NEW HORIZON
COLLEGE OF ENGINEERING
Autonomous College, Affiliated to VTU | Approved by AICTE New Delhi & UGC
Accredited by NAAC with 'A' Grade & Accredited by NBA

Golden Jubilee
YEARS OF EXCELLENCE IN EDUCATION

A MINI PROJECT

REPORT

for

Mini Project using Python (20CSE59)

Life360 – Extending Healthcare to your Desktop

Submitted by

RAHUL MUSALIYATH DINESH
USN: 1NH18CS738, Semester-Section: 5-D

In partial fulfillment for the award of the degree of

Bachelor of Engineering

in

COMPUTER SCIENCE AND ENGINEERING

NEW HORIZON COLLEGE OF ENGINEERING

Autonomous College, Affiliated to VTU | Approved by AICTE New Delhi & UGC
Accredited by NAAC with 'A' Grade & Accredited by NBA

50 Golden Jubilee YEARS OF EXCELLENCE IN EDUCATION

Certificate

This is to certify that the mini project work titled

Life360 – Extending

Healthcare to your Desktop

Submitted in partial fulfillment of the degree of

Bachelor of Engineering in

Computer Science and Engineering by

RAHUL MUSALIYATH DINESH

USN: 1NH18CS738

DURING

ODD SEMESTER 2020-2021

for

COURSE CODE: 20CSE59

Signature of Reviewer　　　　　　　　Signature of HOD

SEMESTER END EXAMINATION

Name of the Examiner	*Signature with date*
1. ______________________	______________________
2. ______________________	______________________

ABSTRACT

Access to an affordable and working healthcare system is one of the aspects by which a country can be classified upon. Today, healthcare in India has become of the fastest growing sectors with respect to both employment and revenue.

But as this sector continues to grow, the poor and downtrodden sections of our society are being left behind. They have to make do with the crumbling public healthcare system which by all means would mostly fail due to lack of adequate infrastructure and doctors.

The COVID 19 pandemic has also taught the world about the importance of blood/plasma/organ banks that have played a key role in this battle against the virus. Plasma banks which collect the plasma of recovered patients have played a pivotal role in helping patients affected by the virus, to recover from it.

This mini-project solves the above-mentioned problems through the development of a desktop application. This desktop application would maintain a database of the users and other relevant information and would have a graphical user interface that would be simple and easy to use.

The entire program has been developed in Python and uses the Eclipse IDE for running the python application.

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be impossible without the mention of the people who made it possible, whose constant guidance and encouragement crowned our efforts with success.

I have great pleasure in expressing gratitude to **Dr. Mohan Manghnani**, Chairman of New Horizon Educational Institutions for providing necessary infrastructure and creating good environment.

I take this opportunity to express my profound gratitude to **Dr. Manjunatha,** Principal, New Horizon College of Engineering, for his constant support and encouragement.

I would also like to thank **Dr. B. Rajalakshmi**, Professor and Head, Department of Computer Science and Engineering, for her constant support.

I also express my gratitude to **Ms. Lakshmi S Hanne**, Assistant Professor, Department of Computer Science and Engineering, my project guide, for constantly monitoring the development of the project and setting up precise deadlines. Her valuable suggestions were the motivating factors in completing the work.

RAHUL MUSALIYATH DINESH

USN: 1NH18CS738

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 PROBLEM DEFINITION

In India, access to free healthcare remains a big issue especially in the rural parts of our country where the many of the downtrodden citizens of our country live. Healthcare when available is also quite expensive and this has impacted the poor of our country the most.

Another issue that is common in our country is the lack of timely access to blood, organ and plasma banks which have resulted in many lives being lost. The COVID-19 pandemic has once again shown the importance of maintaining these banks as plasma from recovered patients has proved to be effective in treating those who have been infected.

1.2 OBJECTIVES

Once developed, the application will provide services to

- Register users and maintain a database with their required information
- Register Blood Banks/Hospitals to record blood/organs available with them
- Help those users who require immediate access to blood, plasma and vital organs
- Allow users to volunteer for any emergency blood donation requests.
- Find out the nearest donor when a user has an emergency request (within a city)
- Provide free lung cancer and pneumonia detection when given a picture using prediction models devised from machine learning algorithms.

1.3 METHODOLOGY TO BE FOLLOWED

The mini-project is completely based on the high-level language, Python and the DBMS language, SQL and uses GUI programming to provide a simple and easy to understand platform for the users.

1.4 EXPECTED OUTCOMES

The mini-project aims to provide a simple and secure platform to register users and maintain a database of their information. It also would be able to register hospitals and blood banks and maintain a database with relevant data. It would be able to provide users who require immediate access to blood, plasma and vital organs. While registering, users would also have the option of signing up for blood donation. It would also help users who are patients to connect with the nearest donors/hospital/organ or blood banks. Additionally, it would also provide free lung cancer and pneumonia detection when the user uploads a CT Scan or X-ray scan image by using machine learning algorithms.

1.5 HARDWARE AND SOFTWARE REQUIREMENTS

• Processor	: Intel 386 or higher
• RAM	: 4 MB or higher
• Hard Disk	: 25 MB or higher
• Input device	: Standard Keyboard and Mouse
• Output device	: VGA and High-Resolution Monitor or higher
• Operating system	: Microsoft DOS, Microsoft Windows 3.1 or later
• Python version	: 3.8
• IDE	: Eclipse

CHAPTER 2

FUNDAMENTALS OF PYTHON

2.1 INTRODUCTION TO PYTHON

Python is a commonly and extensively used general-purpose, high-level programming language. Guido van Rossum in 1991 was the founder of Python and was later developed by Python Software Foundation. It was primarily designed to emphasize on code readability, and its syntax allows programmers to express ideas in few lines of code. Python can be used for things like:

- Server side aka back end mobile app and we development
- Software and desktop app development
- Processing big data and performing data analytics
- Creation of system scripts

In the late 1980's, history was about to be written. It was then that work on Python began. Shortly afterwards, Guido Van Rossum began his career in December 1989 at Centrum Wiskunde & Informatica (CWI) in the Netherlands. The programming language, Python succeeded was the ABC Planning Language, which had links to the Amoeba Operating System. He had taken the ABC syntax, with some of its fine features. It came with a lot of complaints too, so he completely fixed those issues and created a good programming language that eliminated all errors. The idea of the name came from the BBC TV Show - 'Monty Python's Flying Circus', as he was a big fan of the TV show, and also because he wanted a short, unique and less obscure name for his founding which is why he named it Python.

For a while he worked for Google, but at present, he works for Dropbox. The language was finally released in 1991. When it was released, it used very few codes to express ideas, when compared to Java, C ++ & C. Its design philosophy was also very good. Its main purpose is to provide code readability and enhanced productivity for advanced developers. When it was released it had more than enough power to provide several core data types exception handling and functions and classes with inheritance. Following are the illustrations of different versions of Python along with the timeline.

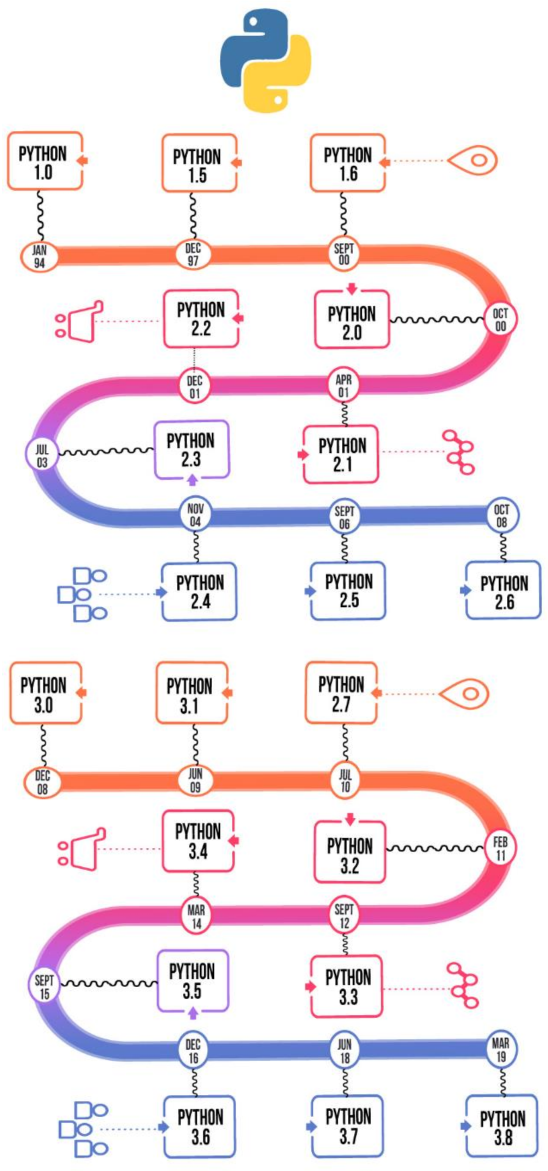


Figure 1: Different versions of Python over the years

2.2 ADVANTAGES OF PYTHON

1) Easy to Learn and Use: Python language is extremely easy to handle and understand for new learners and novice programmers. The python language is one of the most easily

available programming languages because it has simple and clear syntax and is not overly complicated. It additionally gives more stress on natural language. Because of its ease of study and usage and understanding, python codes can be penned and executed much quicker than other programming languages.

2) Mature and Supportive Python Community: Python was developed more than 30 years ago, which is a huge amount of time for any community of programming language to build and mature sufficiently to aid developers ranging from beginner to advanced levels. There are plenty of documentation, tutorials and Video guides available that beginners and programmers of any talent level can use and receive the guidance required to grow their knowledge in the python programming language.

3) Support from Well Known Industry Sponsors: Programming languages develop faster when an industry giant or sponsor supports it. For example, Java is supported by Oracle, PHP is supported by Facebook, and Sun & C# by Microsoft corporation. Python Programming language is extensively supported by Amazon Web Services, Facebook, and most importantly by Google.

4) Numerous Python Libraries, Packages and Frameworks: Due to its industry support and large supportive community of python developers, python has exemplary set of libraries that one can use to decide from and save time and effort in the early days of development. There are also tons of cloud media assistance and services that offer inter-platform support through package-like tools, which can be extremely helpful.

5) Adaptability, Efficiency, Reliability, and Speed: Ask any python programmer, and they will completely agree to the fact that the python language is reliable, efficient, and much quicker than most programming languages. Python can be used in almost every type of environment, and irrespective of the platform one is working, one will not face any type of performance loss issue while working on python.

One more interesting feature about adaptability of python language is that it can be used in many kinds of environment and situations like mobile apps, desktop apps, hardware programming, web development, and many more. The adaptability of python makes it easier and inviting to use due to its high number of applications and usage purposes.

6) Machine Learning, Artificial Intelligence, Big data and Cloud Computing: Artificial Intelligence, Cloud Computing, Machine Learning, and Big Data are some of the blazing trends in the computer science world at the moment, which helps lots of companies and organizations to remodel, transform and improve their work cycles and processes.

Python language is the 2nd most widely used language after analytics and science. A huge number of data processing workloads in the companies and organizations are driven by the python language itself. Many of the development and research takes place in python language due to its many functions and applications, including ease of understanding and organizing the practical data.

7) First-preference Language: Python language is the first preference for many new beginners and learners, the primary reason being the high demand for such programmers in the development market. Learners and programmers continue to look forward to learning a language that is always in high demand. Python is without any doubt, the hottest cake in the market now.

Many developers and data science learners are using python language for their project development. Learning python is one of the key sections in data science certification programs. The python language can provide a wide variety of successful career opportunities for learners. Due to the range of applications that python provides, one can have distinct career choices and will not remain stuck in any particular one.

2.3 DATA TYPES

In Python, everything is treated as an object and therefore every value in Python has a datatype associated with it. Some of the primitive data types used in the Python programming language are:

- Python numbers:
  - int
  - float
  - complex
- Python lists
- Python tuple
- Python strings
- Python sets
- Python dictionaries

2.4 PYTHON NUMBERS

The data types like int, float and complex come under the category of Python numbers. Example:

```
a = 5
print("Type of", a, "is", type(a))
#Output: Type of 5 is <class 'int'>

a = 2.0
print("Type of", a, "is", type(a))
#Output: Type of 2.0 is <class 'float'>
a = 1+2j
print("Type of", a, "is", type(a))
#Output: Type of (1+2j) is <class 'complex'>
```

Integers in the Python programing language do not have any definite size constraint as seen in other programming languages. They are only limited by the amount of memory available.

Floating point numbers have a precision up to 15 decimal places. A key point to note is that while the number 1 is an integer, 1.0 is treated as a floating number in python.

Complex numbers can also be represented in Python. They are given in the format a + b*j* where a is the real number part and b is the imaginary part.

2.5 PYTHON STRINGS

In Python, a string is a progression or sequence of characters, mostly Unicode characters. Unicode was developed to have every character in every language and bring similarity in encoding.

Characters enclosed between single quote or double quotes create a string. Triple quotes can also be used in Python but usually it is used to signify multiple line strings and function docstrings.

```
sample_string = 'Welcome to NHCE'
print(sample_string)

sample_string = "Namaste!"
print(sample_string)

sample_string = '''Good morning!'''
print(sample_string)

# triple quotes string can extend multiple lines
sample_string = """Hello, welcome to
NHCE, Bengaluru"""
print(sample_string)

#Output:
```

```
Welcome to NHCE
Namaste!
Good morning!
Hello, welcome to
NHCE, Bengaluru
```

We can access each character in a string using the Python feature called indexing and a range of characters using the Python feature called slicing. Indexing starts from 0. If we try to access a character out of the range of indices, then Python will raise an IndexError. The index must always be an integer. Floats or any other type of data type cannot be used for indexing as this will result into TypeError.

