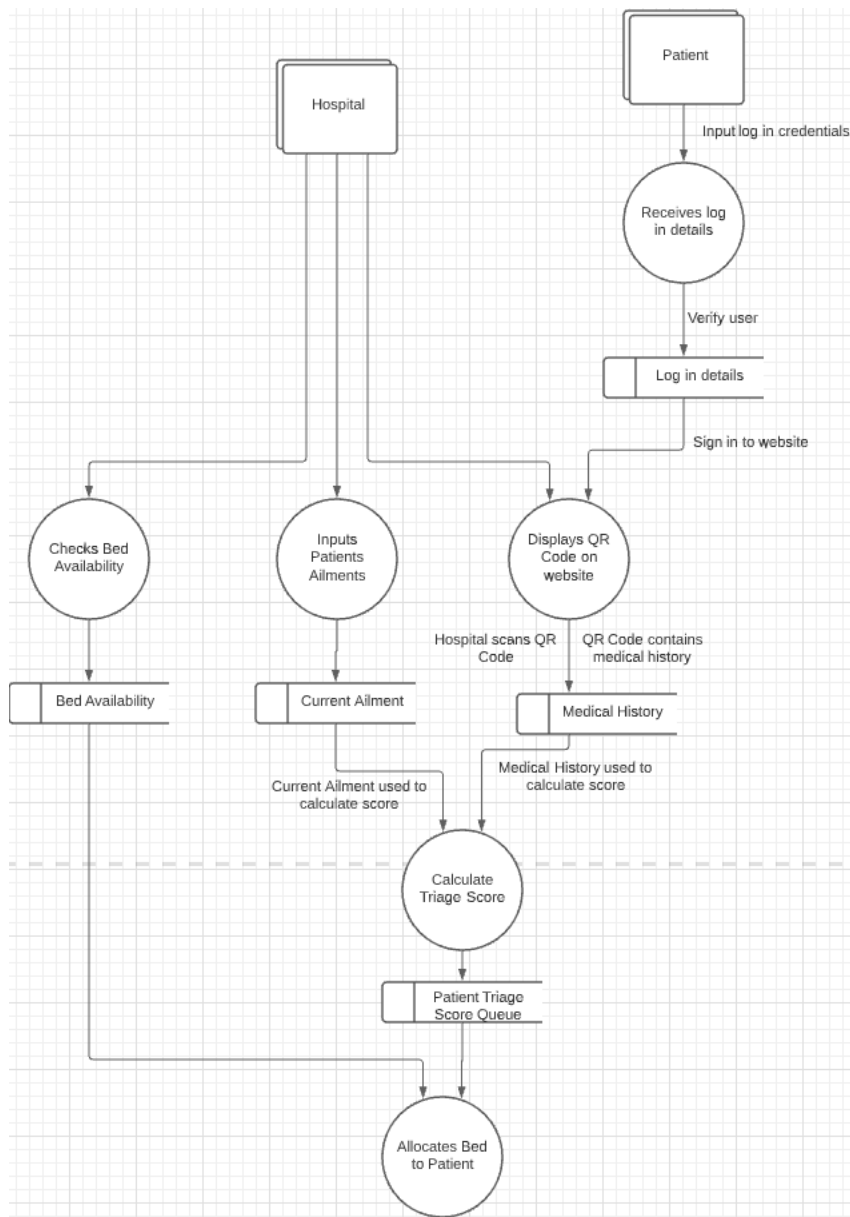


Figure 6.2 Level 1 Data Flow Diagram

7. Entity Relationship Diagram

The diagram below describes how the entities in our system are related and the information each holds. More specifically the structure which can be implemented by databases in the system. Describes the logical structure of the system and its underlying components. The cardinality in the relationships specifies how many instances of one entity relates to another instance of a different entity.