Python also provides negative indexing for its sequences. An index of -1 refers to the last element, -2 to the second last element and so on. One can access a range of elements in a string by using the slicing operator :(colon).

```
my_college = 'New Horizon College of Engineering'
print('my_college = ', my_str)

print('my_college[0] = ', my_college[0])

print('my_college[-1] = ', my_college[-1])

print('my_college[4:11] = ', my_college[4:11])


print('my_college[5:-2] = ', my_college[4:-15])


#Output:
my_college = New Horizon College of Engineering
my_college[0] =  N
my_college[-1] =  g
my_college[4:11] =  Horizon
my_college[5:-2] =  Horizon College
```

Strings are immutable in the Python programming language. This means that content of a string cannot be changed after it has been assigned. Reassigning can be done by assigning different strings to the same name. The del keyword can be used to delete the entire string from memory but individual characters of the string cannot be deleted or removed.

Concatenation is the process of joining of two or more strings into a single one. The addition (**+**) operator is used for concatenation in Python. Just writing two string variables together also concatenates them. A string can be repeated multiple time by using the multiply (*****) operator along with the string.

```
my_str1 = 'Welcome '
my_str2 ='to NHCE!'

print('my_str1 + my_str2 = ', my_str1 + my_str2)

print('my_str1 * 3 =', my_str1 * 3)

#Output:
my_str1 + my_str2 =  Welcome to NHCE!
my_str1 * 3 = Welcome Welcome Welcome
```

2.6 PYTHON LISTS

The list in Python is the closest data type that resembles the array data type seen in other programming languages. It is an ordered sequence of elements. It is one of the widely used data types in Python due to its flexible nature.

A list can have elements of various data types and therefore is both homogenous and heterogenous in nature. It is also mutable, i.e., its content can be changed later. A list is denoted by using the square brackets [ ] with the elements in between the square brackets separated by a comma. Example:

```
     a = [1, 2.0, "Rahul", [738, "NHCE"]]
print(a)
#Output: [1, 2.0, 'Rahul', [738, 'NHCE']]


print(a[3])
#Output: [738, 'NHCE']
```

Lists can also be sliced. Extracting a specific sub list from list is called slicing. This is carried out using a colon (:) called the slicing operator.

```
# Python program to demonstrate removal of elements in a List

# Creating a List
L = ['R','A','H','U','L','.','M','D','I','N','E','S','H']
print("Initial List: ")
print(L)

# Print elements of a range using Slice operation
Sliced_List = L [3:8]
print("\nSlicing elements in a range 3-8: ")
print(Sliced_List)

# Print elements from a  pre-defined point to end
Sliced_List = L [5:]
print("\nElements sliced from 5th element till the end: ")
print(Sliced_List)

# Printing elements from beginning till end
Sliced_List = L [:]
print("\nPrinting all elements using slice operation: ")
print(Sliced_List)
#Output:
Initial List:
['R', 'A', 'H', 'U', 'L', '.', 'M', 'D', 'I', 'N', 'E', 'S', 'H']

Slicing elements in a range 3-8:
['U', 'L', '.', 'M', 'D']

Elements sliced from 5th element till the end:
['.', 'M', 'D', 'I', 'N', 'E', 'S', 'H']

Printing all elements using slice operation:
['R', 'A', 'H', 'U', 'L', '.', 'M', 'D', 'I', 'N', 'E', 'S', 'H']
```

Lists can also be indexed in two ways:

- Forward Indexing
- Backward Indexing

Forward indexing is the usual way we index by starting from 0 for the first element. In backward indexing, -1 refers to the last element, -2 to the second last element and so on.

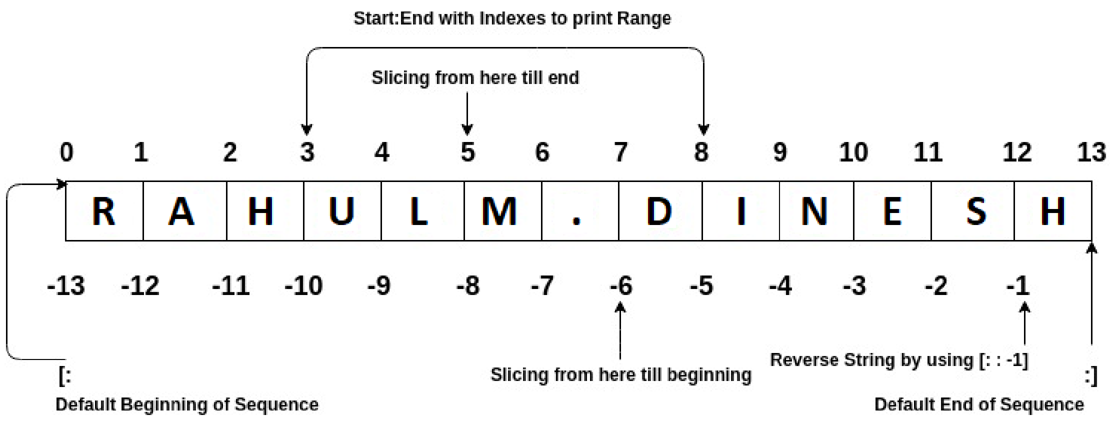

Start:End with Indexes to print Range
Slicing from here till end
0 1 2 3 4 5 6 7 8 9 10 11 12 13
R A H U L M . D I N E S H
-13 -12 -11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1
[:
Default Beginning of Sequence
Slicing from here till beginning
Reverse String by using [: : -1]
:]
Default End of Sequence

Figure 2: String Indexing in Python

```
# Negative indexing in lists
list1 = ['R', 'a', 'h', 'u', 'l']
print("list1[-1] is", list1[-1])
print("list1[-5]", list1[-5])

#Output:
list1[-1] is l
list1[-5] R
```

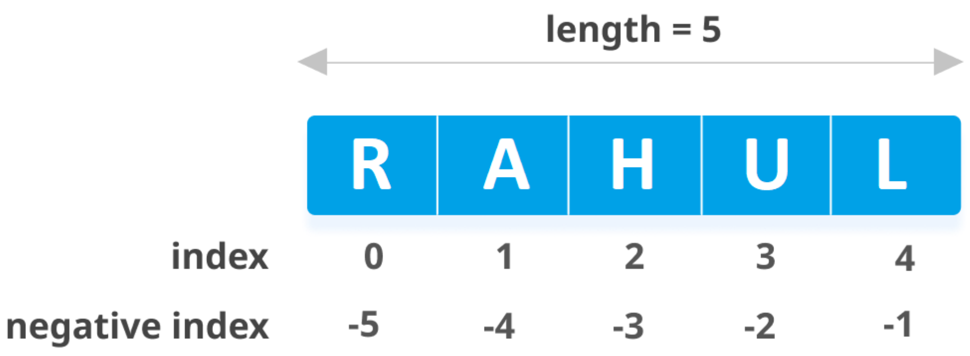


Figure 3: Negative or Backward Indexing in Python

We can add an item to a list which we create by using the method append() or add several items using extend() method by providing a list as an argument.

```
# Appending and Extending lists in Python
odd = [10, 8, 6]
odd.append(4)
print(odd)
odd.extend([2, 0, -2])
print(odd)

#Output:
[10, 8, 6, 4]
[10, 8, 6, 4, 2, 0, -2]
```

2.7 PYTHON TUPLES

The tuple in Python is very similar to the list data type discussed earlier. The only difference between a list and a tuple is that, while a list is mutable, a tuple is immutable, i.e., its content cannot be changed once initialized.

A tuple is denoted by using the parentheses ( ) with the elements in between the parentheses separated by a comma. Example:

```
# Different types of tuples
tuple1 = ()
print("Empty tuple: ", tuple1)

tuple2 = (1, 2, 3)
print("Tuple having integers: ", tuple2)

tuple3 = (738, "Rahul", "2.0")
print("Tuple with mixed datatypes: ", tuple3)

tuple4 = ("NHCE", [7, 3, 8], (1, 2, 3))
print("Nested tuple: ", tuple4)

#Output:
Empty tuple:  ()
Tuple having integers:  (1, 2, 3)
Tuple with mixed datatypes:  (738, 'Rahul', '2.0')
Nested tuple:  ('NHCE', [7, 3, 8], (1, 2, 3))
```

However, to create just a tuple with a single element is a bit difficult. Enclosing just the single element in parentheses will not create a tuple. A trailing comma is required after the element to indicate that it is, in fact, a tuple.

```
tuple1 = ("hello")
print("Type of tuple1: ", type(tuple1))

# Creating a tuple having one element
tuple2 = ("hello",)
print("Type of tuple2: ", type(tuple2))

# Parentheses are optional
tuple3 = "hello",
print("Type of tuple3: ", type(tuple3))

#Output:
Type of tuple1:  <class 'str'>
Type of tuple2:  <class 'tuple'>
Type of tuple3:  <class 'tuple'>
```

The indexing followed in tuples is similar to that followed in lists. Tuples can also be sliced using the colon (:) as seen in lists.

```
my_tuple = ("N", "H", "C", "E")

print("my_tuple[0]:", my_tuple[0])
print("my_tuple[3]:", my_tuple[3])
print("my_tuple[-2]:", my_tuple[-2])
print("my_tuple[1:3]:", my_tuple[1:3])

# nested tuple
n_tuple = ("NHCE", [7, 3, 8], (1, 2, 3))

# nested index
print("n_tuple[0][1]: ", n_tuple[0][1])
print("n_tuple[1][1]: ", n_tuple[1][1])
```

```
#Output:
my_tuple[0]: N
my_tuple[3]: E
my_tuple[-2]: C
my_tuple[1:3]: ('H', 'C')
n_tuple[0][1]:  H
n_tuple[1][1]:  3
```

As discussed earlier, tuples are immutable unlike lists. That is, once a tuple is assigned, the elements of a tuple cannot be changed. But, if the element inside the tuple is a mutable Python data type like list then the nested elements can be changed or are mutable. Reassignment i.e., assigning a tuple to different values is permitted.

```
# Changing tuple values
my_tuple = (7, 3, 8, [2, 4, 6])

# my_tuple[1] = 9
# TypeError: 'tuple' object does not support item assignment

# Item of mutable element in a tuple can be changed
my_tuple[3][0] = 9
print("my_tuple[3][0]:", my_tuple)

# Tuples can be reassigned
my_tuple = ("N", "H", "C", "E")
print("Reassigned my_tuple:", my_tuple)

#Output
my_tuple[3][0]: (7, 3, 8, [9, 4, 6])
Reassigned my_tuple: ('N', 'H', 'C', 'E')
```

2.8 PYTHON SETS

An unordered collection of items is called a set in the python programming language. Every element of the set is unique, that is there are no duplicates and it is immutable. The point to be noted is that a set itself is mutable. We may remove or add elements from it. Set operations like union, intersection, symmetric difference can be performed using sets.

A set is defined by placing all the elements inside curly braces { }, each separated by a comma, or by using Python’s built-in set() method. It can have any number of elements and they may be of different types or in other words heterogenous. Importantly, a set cannot have mutable data types like lists, sets or dictionaries as its elements.

```
sample_set = {1, 2, 3}
print(sample_set)

sample_set = {1.0, "NHCE", (7, 3, 9)}
print(sample_set)

#Output:
{1, 2, 3}
{1.0, (7, 3, 9), 'NHCE'}
```

2.9 PYTHON DICTIONARIES

An unordered collection of data values, used to store data values like a map is called a dictionary. Unlike other Data Types in the Python programming language that hold only a single value as an element, a dictionary in Python holds key: value pairs as its elements. Key value is provided in the dictionary to make it more optimized.

A dictionary is denoted by using the curly braces { } with the elements in between the parentheses separated by a comma. Each element is a key: value pair. Example:

```
my_dict = {}
print("Empty dictionary: ", my_dict)

my_dict = {1: 'NHCE', 2: 'Bengaluru'}
print("Dictionary with integer keys: ", my_dict)

my_dict = {'name': 'Rahul', 1: [2, 4, 3]}
print("Dictionary with mixed keys: ", my_dict)

my_dict = dict({1:'Rahul', 2:'NHCE'})
print("Dictionary created using dict(): ", my_dict)

my_dict = dict([('Name','Rahul'), ('USN',738)])
print("Dictionary created using dict(): ", my_dict)

#Output:
Empty dictionary:  {}
Dictionary with integer keys:  {1: 'NHCE', 2: 'Bengaluru'}
Dictionary with mixed keys:  {'name': 'Rahul', 1: [2, 4, 3]}
Dictionary created using dict():  {1: 'Rahul', 2: 'NHCE'}
Dictionary created using dict():  {'Name': 'Rahul', 'USN': 738}
```

A dictionary uses keys whereas indexing is used in other data types to access elements or values. Keys can be used to retrieve their corresponding values either inside square brackets [] or with the get() method. A KeyError is raised in case a key is not found when we use the square brackets [] in the dictionary. The get() method returns None if the key is not found unlike the square bracket method.

```
my_dict = {'name': 'Rahul', 'age': 20, 'address': 'Bengaluru'}


print("Name:", my_dict['name'])
print("Age:", my_dict.get('age'))
print("Address:", my_dict.get('address'))
print("Address:", my_dict['address'])
```

```
#Output:
Name: Rahul
Age: 20
Address: Bengaluru
Address: Bengaluru
```

Dictionaries are mutable, that is its values can be changed even after assignment. Using an assignment operator, we can add new items or change the value of existing items. The existing value gets updated, if the key is already present. A new (key: value) pair is added to the dictionary in case the key is not present.

```
my_dict = {'name':'Rahul', 'age': 20,'address':'Bengaluru'}

# update value
my_dict['age'] = 21
print(my_dict)

# add item
my_dict['address'] = 'Bengaluru'
print(my_dict)
#Output:
{'name': 'Rahul', 'age': 21}
{'name': 'Rahul', 'age': 21, 'address': 'Bengaluru'}
```

2.10 FUNCTIONS IN PYTHON

Functions are an easy to use and convenient way or method to divide the Python code into blocks of code thus making the entire code more ordered which enhances or increases the readability and shortens the overall length of the program. Using functions also helps us in reusing the code thus reducing the typing load and saving time.

A function is defined by using the keyword *def* followed by the function name and a pair of parentheses which encloses the arguments if any. This is followed by the function body which starts after an indentation (usually a tab) from the margin. The function body may also have a return statement that returns an object to where the function was invoked / called.

```
def greeting(name):
    print("Hello", name)

    return "Your name is Rahul"

s = greeting("Rahul")
print(s)

#Output:
Hello Rahul
Your name is Rahul
```

The portion of a program where the variable is recognized is called the scope of a variable. Variables and parameters defined inside a function are not visible or accessible from outside the function or in other words their scope is limited to within the function. A variable’s lifetime is the time period throughout which the variable exists in the memory. Variables inside a function have a lifetime that is as long as the function executes. Once a function is returned, the variables inside them are destroyed. Hence, the value of a variable from a function’s previous calls is not remembered by a function.

```
def sample_func():
    a = 5
    print("Value inside function:", a)

a = 10
sample_func()
print("Value outside function:", a)

#Output:
```

```
Value inside function: 5
Value outside function: 10
```

In python there are 2 types of function as seen in other programming languages:

- Built-in functions: Those functions whose functionality is pre-defined in Python are called as built-in functions. The python interpreter has many pre-defined functions that are always present for a programmer to use. There are several built-in functions in Python. A few are listed below:
    - abs( ): It takes a single argument and returns the absolute value of the number
    - bin( ): It takes a single argument and returns the binary equivalent of the number
    - sum( ): It can take a list as argument and returns the sum of all elements in the list
    - max( ): It can take a list as argument and returns the maximum element among all elements in the list
    - min( ): It can take a list as argument and returns the minimum element among all elements in the list
    - sorted( ): It sorts the elements of a sequence like a string or a list and returns a collection in the sorted order
    - input( ): This function is used to take in an input from the user. It takes the input as a string.
- User defined functions: These are functions written by a programmer to do a specific task and help reduce the typing load and length of code.

CHAPTER 3

FUNDAMENTALS OF TKINTER

3.1 INTRODUCTION

To create GUI applications, an inbuilt python module called Tkinter is used. It is one of the most widely used packages for developing GUI applications in Python as it is very straightforward, simple and easy to work with. One doesn’t have to bother about installing the Tkinter package separately as it comes with the Python installation package. Tkinter provides an object-oriented touch or interface to the Tk GUI toolkit.

Some less used Python Libraries which are available for developing GUI applications in Python are:

- Kivy
- Python Qt
- wxPython

Among all, Tkinter is the most widely used for creating graphical user interfaces.

3.2 WIDGETS

The GUI application uses various widgets provided by Tkinter which gives various controls to users to interact with the application like:

- Labels
- Buttons
- MenuBars
- ComboBoxes
- RadioButtons
- CheckBoxes

among many others.

WIDGETS	DESCRIPTION
Label	This widget is used to display text or image on the window/frame
Button	This widget is used to add buttons to the user interface
Canvas	This widget allows one to draw pictures and different types of layouts like texts, graphics etc.
Entry	This widget is used to take as input, a single line text entry from user
Frame	This widget is used as box or container. It holds and organizes the widgets in an orderly fashion
SpinBox	This widget allows users to select from a given number of values
ComboBox	This widget contains a down arrow to select from a list of options
CheckButton	This widget displays a number toggle buttons which represent various options from which user can select any number of options.
RadioButton	This widget is similar to the CheckButton but allows only one option to be selected
Scale	This widget is used to provide a slider which allows the user to select any value from the scale
Scrollbar	This widget is used to scroll down the contents of any widget/frame
Text	This widget allows the user to edit multiline text and format it
Menu	This widget is used to create different kinds of menus used by in an application

Table 1: Various widgets available in Tkinter

3.3 GEOMETRY MANAGERS

In order to arrange or organize or place the widgets in the main window, Tkinter provides a few geometric configurations. The Application Layout of the graphical user interface is mainly managed by the geometric managers of the Tkinter package.

It is key to note here that each window and Frame in the GUI application is only allowed to use only one particular type of geometry manager. However, different frames can have different geometry managers, even if they're placed in a frame or window using a different geometry manager. There are mainly three types of Geometry Managers:

- Pack
- Grid
- Place

I have employed the Grid geometry manager throughout my application code.

3.4 LABELS

A Label is a Tkinter widget that is used to create display boxes where we can hold text and/or images. The text displayed in a Label can be modified by the developer at point of time. It is also used to perform operations like underlining a part of the text and span the text across many lines. It is key to note that a label can use a single font at an instant to show text. To use a label, one has to define what to display in it (like a text, a bitmap, or an image).

The syntax for creating a Label is:

```
l = Label (root, option = value, …)
```

where, root is the parent window or frame where the Label is intended to be placed and options signify the various options, we can use upon the Label like

- text: the text to be displayed in the Label
- bg: the background color of the label
- image: to display an image in the label
- font: the font and font size to be used in the label, and many other options.

3.5 BUTTONS

The Button is one of the basic Tkinter widgets. It is used by the user to interact with the application. If the button is pressed by a mouse click then some command might be provoked. Like Labels, they too can also contain text and images. A button can only display text in a single font whereas labels can display text in various fonts. The text of a button can span many lines. A Python function or method can be related with a button. This function or method will be executed, whenever the button is pressed in some way.

The syntax for creating a Button is:

```
b = Button (root, option = value, …)
```

where, root is the parent window or frame where the Button is intended to be placed and options signify the various options, we can use upon the Button like

- text: the text to be displayed in the Button
- bg: the background color of the Button
- command: the command to be executed when the button is pressed
- width: the width of the button

among many other options.

3.6 COMBOBOX

This widget is a mix of an Entry widget and a drop-down menu widget. In the GUI application, when we implement a combobox, we will see the usual text entry area along with a downward-pointing arrow, which when clicked displays a drop-down menu. If the user clicks on any one option, then that choice becomes the current content of the entry. But the user may still type text directly into the entry or edit the current text. (when it has focus and if state is not readonly)

The syntax for creating a Combobox is:

```
cb = Combobox (root, option = value, …)
```

where, root is the parent window or frame where the Combobox is intended to be placed and options signify the various options we can use upon the Combobox like:

- values: the various choices that will be displayed in the drop-down menu
- bg: the background color of the combobox
- fg: the color of the text in the combobox
- width: the width of the combobox

among many other options.

3.7 FRAME

The Frame widget in Tkinter is used to consolidate, group and organize the different widgets in an ordered fashion. The Frame widget is basically a box (an invisible container) whose job is to hold other widgets and organize them with respect to each other widget. The Tkinter frame widget takes up a rectangular area on the screen. It acts as a foundation class which then places and organizes complex widgets.

The syntax for creating a Frame is:

```
f = Frame (root, option = value, …)
```

where, root is the parent window or frame where the Frame is intended to be placed and options signify the various options we can use upon the Frame:

- relief: the type of the border the Frame has
- bg: the background color of the Frame
- height: the height of the Frame
- width: the width of the Frame

among many other options.

CHAPTER 4

FUNDAMENTALS OF DBMS

4.1 INTRODUCTION

A collection of logically related data is generally defined as a database. In other words, it is a collection of large volumes of facts and figures in an organized and orderly manner. It has few inherent properties:

- It personifies some facet of the actual world, called the miniworld or universe of discourse.
- It is a collection of data with some inherent meaning where the data is logically coherent.
- A database is designed, developed, and filled with data for a peculiar purpose. It has a targeted group of users. It can be of any size and of differing complexity.

A database management system (DBMS) is a collection of applications that enable users to create and maintain a database. It can be described as an all-purpose software application that provides the platform for defining, constructing, and manipulating databases for various uses. MySQL, Microsoft Access, Sybase, Oracle and IBM DB2 etc. are some examples of popular DBMS software.

4.2 CHARACTERISTICS OF A DBMS

A database management system generally has the following features:

1. Controlling Redundancy: A good DBMS has to ensure minimal or nil redundancy. Redundancy refers to the storing of the exact same data many times. This leads to many issues like duplication of effort, wastage of storage space and inconsistency. Storing each logical data item at only one place in database is generally a feature of an ideal DBMS software.

2. Data Consistency: By managing redundancy of data, the consistency of data is achieved. If a data item is seen only once, then any update to its value has to be done only once and the new value of item is immediately available to all users accessing the database.
3. Data Integration: Data in the database is stored as tables or relations. A single database contains several tables and relationships can be established between tables. This makes it easy to query, i.e., retrieve and update data.
4. Enforcing Integrity Constraints: Consistency rules or Integrity constraints can be enforced to a database so that the actual data can be entered into database. The constraints may be enforced to data within a single record or they may be enforced to relationships between different records. Examples: The examples of integrity constraints are:
   (i) 'Marks' in a Grading system cannot be greater than the ‘Maximum Marks’
   (ii) The age of a person cannot be less than 0
5. Restricting Unauthorized Access: When multiple users jointly use a database, it is likely that some users will not have the permission to access all data in the database. A DBMS has a security and authorization subsystem, which the Database Administrator uses to make accounts and to specify restrictions on the account. The DBMS should then apply these restrictions accordingly.
6. Providing Storage Structures for Efficient Query Processing: Indexes (typically in the form of trees and/or hash tables) are maintained by DBMS softwares that are used to enhance the execution time of queries and updates. The choice of which indexes to make and maintain is a part of the responsibility of the DBA along with the actual database design and tuning. The query processing and optimization module is the one that is responsible for selecting an optimized query execution process for each query given to the system.

7. Providing Backup and Recovery: A DBMS must have the provision for recovering from hardware or software crashes. The backup and recovery subsystem of the DBMS is the main subsystem that is responsible for recovery of data.
8. Providing Multiple User Interfaces: Because many different types of users with differing levels of knowledge on how to use a database, a DBMS provides a wide variety of interfaces for the user. These include query languages, form-style interfaces, menu-driven interfaces, programming language interfaces and natural language interfaces.
9. Sharing of Data: Many users can have the permission to access the same part of information at the same time. The remote users can also share the exact data. Similarly, the data of the same database can be shared between various applications and programs.

4.3 DATA MODEL

One basic characteristic of the approach that a database takes is that it provides a certain level of data abstraction by hiding implementation details of storage of data that are not required by most database users. A Data Model is a group of concepts which is used to describe the Database's logical structure i.e., the relationships, data types and constraints that should be applied on the data.

Categories of Data Model:

- High level or conceptual data models: They provide approaches on how users perceive the data. Eg. the ER model, which uses concepts such as
- Low level or physical data models: These type of data models provides concepts that characterize the details of how data is stored in the computer's memory. e.g. record orderings, record formats, access path, index etc.
- Representational data models: They provide the characteristics which is in between the above two extremes.

4.4 THREE - SCHEMA ARCHITECTURE

The aim of the 3-schema architecture is to differentiate the user applications and the physical database. In this architecture, schemas can be determined at the following three levels:

- Internal level: which has an internal schema
- Conceptual level: which has a conceptual schema
- External level: which includes a number of user views or external schemas.

The system catalog stores information about all three schemas.

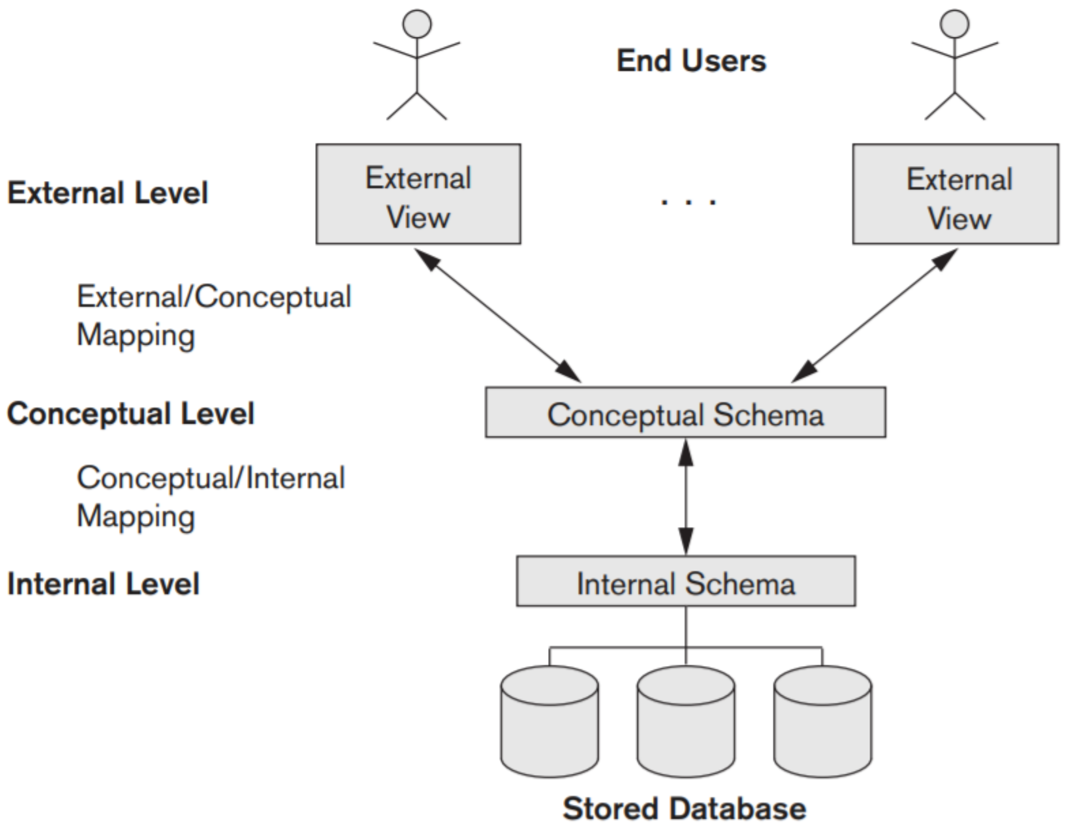


Figure 4: Three Schema Architecture

4.5 DBMS COMPONENT MODULES

Figure 4 illustrates, in a very simple form, the typical components of a DBMS. The figure is divided into two halves. The top portion of the figure refers to the various users of the database and their respective interfaces. The lower portion shows the various internal modules of the DBMS which are responsible for the processing of transactions and storage of data.