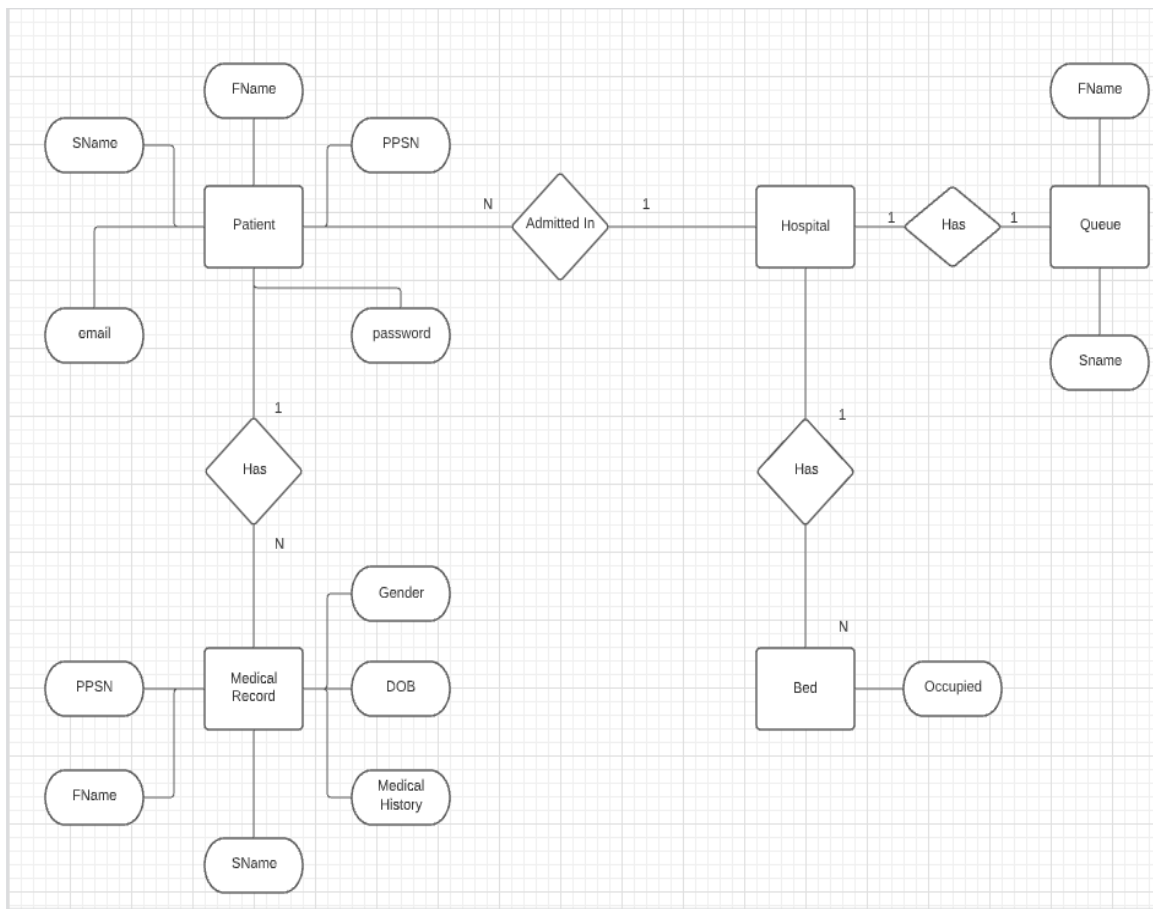


Figure 7 Entity Relationship Diagram

8. Implementation

In summary, a user, using their details, logs in to the website, via two input boxes 'Email Address' and 'Password'. On submitting these details, and in the case that they are correct (an error shows otherwise), a unique QR code will be displayed to the patient. To trigger the accessibility of their personal details, a nurse and/or admin staff of the hospital must select the 'Add Patient' button.

The QR code is then scanned via a webcam. A pre-filled form will appear on the nurse's device, detailing who the patient is and their previous medical history, if any. In the case that the patient does not have a QR code, a nurse has the ability to fill out the form manually, these details will then be added and saved to the existing HSE database. The nurse/admin staff can then select a current ailment based on the ones listed via a drop-down menu, with matching selections appearing based on what is typed. The associated triage score of the selected ailment will then be calculated.

Once the form is submitted, a patient 'object' is created. This consists of the patient's forename and surname and their current designated triage score. This object is then placed within a priority queue based on the triage score assigned to the patient, with the more serious illnesses taking precedence. In the event of human error, via the 'Edit Queue' button, the nurse can select one or more patients and delete them from the queue. To finish the process, when a bed becomes free in the hospital the nurse will click the 'Assign Bed to Patient' button. The patient 'object' existing on the top of the queue, based on their triage score, is appointed the bed. The nurse is notified of their name.

Unsurprisingly, the project was subject to several changes throughout its development, wavering from the initial ideas in an attempt to increase efficiency.

Originally, it was intended that the login system would be implemented through utilising the CGI method; this encompasses a script that will be invoked by a server through HTML inputs. However, as the project progressed it was recognised that employing a flask framework was a more appropriate technique to adopt. This is due to its scalability and accessibility to all involved on the team, offering pre-existing libraries and tools for use. Storing the ailments in a dictionary was also altered. It was realised that, due to its size and complexity, those details would be more suitably maintained in a database, because it allows for simple modification.

To expand the scope of the project, numerous elements were implemented that had not been in the initial outline. Relating to the QR code, at first it was assumed that it would have been decoded by a function. By the end, the use of a webcam to physically scan the QR code was executed. Along with this, the 'Edit Queue' button was introduced to the admin's homepage. The functionality of this is to be able to delete individuals from the queue, when necessary. This was deemed an essential addition because it eliminates potential issues that may arise due to human error on input.

Working Demonstration:

<https://drive.google.com/drive/folders/1E-aizndN-kFPFJpYCI0hHQUXFEKBFGjh?usp=sharing>

Code in GitHub:

<https://github.com/CianMcDonald/ECHBAS.git>

9. White Box Testing

We decided as a group to complete five white boxes. Each member took a method in our code and testing it thoroughly to make sure it was free of any bugs. Each of us created a Control Flow Graph (CFG) and calculated the Cyclomatic Complexity to determine the number of test cases to be completed on each method.

9.1 Key Entered Test

The 'Key entered' method takes a user's input from a text box and checks to see if it's in our list of ailments. If the list is empty, it returns the full list, if the value is not in the list, it returns nothing and if it's in the list it returns what was inputted.

```
def key_entered(event):  
1         entered = e6.get()  
2         if entered == " and entered == '<BackSpace>':  
3             totallist = ailmentsearch_list  
         else:  
4             new_val_list = []  
5             for val in ailmentsearch_list:  
6                 if entered.lower() in val.lower():  
7                     new_val_list.append(val)  
8             update_list(new_val_list)
```



```

def key_entered(event):
    """
    When a key is entered into the sixth entry box (current ailment)
    this function searches the list for that ailment and shortens the list.
    """

    # get what letters was entered by person
    entered = e6.get()
    # if box is blank and backspace is pressed
    if entered == '' and entered == '<BackSpace>':
        #display entire list
        totallist = ailmentsearch_list
    # else a key has been entered
    else:
        # create a list with new values
        new_val_list = []
        # look for a value in our complete list that matches what was entered in the box
        for val in ailmentsearch_list:
            # if value entered equals value in the list
            if entered.lower() in val.lower():
                #add it to our new list
                new_val_list.append(val)
        # display only values that match the entered value
        update_list(new_val_list)

```

Figure 9.1: Key entered method

Cyclomatic Complexity:

Number of edges (E): 11

Number of nodes (N): 8

$$V(G) = E - N + 2$$

$$V(G) = 11 - 8 + 2$$

V(G) or Branch Coverage = 5

Three Test Cases:

- Value entered was nothing therefore it should return full list.
- Value entered was in the list so it should just return that value in a list.
- Value entered was not in the list so it should return an empty list.