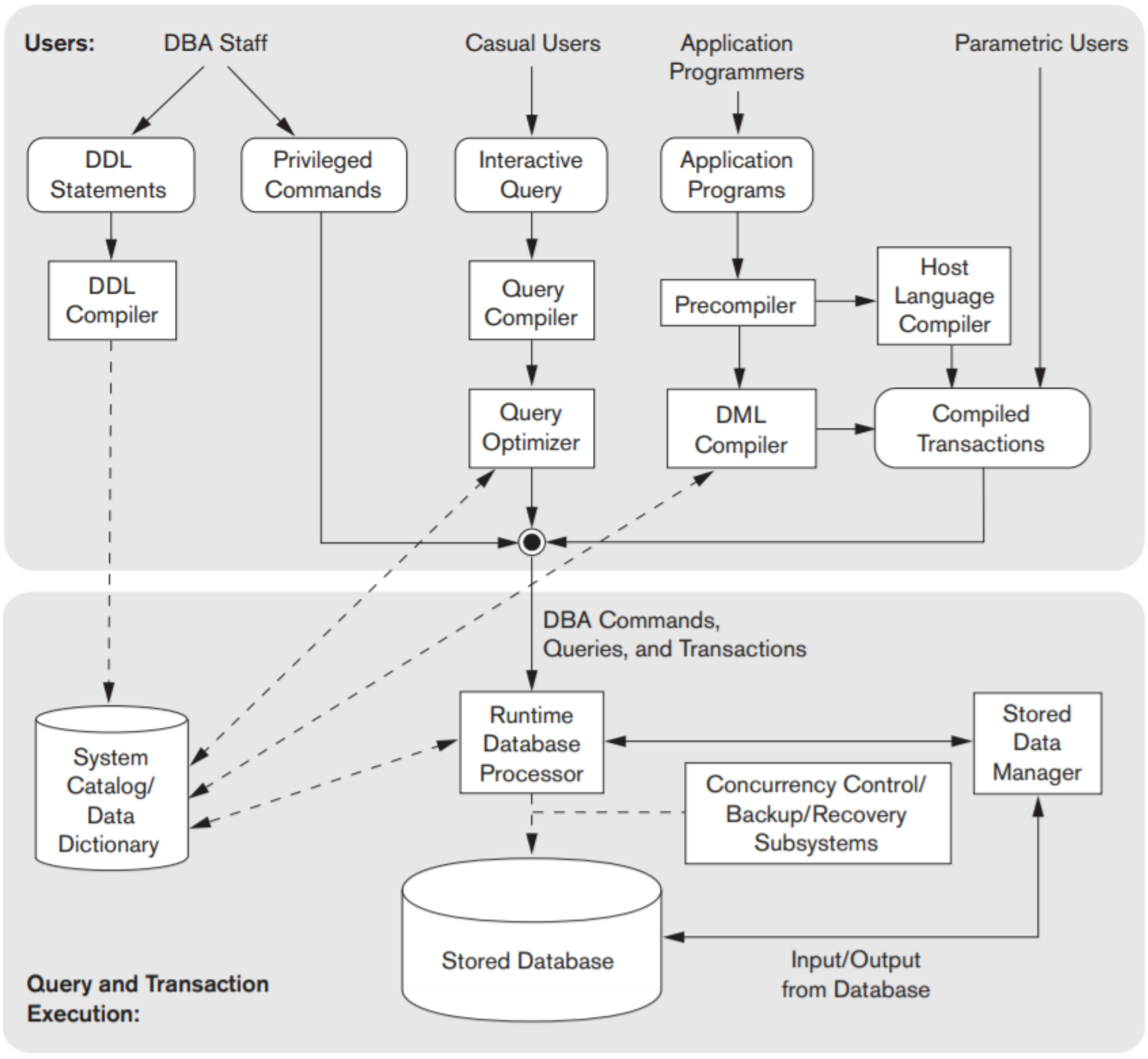


Figure 5: DBMS Component Modules

The DBMS catalog and the database are saved and stored on the disk. A module of DBMS named stored data manager manages access to DBMS data that is stored on disk whether it is part of the catalog or database. Various users such as

- Casual users & DBA staff work with interactive interfaces to create queries.
- Application programmers who write programs via a host language.

- Parametric users who make enter data into the database by providing parameters to previously defined transactions.

The DB Administrator staff's job is to define the database and tune it using DDL and other privileged commands.

DDL compiler processes schema definitions which are specified in the Data Definition Language, and stores the description of the database schema in the catalog. Casual users are users who occasionally use the database using interactive query interface. These queries and transactions are parsed and analyzed for correctness of the query operations along with the names of data elements by the query compiler that compiles. Then the internal query is optimized by the query optimizer.

The query optimizer performs reordering, rearrangements of operations, deletion of redundancies and usage of correct indexes & algorithms. It looks up the system catalog for information about the storage and for generating executable codes. These executable codes in turn make calls to runtime processor.

Application programmer code programs in host language like COBOL, C and Java that are then submitted to the precompiler. The precompiler extracts the Data Manipulation Language commands from an application program and sends it to the Data Manipulation Language compiler for compilation. The remaining part of the program is sent to the host language compiler. The object codes for the Data Manipulation Language commands and rest of the program are linked, thus creating a canned transaction whose executable code contains calls to the runtime database processor. Parametric users supply the parameters to these canned transactions so that they can run these transactions many times.

Runtime database processor executes

- Privileged commands
- Executable query plans
- Canned transactions with runtime parameters.

- It works along with the system dictionary, stored data manager which in turn uses primitive OS services for carrying out low level input and output operations between the disk and memory.
- It also manages some aspects of control of buffers in the main memory. Some DBMS have their own buffer management.

Concurrency control, backup and recovery manager are integrated into the working of the runtime database processor for purposes of transaction management.

4.6 ENTITY-RELATIONSHIP (ER) MODEL

The Entity-Relationship (ER) Model is an attractive high level conceptual data model. It has an entity which may be an object with a physical existence like a particular car, house, person or employee or it may be an object with a coneptual existence like an organization, a profession, or a university course. Each enity has attribtes—the defnite properies that characterize it. For exmple, a student entty my be described by the student’s name, age, address, USN etc.

For example a specific person entity may have Name=‘Rahul', USN= ‘1NH18CS738’, Address ='738, NHCE, Bangalore, Karnataka', Sex='M', BirthDate='09-MAY-00’. Each attribute has a data type associated with it e.g. integer, string, date etc.

A relationshi is an assoiation amog two or more entiies. Whenever an attribte of one entity type refers to the atribute of another entty type, some relatonship exists.

For example, in Figure 5, the WORKS_FOR relationship type, in which EMPLOYEE s and DEPARTMENTs entities participate. The MANAGES relationship type in which EM PLOYEEs and DEPARTMENTs entities participate.

One or more relationship types can exist with the same participating entity types. For example, MANAGES and WORKS_FOR are distinct relationships between EMPLOYEE and DEPARTMENT, but with different meanings and different relationship instances.

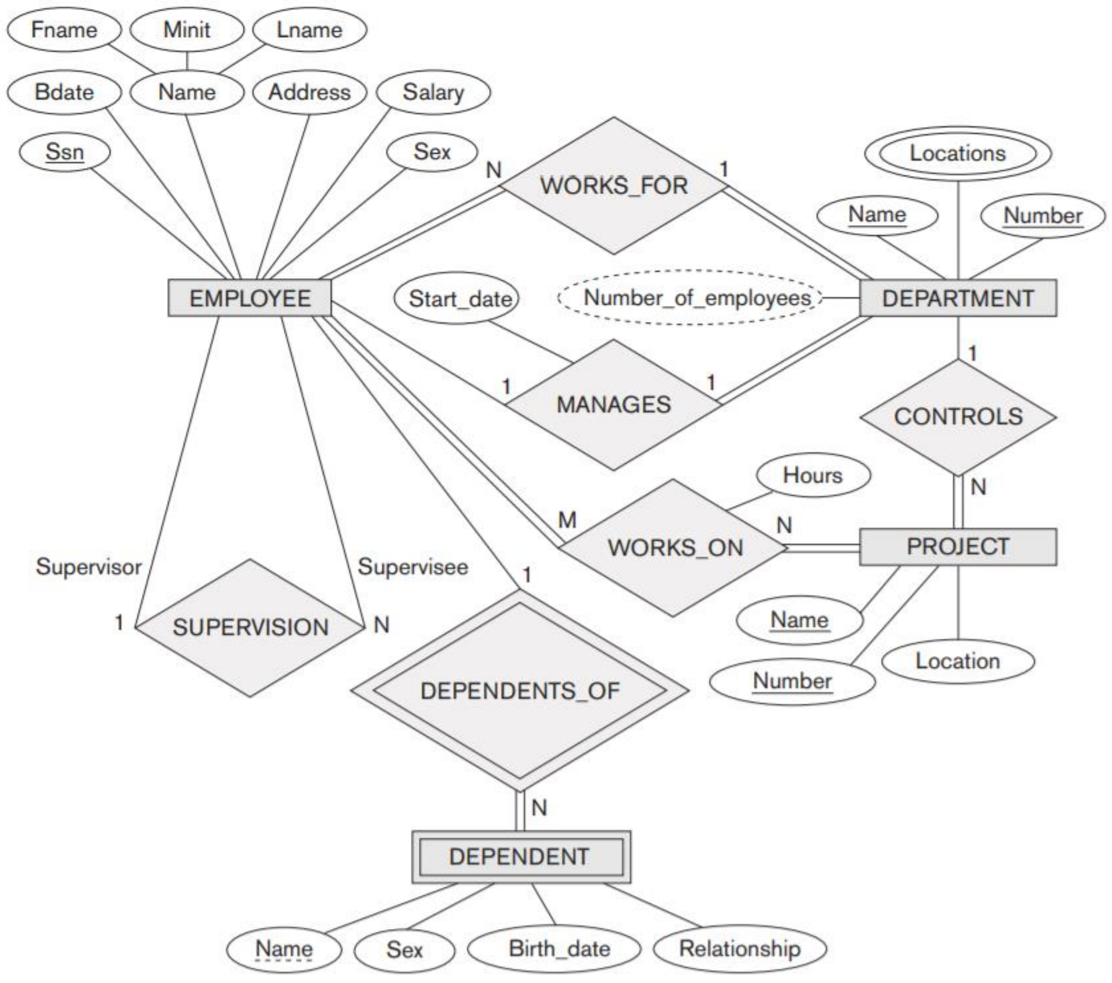


Figure 6: ER Diagram of a Company Database

4.7 RELATIONAL SCHEMA

A relation schema R, where R is the name of the relation is denoted by R(A_1, A_2, A_3. . . .,A_{n-1}, A_n), where A_1, A_2, A_3. . . .,A_{n-1}, A_n is a list of its attributes. A relation schema is used to describe a relation R

- Where, each A_i is the name of each unique attribute
- Domain of A_i is D and is denoted by dom(A_i).
- The number of attributes n in R is the degree of the relation R

An example of a relation schema for a relation of degree 5, which describes university students, is the following: STUDENT(Stud_Name, USN, Mobile_Num, Department, DOB) For this relation schema, STUDENT is the name of the relation, which has 5 attributes.

CHAPTER 5

FUNDAMENTALS OF SQL

5.1 INTRODUCTION

SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in a relational database. SQL is the standard language for Relational Data base Syste m. All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language.

SQL is widely popular because it offers the following advantages:

- Allows users to access data in the relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in a database and manipulate that data.
- Allows to be embedded within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create view, stored procedure, functions in a database.
- Allows users to set permissions on tables, procedures and views.

5.2 SQL COMMANDS

There are five types of SQL commands:

- DDL: Data Definition Language
- DML: Data Manipulation Language
- DCL: Data Control Language
- TCL: Transaction Control Language
- DQL: Data Query Language

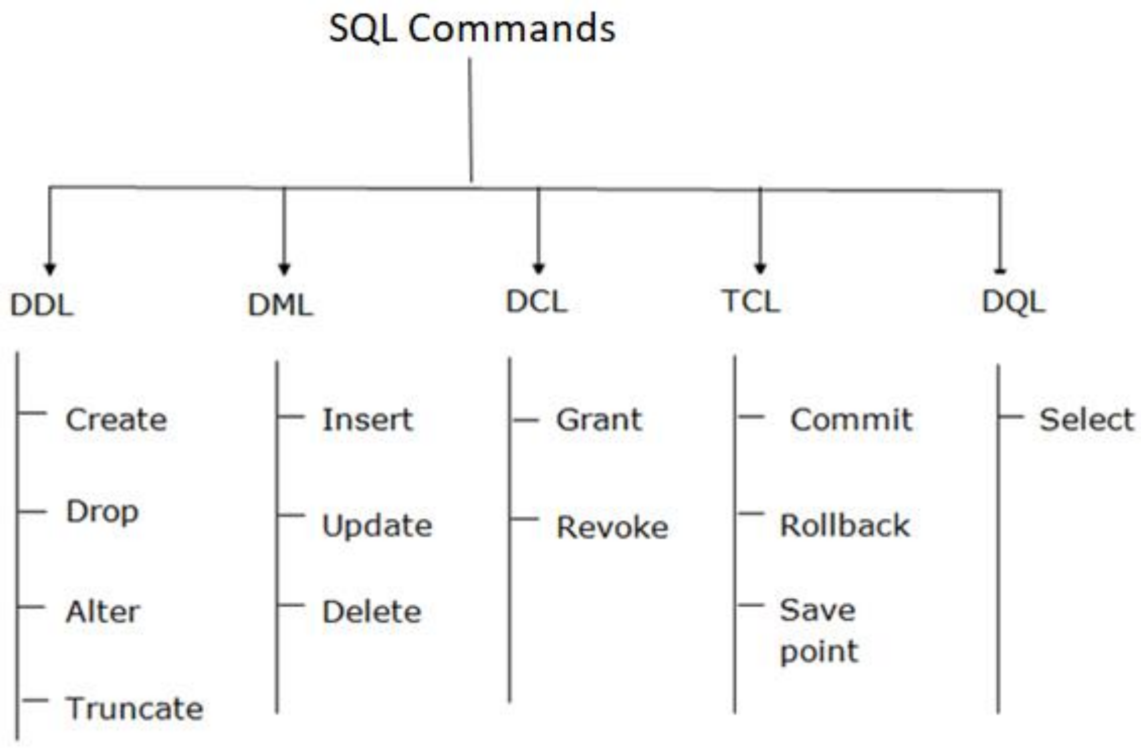


Figure 7: Various types of SQL commands

5.3 DATA DEFINITION LANGUAGE

The DDL commands are used for making changes to the structure of a table. DDL commands are by default auto-committed which means that that all changes made in the database using DDL commands are permanent. The DDL commands include:

- ALTER: To alter the structure of the database by adding/deleting a column etc.
- TRUNCATE: To delete all existing content/records while preserving structure
- DROP: To delete both the structure and contents of the table
- CREATE: To create new tables and databases

5.4 DATA MANIPULATION LANGUAGE

The DML commands are used for making changes to the content/records of a table. DML commands are by not default auto-committed which means that that all changes made in the database using DML commands are not permanent and hence can be roll backed. The DML commands include:

- INSERT: To insert a row or value into the table or a specific column of a table
- UPDATE: To update a value of a specific column of a table
- DELETE: To delete rows from the table

5.5 DATA CONTROL LANGUAGE

The DCL commands are used for granting or take back authority from any database user. The DCL commands include:

- GRANT: To grant access privileges to a database user
- REVOKE: To revoke access privileges from a database user

5.6 TRANSACTION CONTROL LANGUAGE

The TCL commands are used only along with commands belonging to the DML category like INSERT, UPDATE and DELETE. TCL commands include:

- COMMIT: To ensure that all transactions processed are saved to the database
- ROLLBACK: To undo all transactions that have processed but not saved onto the database
- SAVEPOINT: To roll back the transactions to a certain point but not entirely like the ROLLBACK command does

5.7 DATA QUERY LANGUAGE

The DQL command is used to fetch data from the database and has only one command:

- SELECT: To select all or specific attributes from a table if required by using a WHERE clause to describe a specific condition

CHAPTER 6

DESIGN

6.1 DESIGN GOALS

This mini project has ensured that the user has an interactive and explorable environment. The interface is user friendly, simple to understand and has tried to ensure that there are no bugs.

I have tried to ensure that the code used is simple yet logical without compromising on the final output.

The mini-project has two components:

- The front end, which is a Graphical User Interface or GUI developed using Python's Tkinter package
- The back end, where the data is stored in the form of a database. Data is queried using the SQL Lite relational database management system (RDBMS).

6.2 DATABASE STRUCTURE

The mini project's database part/back end is designed using the SQL Lite relational database management system (RDBMS). The database I designed for this project has several tables or relations which are required for storing user information and other essential data which are required for the proper querying of information from the database.

There are broadly, 4 key entities aka relation or tables in my mini project:

- The USERS entity: This entity stores all user related information like the user ID assigned at the time of registration, user's name, username, password and address. Depending upon the user type (individual or organization), additional attributes like age, gender, date of birth, donor status may be applicable.

- The BLOOD INVENTORY entity: This entity stores the quantity of blood that each organization type user has in their blood inventory. The information stored in this database is used for helping the users of individual user type to locate blood banks within their city which have a particular type of blood group.
- The PLASMA INVENTORY entity: This entity stores the quantity of plasma that each organization type user has in their plasma inventory. The information stored in this database is used for helping the users of individual user type to locate plasma banks within their city which have a particular type of blood group's plasma.
- The ORGAN INVENTORY entity: This entity stores the count of various lifesaving organs that each organization type user has in their organ inventory. The information stored in this database is used for helping the users of individual user type to locate organ banks within their city which have a particular organ available with them.

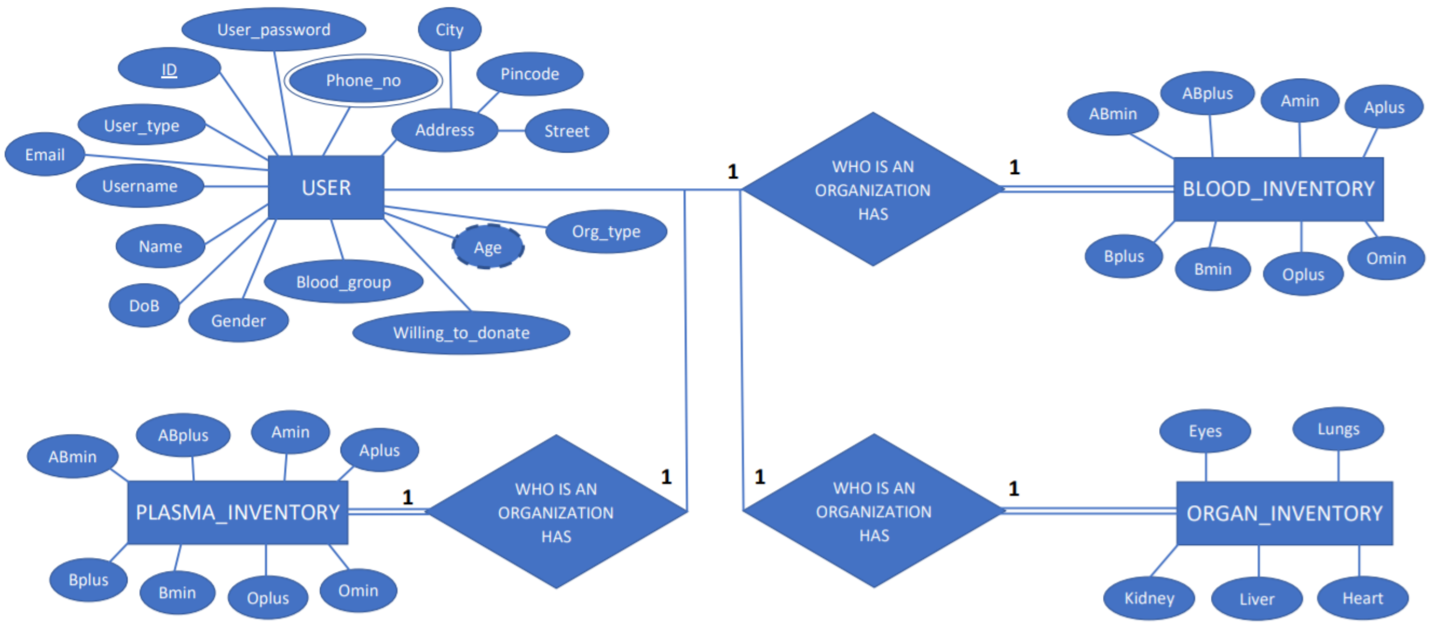


Figure 8: ER Diagram of the Database

Apart from the above 4 entities that are used for storing both type of user type related data, additional entities were also used to provide data integrity and referential integrity like:

- BG_Chart: This entity provided the data to match a blood group with their corresponding compatible blood groups that can be used as substitutes.
- Blood_Group_Types: This entity contains all the various types of blood groups known to us like A+, A-, B+, B-, AB+, AB-, O+ and O-. This table was created so that invalid blood groups would not be added to the user data.
- Gender: This entity contains data of genders like male and female and is used to ensure any other type of data is not accepted.
- Organization_types: This entity enlists the data of the type of possible organization types available for the organization user type like Hospital, Blood Bank or Organ Bank.
- Serviceable_cities: This entity contains data of all cities currently serviced by the application.
- User_types: This entity contains a list of permitted user types available at the time of user registration.
- WTD_Status: This entity contains data for the possible status of *Willing to donate* field available to the individual user type at the time of registration.

Various views were also created for easier querying purposes:

- Individual_users: This view was created from the USERS entity but extracts only those tuples/rows having the attribute ‘user_type’ as ‘Individual’.

```
select * from users where usertype = "Individual"
```

- Organization_users: This view was created from the USERS entity but extracts only those tuples/rows having the attribute ‘user_type’ as ‘Organization’.

```
select * from users where usertype = "Organization"
```

- Organization_Blood_Inventory: This view was created by joining the USERS table and BLOOD_INVENTORY table.

```
select * from users u, blood_inventory bi where u.ID = bi.ID
```

- Organization_organ_inventory: This view was created by joining the USERS table and ORGAN_INVENTORY table.

```
select * from users, organ_inventory o where users.id = o.ID
```

- Organization_plasma_inventory: This view was created by joining the USERS table and PLASMA_INVENTORY table.

```
select * from users, plasma_inventory p where users.id = p.id
```

Various triggers were also used to ensure referential integrity and data integrity:

- Age_Calc: This trigger calculates the age from the attribute 'date_of_birth' when a new tuple is inserted into the table.

```
CREATE TRIGGER age_calc

AFTER INSERT ON users

FOR EACH ROW

BEGIN

UPDATE users

SET age = CAST (strftime('%Y.%m%d', 'now') - strftime('%Y.%m%d',
new.dob) AS INT)

WHERE ID = new.ID AND new.usertype = "Individual";

END;
```

- Update_Age: This trigger updates the age from the attribute 'date_of_birth' when a a tuple is updated in the table.

```
CREATE TRIGGER update_age

AFTER UPDATE OF dob
```

```
ON users

BEGIN

UPDATE users

SET age = CAST (strftime('%Y.%m%d', 'now') - strftime('%Y.%m%d',
new.dob) AS INT)

WHERE ID = new.ID AND

new.usertype = "Individual";

END;
```

- Update_BI: This trigger fires after a new tuple where the attribute ‘org_type’ value is either ‘Hospital’ or ‘Blood Bank’ is inserted into the USERS table and creates a new tuple in the BLOOD_INVENTORY table with the ID matching the ID of the newly inserted tuple in the USERS table.

```
CREATE TRIGGER update_bi

AFTER INSERT

ON users

FOR EACH ROW

WHEN new.org_type = "Hospital" OR new.org_type = "Blood Bank"

BEGIN

INSERT INTO blood_inventory VALUES (new.ID, 0, 0, 0, 0, 0, 0, 0,
0);

END;
```

- Update_OI: This trigger fires after a new tuple where the attribute ‘org_type’ value is either ‘Hospital’ or ‘Organ Bank’ is inserted into the USERS table and creates a new tuple in the ORGAN_INVENTORY table with the ID matching the ID of the newly inserted tuple in the USERS table.

```
CREATE TRIGGER update_oi

AFTER INSERT

ON users

FOR EACH ROW

WHEN new.org_type = "Hospital" OR new.org_type = "Organ Bank"

BEGIN

INSERT INTO organ_inventory VALUES (new.ID, 0, 0, 0, 0, 0);

END;
```

- Update_PI: This trigger fires after a new tuple where the attribute 'org_type' value is either 'Hospital' or 'Blood Bank' is inserted into the USERS table and creates a new tuple in the PLASMA_INVENTORY table with the ID matching the ID of the newly inserted tuple in the USERS table.

```
CREATE TRIGGER update_pi

AFTER INSERT

ON users

FOR EACH ROW

WHEN new.org_type = "Hospital" OR new.org_type = "Blood Bank"

BEGIN

INSERT INTO plasma_inventory VALUES (new.ID, 0, 0, 0, 0, 0, 0,
0, 0);

END;
```

6.3 GUI STRUCTURE

The mini project's GUI part/front end is designed using Python's Tkinter package which is used for creating Graphical User Interfaces.

In the mini-project's front-end part, I have used a class that inherits from the tkinter.Tk class and various other classes that inherit from the tkinter.ttk.Frame class. The various classes that I have used in my mini-project are:

- Life360: This is the main parent class that inherits from the tkinter.Tk class and starts the application. It displays the main logo and name of the application along with buttons that lead to the registration and login modules.
- The following classes inherit from the tkinter.ttk.Frame class:
  - Register: This class is used for registering a new user and adding his/her details to the database. It additionally validates the various fields like ensuring that a username that is not already taken is used, and email address, phone number and pin-code are valid entries.
  - Login: This class is used for logging-in an existing user. If the credentials entered are valid, then depending upon the user type (Individual or Organization), the class for the appropriate user interface is instantiated. If the credentials entered are incorrect/not valid, then an appropriate message is displayed.
  - IndividualUI: This class is used right after the user who is of 'Individual' user type logins with his/her credentials. A welcome message is printed and the various functions that are available for the user are displayed like:
    - View Profile
    - Log Out
    - Search for near-by donors
    - Search for near-by blood banks
    - Search for near-by organ banks
    - Search for near-by COVID-19 plasma banks
    - Lung Cancer Prediction

- Pneumonia Prediction

- Predict_Pneumonia: This class is used for displaying the module that is used for predicting whether a X-Ray Scan image of the chest uploaded by the user contains pneumonia or not. This prediction is made using a model developed from Convoluted Neural Network Algorithms. The model was made by using a test and train data set consisting of various images belonging to two sets: Abnormal and Normal.
- Predict_LC: This class is used for displaying the module that is used for predicting whether a CT Scan image of the lungs uploaded by the user contains lung cancer or not. This prediction is made using a model developed from Convoluted Neural Network Algorithms. The model was made by using a test and train data set consisting of various images belonging to two sets: Abnormal and Normal.
- SearchOB: This class is used for locating near-by organ banks within the user's city where a particular type of organ as specified by the user is available.
- SearchPlasma: This class is used for locating near-by COVID-19 plasma banks within the user's city where plasma of a particular blood group as specified by the user or is compatible with the blood group type specified by the user is available.
- SearchBB: This class is used for locating near-by blood banks within the user's city where blood of a particular blood group as specified by the user or is compatible with the blood group type specified by the user is available.
- SearchDonors: This class is used for locating near-by users within the user's city who have given their *Willing_to_donate* status as Yes and have a matching or compatible blood group.