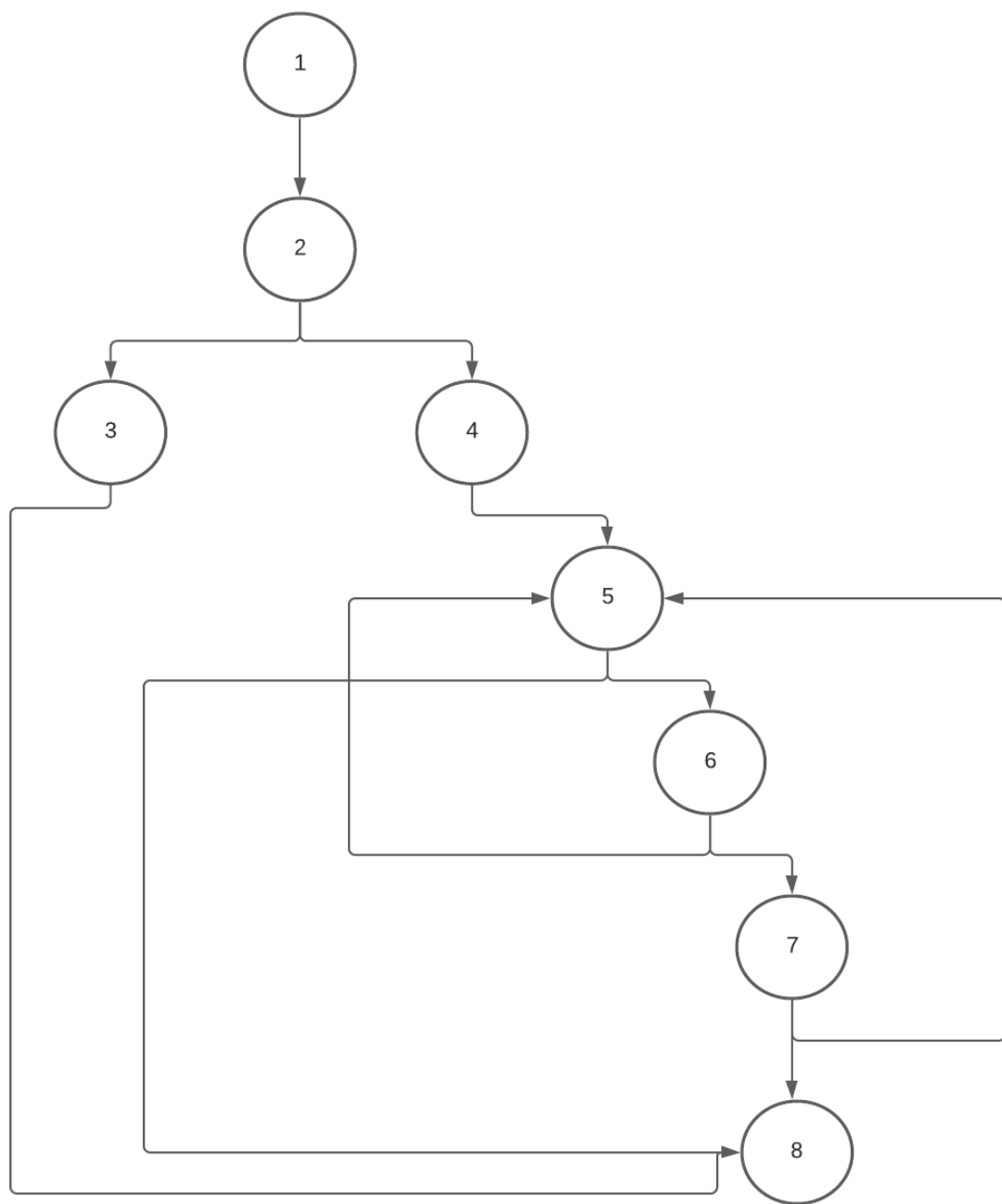


Figure 9.2: CFG for Key entered method

9.2 Calculate Triage Test

The 'Calculate Triage' method takes a user's input from a text box and checks to see if it's a valid triage score. It is valid if it is a number that is greater than or equal to one or less than or equal to five.

```
def calculate_triage(event):
```

```
1     current_injury = current_injury_submit.get()

    patient_triage_query = "SELECT score FROM ailments WHERE name = '"+current_injury+"'"

    cursor.execute(patient_triage_query)

    triage_db_result = cursor.fetchone()

    if str(triage_db_result[0]).isdigit():

2         if int(triage_db_result[0]) <= 5 AND int(triage_db_result[0]) >= 1:

3             triage_score = int(triage_db_result[0])

                e7.delete(0, 'end')
                e7.insert(0, triage_score)
                validate_form(None)

        else:

4             Master.messagebox.showerror.(title="Triage Score Error", message= "The value

                entered in the triage score is incorrent!")
```

```
def calculate_triage():
    """
    Function that gets data from form and calculates the patients triage score
    """
    # get triage from db
    current_injury = current_injury_submit.get()
    patient_triage_query = "SELECT score FROM ailments WHERE name='"+ current_injury +"'"
    cursor.execute(patient_triage_query)
    triage_db_result = cursor.fetchone()
    # if the triage is a number
    if str(triage_db_result[0]).isdigit():
        # add triage to entry box
        triage_score = int(triage_db_result[0])
        e7.delete(0, 'end')
        e7.insert(0, triage_score)
        validate_form(None)
    else:
        # triage is not a number so display error
        master.messagebox.showerror(title="Triage Score Error", message="The value entered into Triage Score is incorrect!")
```

Figure 9.3: Calculate triage method

Cyclomatic Complexity:

Number of Edges (E): 4

Number of Nodes(N): 4

$$V(G) = E - N + 2$$

$$V(G) = 4 - 4 + 2$$

$$V(G) \text{ or Branch Coverage} = 2$$

Three Test Cases:

-Value entered is a digit.

-Value entered is not a digit.

-Value entered is a null.