- ViewProfileInfo_I: This class is used for displaying the user's personal information and for changing any of the existing data.
- OrganizationUI: This class is used right after the user who is of 'Organization' user type logins with his/her credentials. A welcome message is printed and the various functions that are available for the user are displayed like:
    - View Profile
    - Log Out
    - Update Blood Inventory
    - Update Organ Inventory
    - Update COVID-19 Plasma Inventory
- Update_PI: This class is used for updating the organization's Plasma Inventory. The various fields for the various blood groups are shown along with their corresponding quantity in stock.
- Update_BI: This class is used for updating the organization's Blood Inventory. The various fields for the various blood groups are shown along with their corresponding quantity in stock.
- Update_OI: This class is used for updating the organization's Plasma Inventory. The various fields for the various organs available are shown along with their corresponding count.
- ViewProfileInfo_O This class is used for displaying the organization's profile information and for changing any of the existing data.

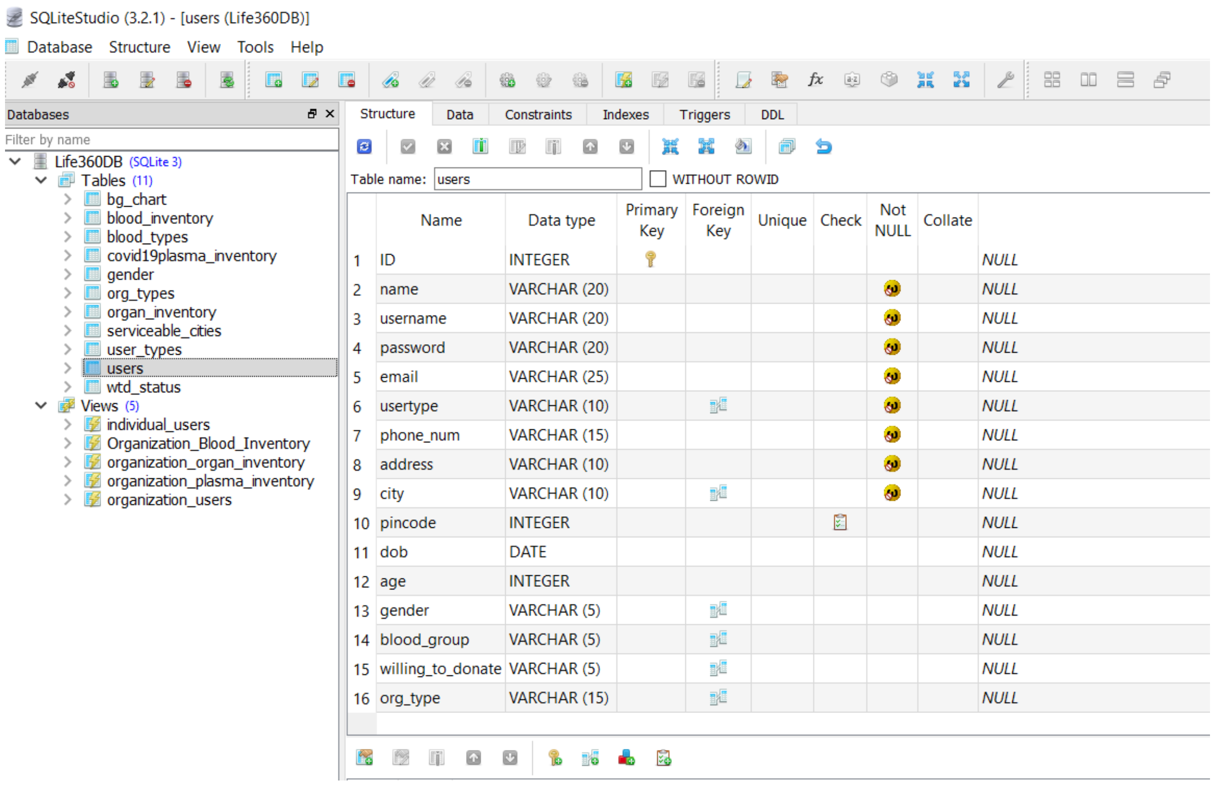
The classes mentioned above are called and instantiated upon appropriate user clicks through buttons. The main class, 'Life360' is used for switching from one class to another by raising the called class Frame object above the previous Frame object.

CHAPTER 7

IMPLEMENTATION

7.1 CREATING THE DATABASE

The database was created using SQLite Studio which use the SQL Lite relational database management system. The various tables were created along with their corresponding attributes and triggers. Appropriate views were also made to help make the querying process easier.


SQLiteStudio (3.2.1) - [users (Life360DB)]
Database Structure View Tools Help
Databases
Filter by name
Structure Data Constraints Indexes Triggers DDL
WITHOUT ROWID

Life360DB (SQLite 3)
- Tables (11)
  - bg_chart
  - blood_inventory
  - blood_types
  - covid19plasma_inventory
  - gender
  - org_types
  - organ_inventory
  - serviceable_cities
  - user_types
  - users
  - wtd_status
- Views (5)
  - individual_users
  - Organization_Blood_Inventory
  - organization_organ_inventory
  - organization_plasma_inventory
  - organization_users

Table name: users

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	
1	ID	INTEGER	✓						*NULL*
2	name	VARCHAR (20)					✓		*NULL*
3	username	VARCHAR (20)					✓		*NULL*
4	password	VARCHAR (20)					✓		*NULL*
5	email	VARCHAR (25)					✓		*NULL*
6	usertype	VARCHAR (10)		✓			✓		*NULL*
7	phone_num	VARCHAR (15)					✓		*NULL*
8	address	VARCHAR (10)					✓		*NULL*
9	city	VARCHAR (10)		✓			✓		*NULL*
10	pincode	INTEGER				✓			*NULL*
11	dob	DATE							*NULL*
12	age	INTEGER							*NULL*
13	gender	VARCHAR (5)		✓					*NULL*
14	blood_group	VARCHAR (5)		✓					*NULL*
15	willing_to_donate	VARCHAR (5)		✓					*NULL*
16	org_type	VARCHAR (15)		✓					*NULL*

Figure 9: Various tables and views in the database

7.2 CONNECTING THE DATABASE TO THE APPLICATION

The database was then connected to the Python application. A variable named *conn* was initialized with the value *None.* This variable is used to establish and close a connection between the database and the application.

```
import tkinter as tk

from tkinter import ttk

import sqlite3

conn = None

def connect():

    global conn

    if not conn:

        conn = sqlite3.connect ("P:\Eclipse\Workspace\Life360\
               db\db.sqlite")

        conn.row_factory = sqlite3.Row


def close():

    if conn:

        conn.close()


connect()

app = tk.Tk()

app.mainloop()

close()
```

The above code segment contains the code that establishes a connection between the database and connection and closes it upon exiting the application. The SQLite3 package aids in the transaction and querying process which are used inside the Python application. The queries and transactions are executed and results from a query are fetched using this package. The following diagram depicts the connection:

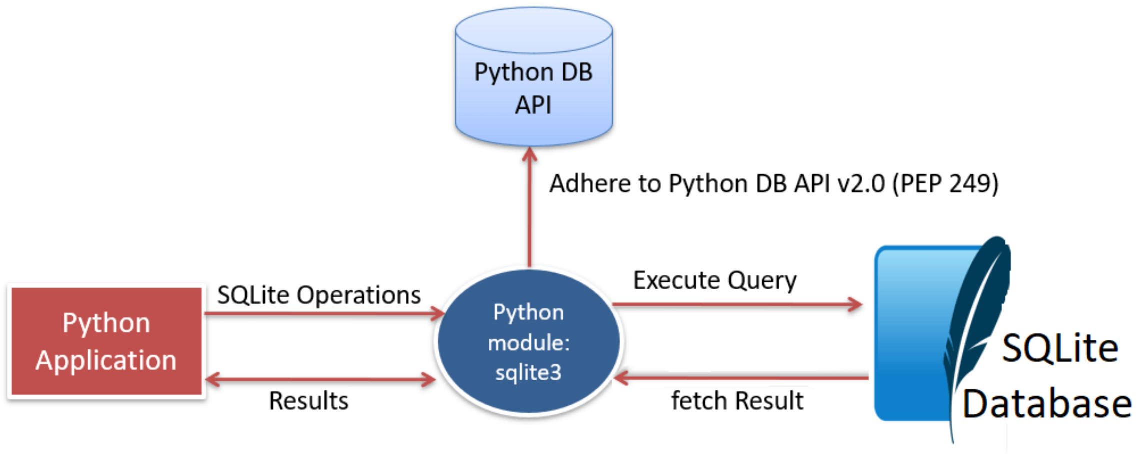


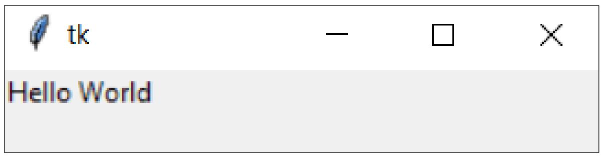
Figure 10: Connecting a Python application to the SQLite Database

7.3 CREATING THE MAIN WINDOW

The main window in the mini-project's front end is created by instantiating a class that inherits from the tkinter.Tk class. This class object forms the main window of the application and other modules used in the application are raised over this window.

```
import tkinter as tk

from tkinter import ttk

class Life360(tk.Tk):

    def __init__(self, *args, **kwargs):

        super().__init__(*args, **kwargs)

        label = ttk.Label(self, text="Hello World")

        label.grid(row=0, column=0)


app = Life360()

app.mainloop()
```

In the above code segment, the object called 'app' of the class 'Life360' is instantiated and is the main window is displayed with a simple window and label as shown below:



Hello World

Figure 11: A simple Tkinter main window

Now, other lines of code to add more widgets and configure various specifications are added to the class.

7.4 DISPLAYING FRAMES OVER THE MAIN WINDOW

Now, this main window we created previously will remain on the screen till we close the application. We can now display other frames over this main window to go through the various other classes that inherit from the tkinter.ttk.Frame class. For this, we make a dictionary called *frames*.

We then instantiate objects of the classes that inherit from the tkinter.ttk.Frame class and place them in the window using a geometry function like *grid*. Then we set the dictionary *frames* to have a key with the class name of the class and value as the object of the class we instantiated. For example, suppose we have a class called *Register* that inherits from the tkinter.ttk.Frame class. We first create an object *registration_frame* and then place it on the main window. Then we set, *frames[Register] = registration_frame* where *Register* is the class which is the key inside the dictionary and *registration_frame* is the object of the *Register* class being passed as a value to the dictionary.

We can now define a class function called *show_frame* that raises the appropriate frame that is passed as argument to the function as:

```
def show_frame(self, container):
        frame = self.frames [container]
        frame.tkraise()
```

We can now display any frame over each-other by calling the above function at appropriate places. For example:

```
import tkinter as tk

from tkinter import ttk

class Life360(tk.Tk):

    def __init__(self, *args, **kwargs):

        super().__init__(*args, **kwargs)

        self.frames = dict()


        mainFrame = ttk.Frame(self)

        mainFrame.grid(row=0, column=0)


        mod1 = Module1(self)

        mod1.grid(row=0, column=0)


        self.frames[Life360] = mainFrame

        self.frames[Module1] = mod1


        label = ttk.Label(mainFrame, text="This is the
                MainFrame")

        label.grid(row=0, column=0)


        button = ttk.Button(mainFrame,

                            text="Change to Module 1",

                            command =
                            lambda:self.show_frame(Module1))

        button.grid(row=1, column=0)


        self.show_frame(Life360)
    def show_frame(self, container):
        frame = self.frames [container]
        frame.tkraise()

class Module1(ttk.Frame):
    def __init__(self, parent, **kwargs):
        super().__init__(parent, **kwargs)
        self.grid_configure(sticky="NSEW")

        label = ttk.Label(self, text="This is Module1 Frame")
        label.grid(row=0, column=0)

        welcome = ttk.Label(self, text="Welcome to Module 1")
        welcome.grid(row=1, column=0)

app = Life360()
app.mainloop()
```

The above code segment has two classes. One is the parent class which inherits from the Tkinter.Tk class called *Life360* and the other one is a class that inherits from the tkinter.ttk.Frame class called *Module1*.

What is essentially happening is that, many frames are being created one over each other inside the main class *Life360* and the the class function *show_frame* decides which frame is displayed at the top. Here, in the above example, the frame object *mod1* is displayed over the frame *mainFrame* when the button is clicked. The output of the above code segment is as shown below.

Figure 11 shows the main window upon running the above code where the frame named *mainFrame* is displayed initially.

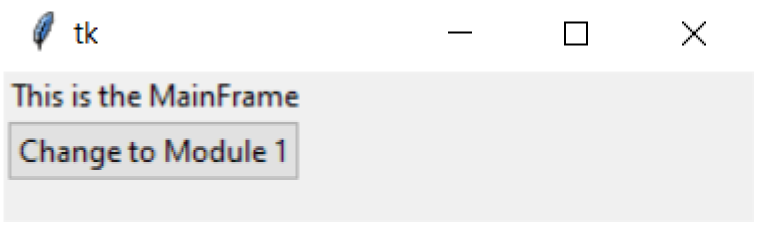


Figure 12: The 'mainFrame' frame in the main window

On clicking the button *Change to Module 1*, the *show_frame* function is invoked and the class, *Module1* is passed as an argument to raise its object, the frame *mod1* above the existing frame, *mainFrame*. This what we see, when we click on the button.