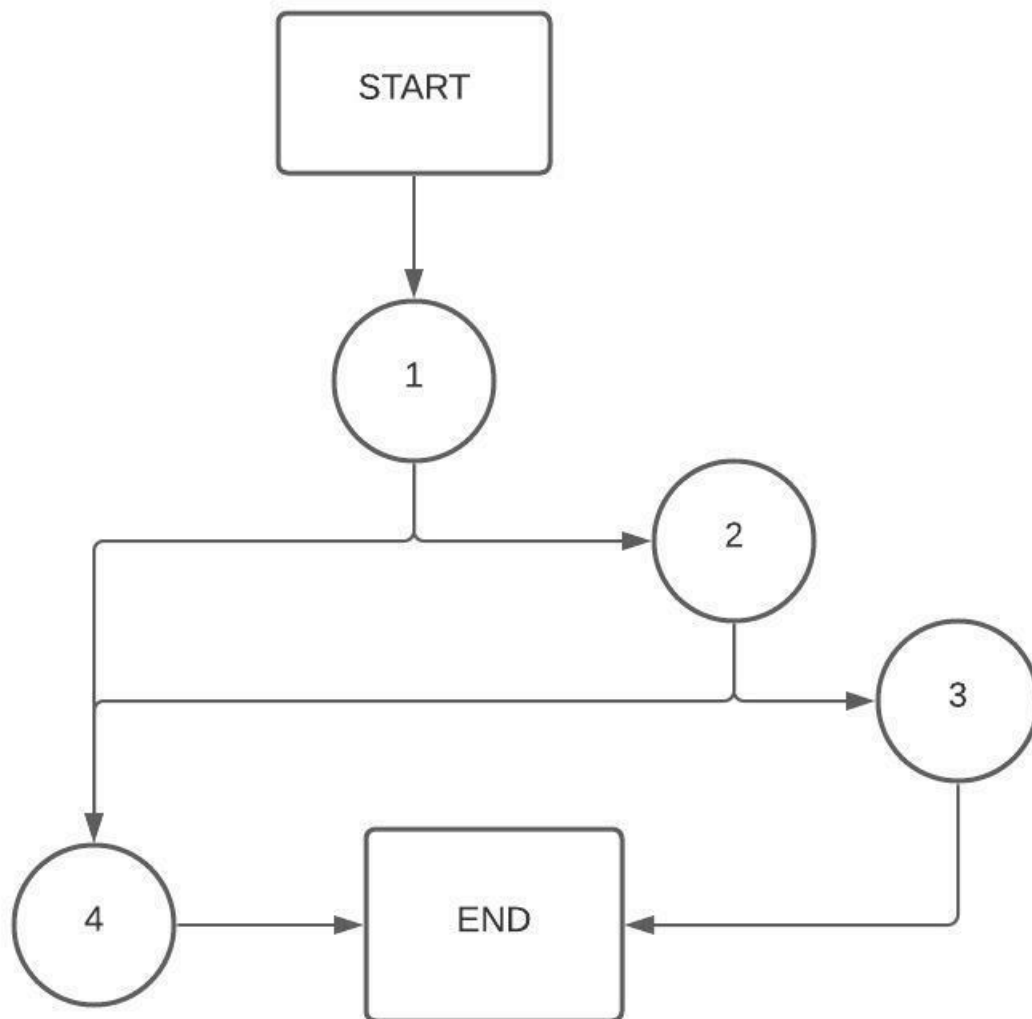


Figure 9.4: CFG for Calculate Triage method

9.3 Calculate Triage Test

The 'index' function is a login form. It takes in a user's email and password, compares it with all the emails and passwords in the database. If it finds a match it will log the user in. Otherwise an error message is printed.

```
def index():  
    page = 'index.html'  
  
1    if request.method == 'POST':  
        connection = sqlite3.connect("user_data.db")  
        cursor = connection.cursor()  
        email = request.form['email']  
        password = request.form['password']  
        verify_query = "SELECT email, password, ppsno, fname, sname FROM users WHERE email='"+  
email +"'" AND password='"+ password +"'"  
        cursor.execute(verify_query)  
        result = cursor.fetchall()  
  
2    if len(result) == 0:  
        error = "Sorry! Your email or password is incorrect."  
  
3    page = 'index.html'  
else:  
    for row in result:  
        ppsno = row[2]  
        fname = row[3]  
        sname = row[4]  
        QRgen(ppsno, fname, sname)  
        page = 'home.html'  
  
4    user_image = 'static/qr_photos/qrdata.png'  
  
    return render_template(page, user_image=user_image, error=error)
```

```

def index():
    """
    Function that runs checks the user logging is correct and displays home page
    with qr code.
    """
    user_image=None
    error = None
    page = 'index.html'
    # use post to send user's data
    if request.method == 'POST':
        #Connect to db
        connection = sqlite3.connect("user_data.db")
        cursor = connection.cursor()
        # get data entered by user
        email = request.form['email']
        password = request.form['password']

        # for admin to check in terminal
        #print(email, password)

        # query used to verify that the email and password is in our database
        verify_query = "SELECT email, password, ppsno, fname, sname FROM users WHERE email='"+ email +"' AND password='"+ password +'"
        cursor.execute(verify_query)

        # if the result is zero the password/email is incorrect (not in the db)
        result = cursor.fetchall()
        if len(result) == 0:
            # retry login
            error = "Sorry! Your email or password is incorrect."
            page = 'index.html'
        else:
            # get all data for qr
            for row in result:
                print(result)
                ppsno = row[2]
                fname = row[3]
                sname = row[4]
            # generate qr code
            QRgen(ppsno, fname, sname)

        # if its right bring them to their homepage
        page = 'home.html'
        # set the image on homepage to qr code
        user_image = 'static/qr_photos/qrdata.png'
    return render_template(page, user_image=user_image, error=error)

```

Figure 9.5: Login method

Cyclomatic Complexity:

Number of edges (E): 6

Number of nodes (N): 4

$$V(G) = E - N + 2$$

$$V(G) = 6 - 4 + 2$$

V(G) or Branch Coverage = 4

Three Test Cases:

- Value entered is correct login details
- Value entered is incorrect login details
- Value entered are null values

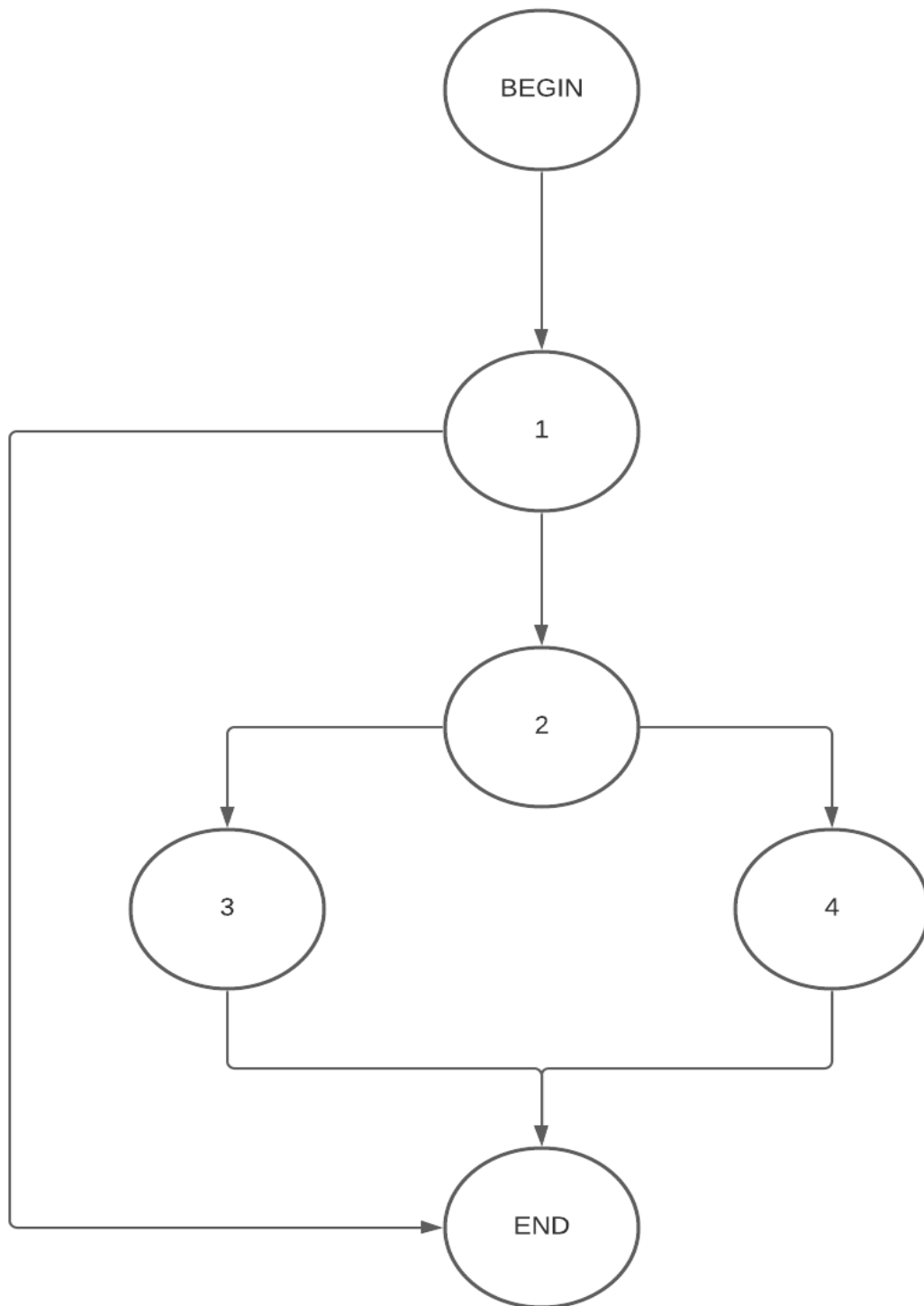
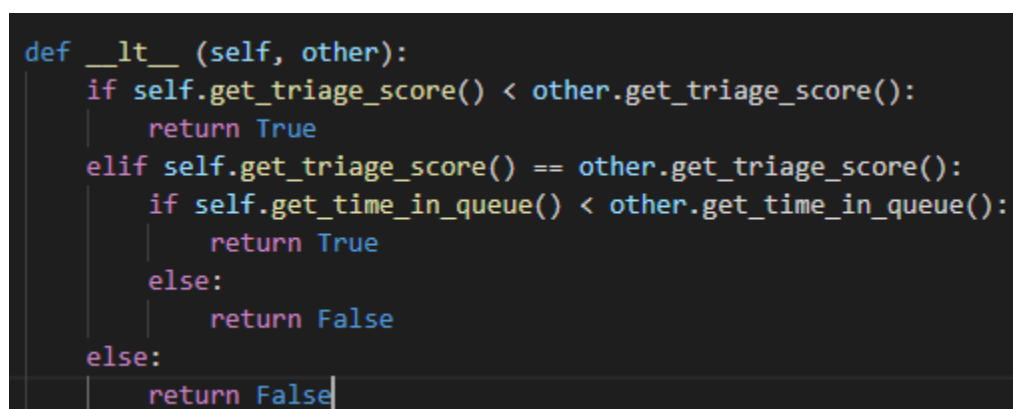


Figure 9.6: CFG for Login method

9.4 Calculate Triage Test

The '`__lt__`' method is used to compare to Patient's triage score in the queue. If one patient has a lower triage score than the other, it will go after it. When two Patient's have the same triage the person who has been in the queue longer will be given higher priority.

```
def __lt__(self, other):  
1         if self.get_triage_score() < other.get_triage_score():  
2             return True  
3         elif self.get_triage_score() == other.get_triage_score():  
4             if self.get_time_in_queue() < other.get_time_in_queue():  
5                 return True  
6         else:  
7             return False  
8         else:  
9             return False
```



```
def __lt__(self, other):  
    if self.get_triage_score() < other.get_triage_score():  
        return True  
    elif self.get_triage_score() == other.get_triage_score():  
        if self.get_time_in_queue() < other.get_time_in_queue():  
            return True  
        else:  
            return False  
    else:  
        return False
```

Figure 9.7: Less than Queue method.