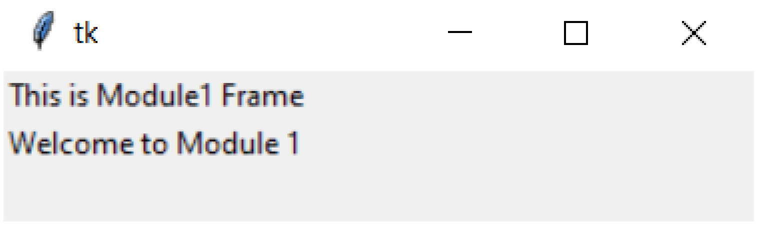


Figure 13: The 'mod1' frame raised over 'mainFrame'

I have used the above concept extensively throughout my project to switch between the various functionalities / modules.

7.5 PROCESSING QUERIES

The mini-project has used several queries throughout the application to SELECT, UPDATE and INSERT values or tuples to and from the database. For processing queries in a python application, we have to initially connect to the database as explained earlier and then write a query for the operation we intend to perform. This query is then executed and any arguments used in the query is also passed to the database for performing the operation.

If the operation is an INSERT or UPDATE operation, then we commit the changes we have made if the operation is successful and if the operation is a SELECT statement, then we fetch the results of the operation.

For performing the above operations, we use the following Python sqlite3 module APIs:

- sqlite3.connect(): To establish a connection the database file. It returns an object of *Connection* type if the connection is successfully established.
- sqlite3.Connection.cursor(): To create a cursor is used to execute queries
- sqlite3.Connection.commit(): To commit the changes the of the current query or transaction.
- sqlite3.Connection.execute(): To execute a query. Parameters can also be given as a tuple
- sqlite3.Connection.row_factory: To return the result from the query as a tuple
- sqlite3.Cursor.fetchone(): Fetches the next row of the query result set
- sqlite3.Cursor.fetchall(): Fetches all the rows of the query result set

The above APIs provided by the Python sqlite3 package have been used extensively for querying purposes.

7.6 BUILDING THE CNN MODEL FOR PREDICTION

For implementing the machine learning part of the mini-project to predict lung cancer and pneumonia, I have used convolutional neural network (CNN) algorithms to construct a model that is developed from a data set of test and train images.

For this, I required Python's Keras and Tensorflow packages which are commonly used for machine learning and deep learning projects. The models developed using this process were used for the prediction of lung cancer and pneumonia in this mini-project.

The initial step is to collect and organize the data set which is used to develop the prediction model. In both pneumonia and lung cancer, we have two types of classifications: normal and abnormal. These two classifications are identified. Then we normalize the dataset and define the CNN by constructing the various layers using a series of built-in functions which are provided by the Keras and Tensorflow libraries. The resulting model at the end of the process is saved as a (.h5) file and is used in the mini-project.

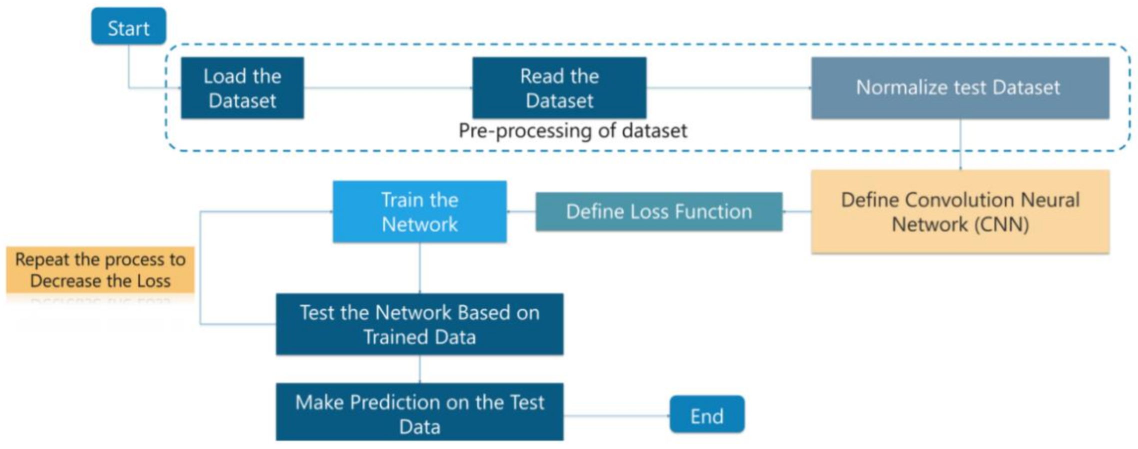

Start
Load the Dataset
Read the Dataset
Normalize test Dataset
Pre-processing of dataset
Define Convolution Neural Network (CNN)
Define Loss Function
Train the Network
Repeat the process to Decrease the Loss
Test the Network Based on Trained Data
Make Prediction on the Test Data
End

Figure 14: The process of building a CNN prediction model

7.7 ADDING STYLES USING TKINTER

The front-end part of the GUI using Tkinter has used the tkinter.ttk module which provides various themed widgets that can be custom styled. The default Tkinter widgets do not allow styling. For this, the mini-project imports the ttk module from the tkinter package. The ttk.Style class is also imported which provides the styling effects for the ttk widgets.

Depending upon the system, ttk provides various themes. By default, the *vista* theme is used in a Windows 10 OS. The various themes available in a Windows 10 OS are:

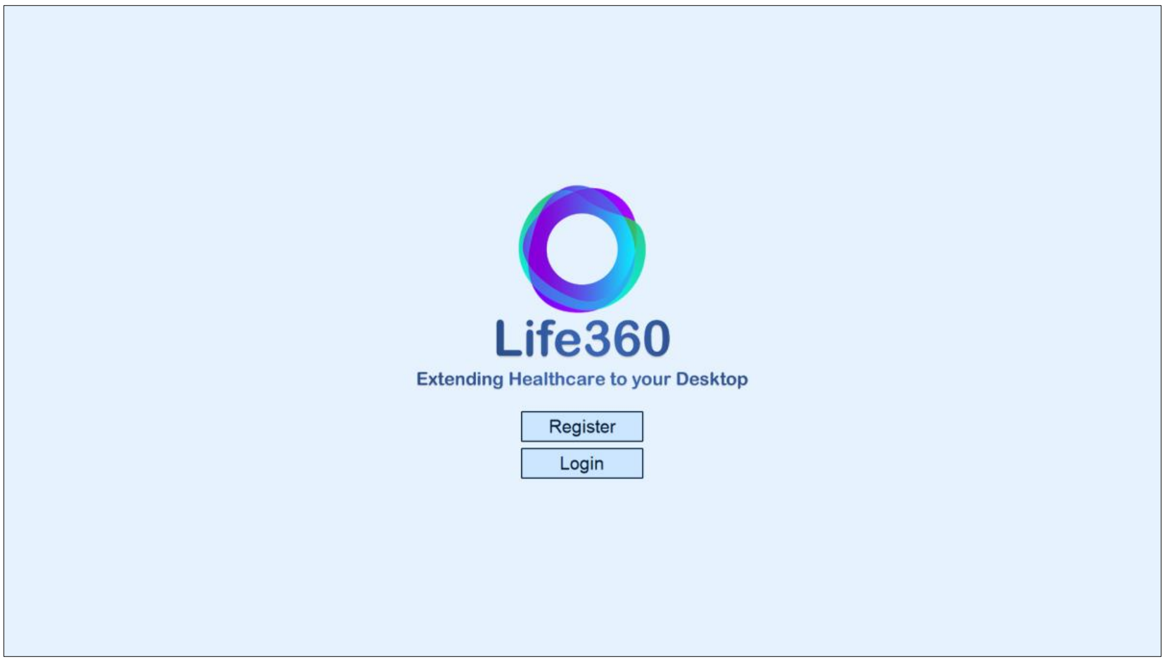
- Windows native
- Clam
- Alt
- Default
- Classic
- Vista
- XP native

I have used the *clam* theme in my mini-project as it is highly customizable compared to other themes. The *ttk.Style.configure* method is used to configure the various widgets.

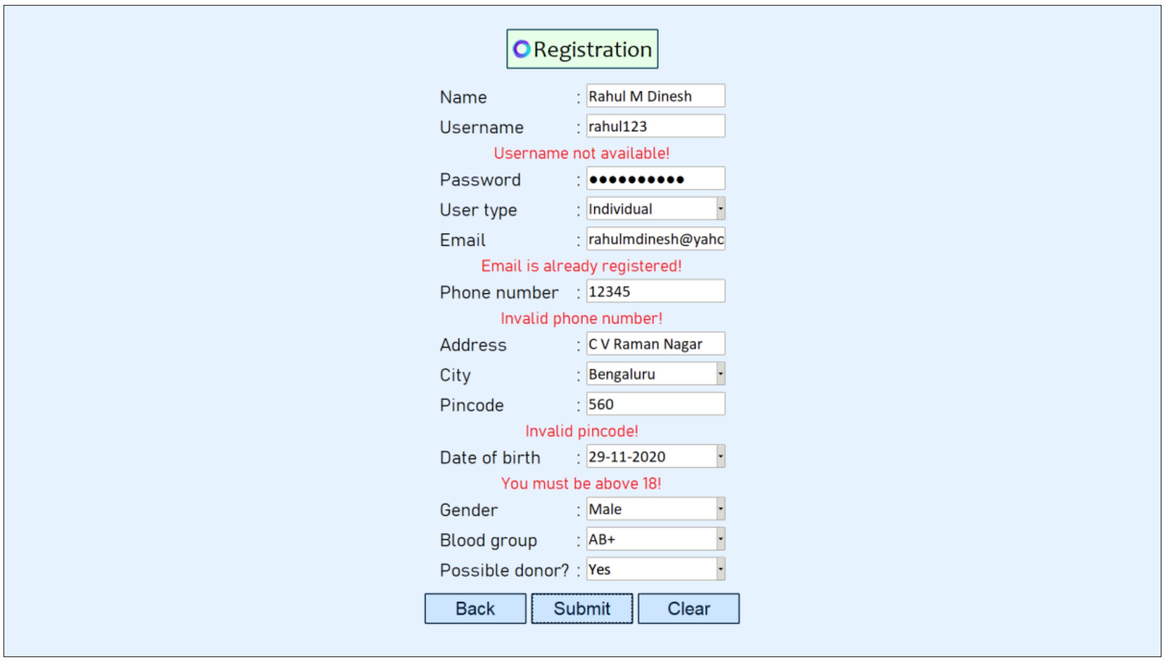
CHAPTER 8

RESULTS

Launch Screen

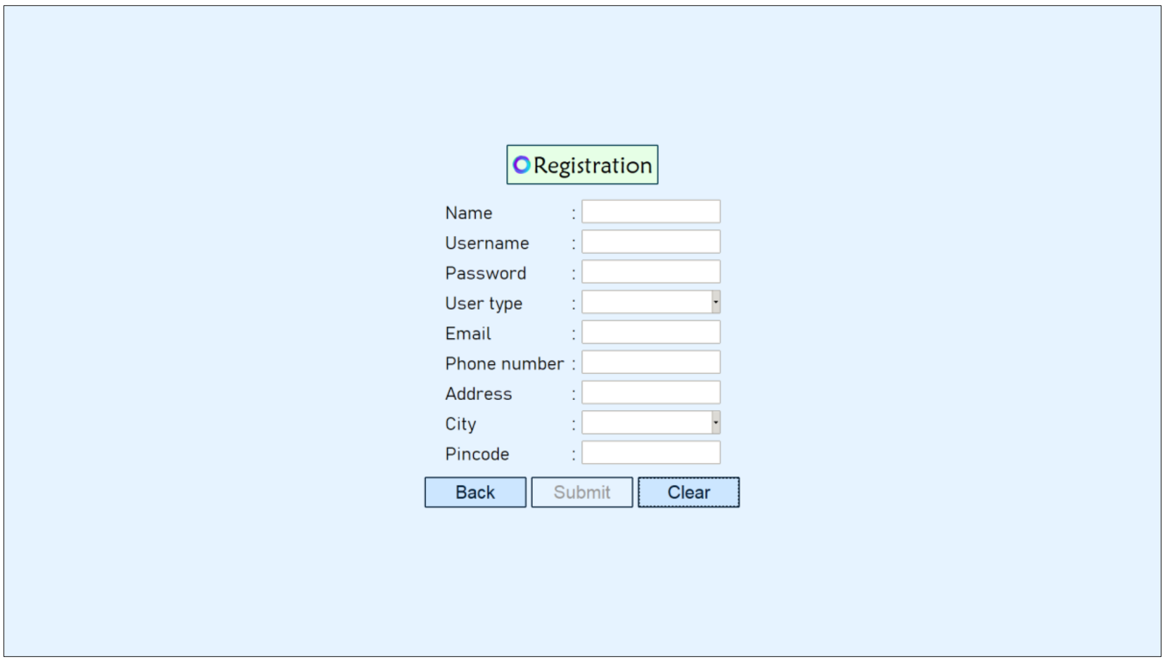


8.1 REGISTERING A NEW USER (VALIDATION)


Registration

Name	:	Rahul M Dinesh
Username	:	rahul123
		Username not available!
Password	:	●●●●●●●●●●
User type	:	Individual
Email	:	rahulmdinesh@yahc
		Email is already registered!
Phone number	:	12345
		Invalid phone number!
Address	:	C V Raman Nagar
City	:	Bengaluru
Pincode	:	560
		Invalid pincode!
Date of birth	:	29-11-2020
		You must be above 18!
Gender	:	Male
Blood group	:	AB+
Possible donor?	:	Yes