Cyclomatic Complexity:

Number of edges (E): 6

Number of nodes (N): 7

$$V(G) = E - N + 2$$

$$V(G) = 6 - 7 + 2$$

$$V(G) \text{ or Branch Coverage} = 1$$

One Test Case:

-Test that when triage scores are the same whoever is in the queue longer is given priority.

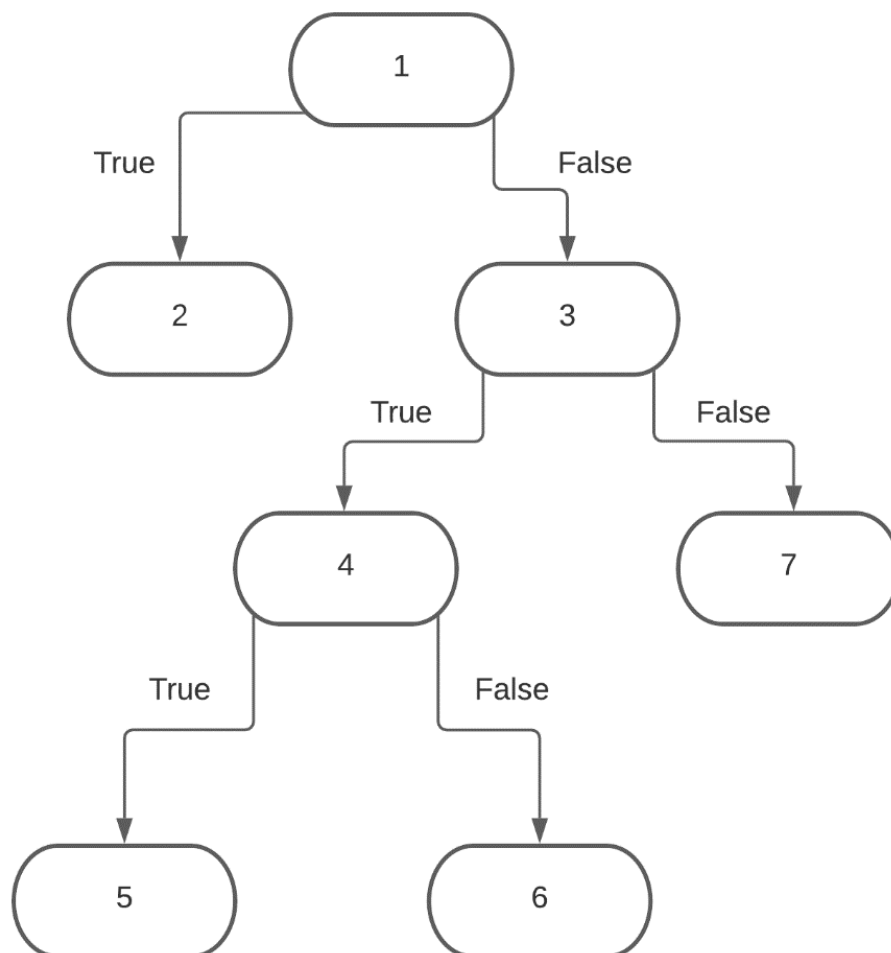


Figure 9.8: CFG for Less than Queue method

9.5 Remove from Queue Test

The 'remove items selected' method is used to remove patients from the queue when selected. It displays an error message if the button is pushed, and a person is not selected.

```
def remove_items_selected(self, listbox):  
1     indexs_to_remove = listbox.curselection()  
  
     if len(indexs_to_remove) == 0:  
2         tk.messagebox.showerror("Error", "No patient selected")  
  
     else:  
3         msg = tk.messagebox.askquestion("Remove from queue", "Are you sure?")  
  
         if msg == 'yes':  
4             for x in indexs_to_remove:  
5                 names.append(self.list_name[x])  
6                 self.queue.remove_from_queue(names)  
  
                 self.destroy()
```

```
def remove_items_selected(self, listbox):  
    indexs_to_remove = listbox.curselection()  
    if len(indexs_to_remove) == 0:  
        tk.messagebox.showerror("Error", "No patient selected")  
    else:  
        msg = tk.messagebox.askquestion("Remove from queue", "Are you sure?")  
        if msg == 'yes':  
            names = [self.list_name[x] for x in indexs_to_remove]  
            self.queue.remove_from_queue(names)  
            self.destroy()
```

Figure 9.9: Remove From Queue method.

Cyclomatic Complexity:

Number of edges (E): 8

Number of nodes (N): 6

$$V(G) = E - N + 2$$

$$V(G) = 8 - 6 + 2$$

V(G) or Branch Coverage = 4

Four Test Case:

- Test when queue empty.
- Test when nothing is selected to be removed.
- Test when a patient is chosen, and we continue with the operation
- Test when a patient is chosen, and we do not continue with the operation

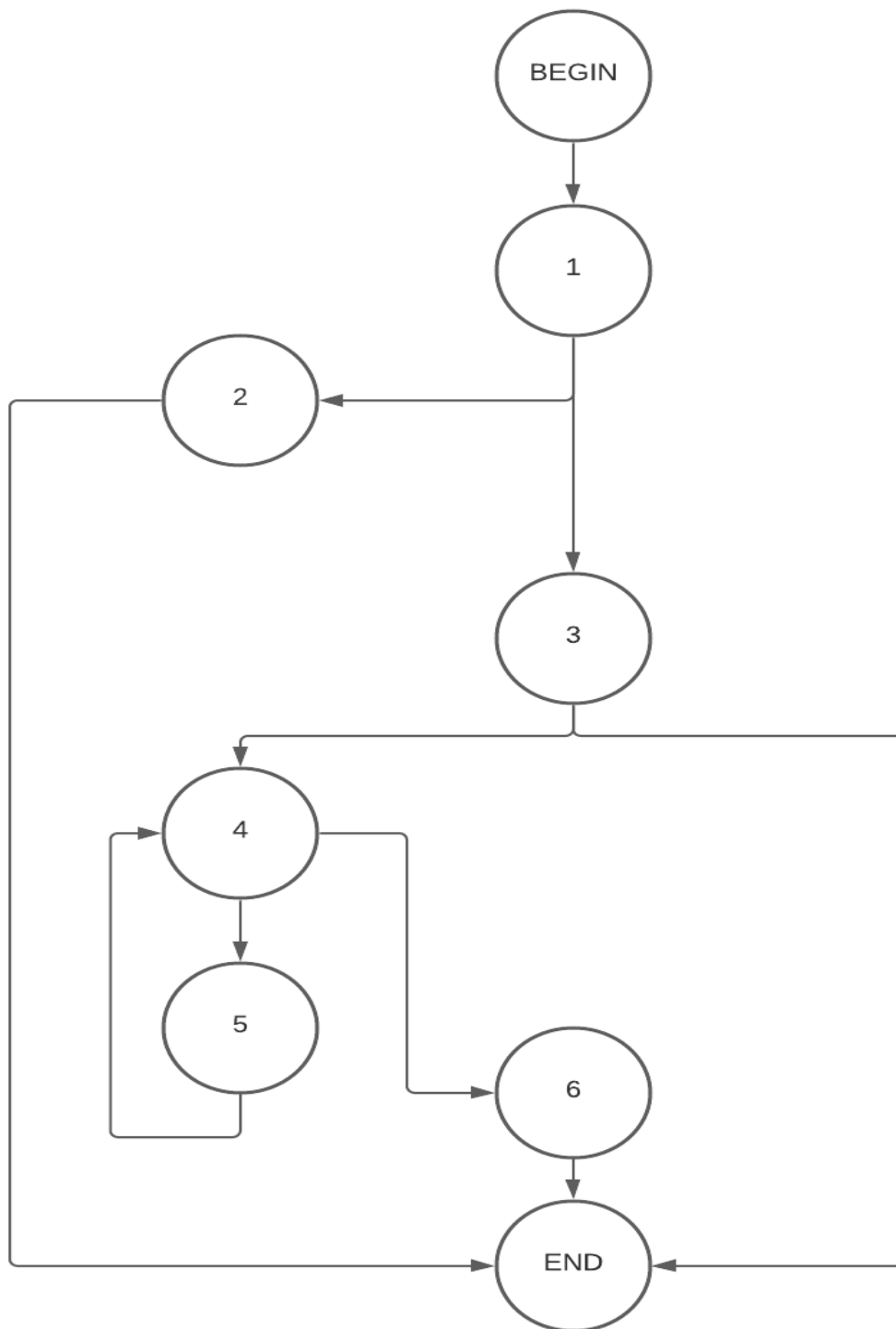


Figure 9.10: CFG for Remove from Queue method.

10. Black Box Testing

We decided to do a total of two black box tests. The point of our project is automating the process of getting a patient into the hospital so there is not a lot of user interaction and so it only seemed appropriate to do a few black box tests. A lot of the input fields are strict on what type of values can be entered so there is not a lot of testing with regards to this.

10.1 Calculate Triage Black Box

As mentioned above the calculate triage method is extremely important as it decides how up in the queue a patient will be place. Therefore we need to extensively test what can be entered for this value as if its not correct a patient's life could be affected.

```
def calculate_triage(entered):  
    """  
    Function gets data from form and gets the patients triage score  
    """  
    # get triage  
    triage_score = entered  
    # if the triage is a number  
    if str(triage_score).isdigit():  
        #if triage is a number less than or equal to 5  
        if int(triage_score) <= 5 and int(triage_score) >= 1:  
            #triage is correct  
            return True  
    else:  
        return False
```

Test:

- A string value is entered into the 'triage score' text field.
- The value returned is True if the value entered is greater than or equal to one or less than or equal to five.
- Otherwise, the value is false.

Input Domain:

Equivalence class	Triage entered	Output return
E1	{3}	True
E2	{{a}, {@}, {!neg}, {!}, {five}}	False
E3	{{-1}, {20}, {23540}, {0.3}, {123}}	False
E4	{null}	False

Equivalence Classes

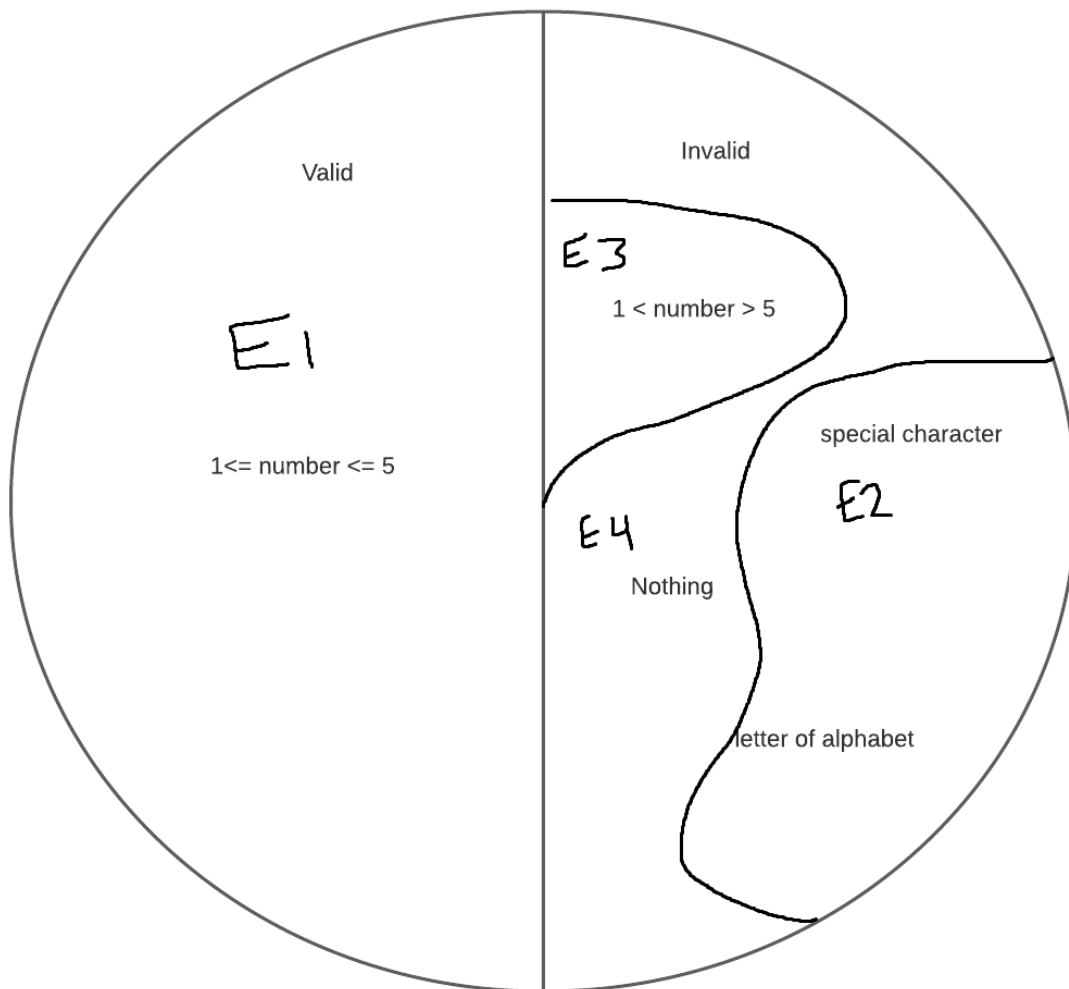


Figure 10.1: Equivalence Class for Calculate Triage

10.2 Key Entered Black Box

The 'Key entered' method is very important too as it can speed up the process of the entire form procedure. If this user filling it in can add an appropriate ailment quickly then the triage score will be auto filled, and the form can be submitted immediately.

```
def key_entered(entered):  
    """  
    Key entered the current ailment entry box this function  
    searches the list for that ailment and shortens the list. """  
    # get what letters was entered by person  
    #entered = e6.get()  
    entered = entered  
    # if box is blank and backspace is pressed  
    if entered == '' and entered == '<BackSpace>':  
        #display entire list  
        totallist = ailmentsearch_list  
        return True  
    # else a key has been entered  
    else:  
        # create a list with new values  
        new_val_list = []  
        # look for a value in our complete list that matches what was  
        entered in the box  
        for val in ailmentsearch_list:  
            # if value entered equals value in the list  
            if entered.lower() in val.lower():  
                #add it to our new list  
                new_val_list.append(val)  
            return True  
        # display only values that match the entered value  
        return False
```


Test:

- A user will start to type a string value
- If the value is in the list of ailments, it will shorten the list box to a string in the list that matches and will then return True.
- If the box is empty or Backspace is pressed the box shows the whole list of ailments and returns True.
- Otherwise, the value entered is invalid and it will return False.

Input Domain:

<u>Equivalence class</u>	<u>Triage entered</u>	<u>Output return</u>
E1	{airway compromise}	True
E2	{Apple}	False
E3	{null}	True

Equivalence Classes



Figure 10.2: Equivalence Class for key entered.

11. Revisions

We believe that revisions and changed from our original design is crucial to the success of a project. It may be that when you begin working on a project that you discover the limits of a project or unnecessary additions to a project. Throughout the task we may also want to update our documentation to be more accurate and to make specific changes necessary for the success of the software. Listed below are changes made through out project's development.

Deliverable 1 changes:

- Fixed wording of use cases to better describe the action taking place.
- Removed "error bubble" in use case diagram.

Deliverable 2 changes:

- Fixed Data flow diagrams as we discovered "Data Black Holes".
- Upgraded resolution of sequence diagrams.

Deliverable 3 changes:

- Removed the requirement "**R5.** *The system must track what beds in the hospital are free.*"
We considered an unnecessary requirement to our system as we are developing a bed allocation system and not a bed management system. The task of managing bed would be significantly harder as you would have to consider the different types of wards, different size hospitals, etc.

12. References

EuroStat, 2019, 'Hospital Beds by Type of Care', Available at:
<https://appsso.eurostat.ec.europa.eu/nui/submitViewTableAction.do>

Irish Nurses and Midwives Organisation (INMO), 2019, '2019 breaks record for most patients on trolleys – and it's not even December' Available at:

<https://inmo.ie/Home/Index/217/13549>

Keegan, C, Brick, A, Walsh, B, Bergin, A, Eighan, J, Wren, M-A. 'How many beds? Capacity implications of hospital care demand projections in the Irish hospital system', 2015-2030. *Int J Health Plann Mgmt*. 2019; 34: e569– e582

Marin, James P. MD, MPH; Pollack, Murray M. MD, FCCM 'Triage scoring systems, severity of illness measures, and mortality prediction models in pediatric trauma', *Critical Care Medicine*: November 2002 - Volume 30 - Issue 11 - p S457-S467

Statistica, 2021, 'Number of Hospital Beds in Ireland from 2000 to 2019', Available at:

<https://www.statista.com/statistics/557287/hospital-beds-in-ireland/>