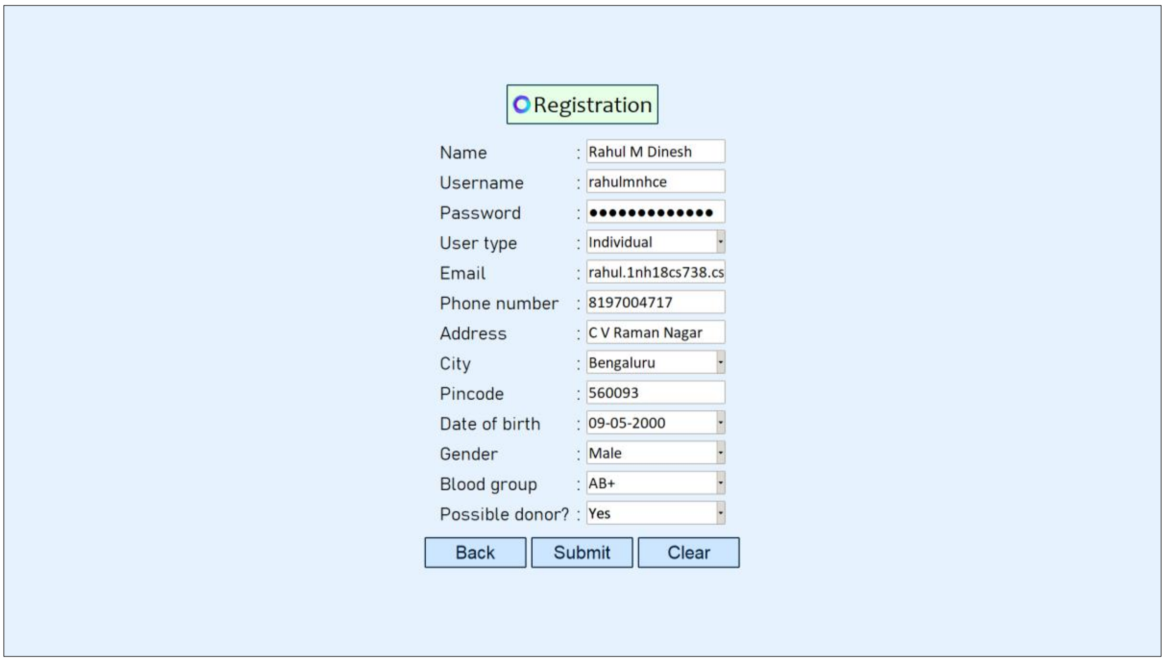
8.2 REGISTERING A NEW USER (INDIVIDUAL)



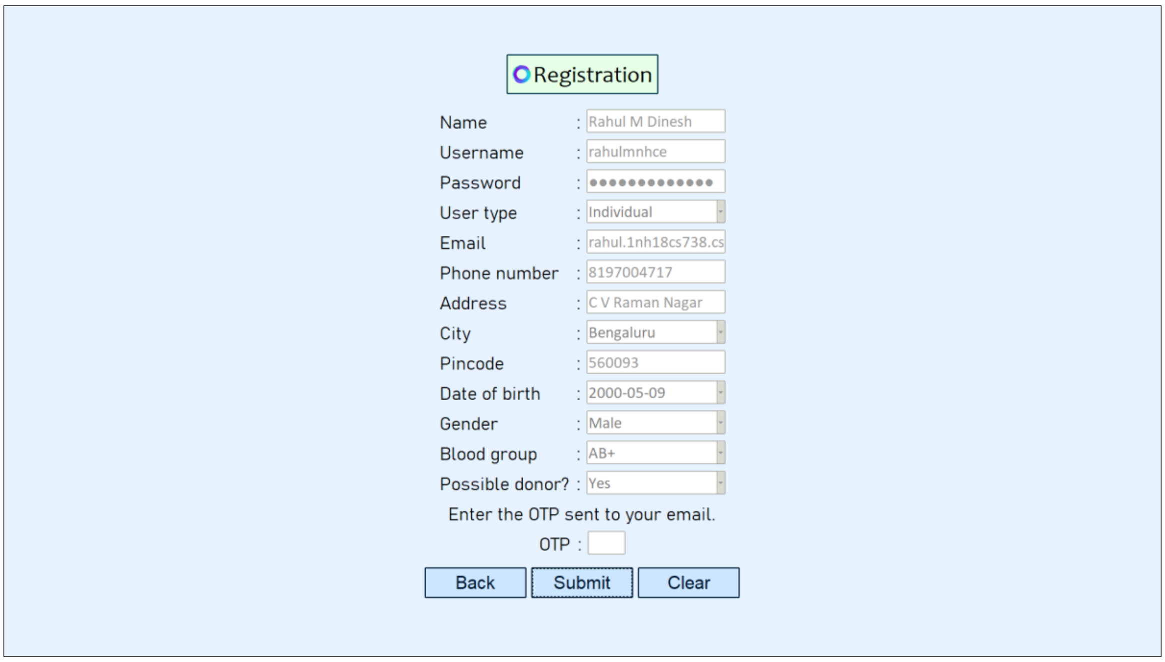
Name	:	
Username	:	
Password	:	
User type	:	
Email	:	
Phone number	:	
Address	:	
City	:	
Pincode	:	

Front end

(Life360 app)



Name	:	Rahul M Dinesh
Username	:	rahulmnhce
Password	:	•••••••••••••
User type	:	Individual
Email	:	rahul.1nh18cs738.cs
Phone number	:	8197004717
Address	:	C V Raman Nagar
City	:	Bengaluru
Pincode	:	560093
Date of birth	:	09-05-2000
Gender	:	Male
Blood group	:	AB+
Possible donor?	:	Yes



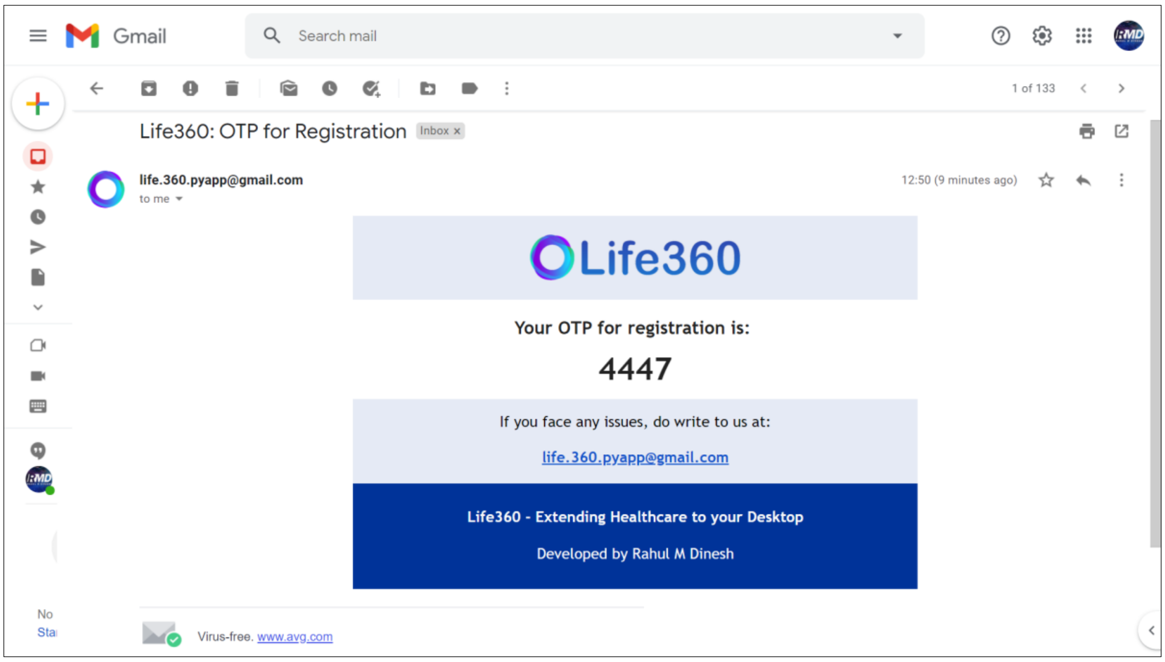
Registration

Name	:	Rahul M Dinesh
Username	:	rahulmnhce
Password	:	●●●●●●●●●●●●●
User type	:	Individual
Email	:	rahul.1nh18cs738.cs
Phone number	:	8197004717
Address	:	C V Raman Nagar
City	:	Bengaluru
Pincode	:	560093
Date of birth	:	2000-05-09
Gender	:	Male
Blood group	:	AB+
Possible donor?	:	Yes

Enter the OTP sent to your email.

OTP :

OTP for Registration sent to user's email



Life360: OTP for Registration Inbox ×

life.360.pyapp@gmail.com
to me

12:50 (9 minutes ago)

Life360

Your OTP for registration is:

4447

If you face any issues, do write to us at:

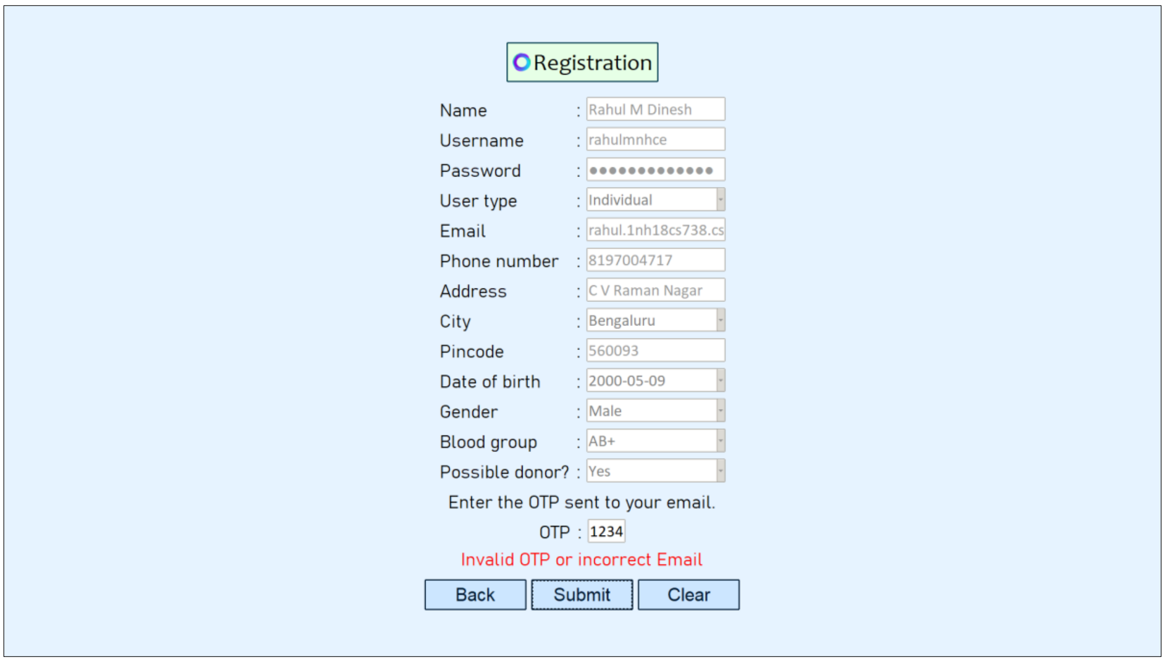
life.360.pyapp@gmail.com

Life360 - Extending Healthcare to your Desktop

Developed by Rahul M Dinesh

Virus-free. www.avg.com

In case, user enters wrong OTP



Registration

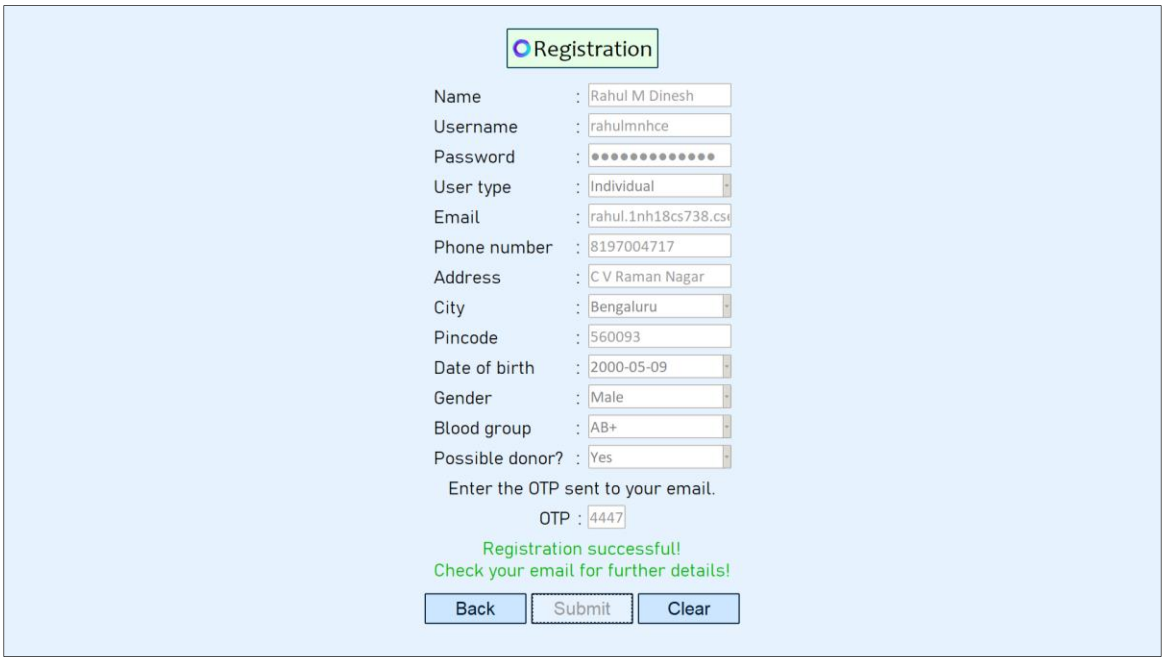
Name	:	Rahul M Dinesh
Username	:	rahulmnhce
Password	:	●●●●●●●●●●●●●
User type	:	Individual
Email	:	rahul.1nh18cs738.cs
Phone number	:	8197004717
Address	:	C V Raman Nagar
City	:	Bengaluru
Pincode	:	560093
Date of birth	:	2000-05-09
Gender	:	Male
Blood group	:	AB+
Possible donor?	:	Yes

Enter the OTP sent to your email.

OTP : 1234

Invalid OTP or incorrect Email

Upon entering the correct OTP, the user is successfully registered



Registration

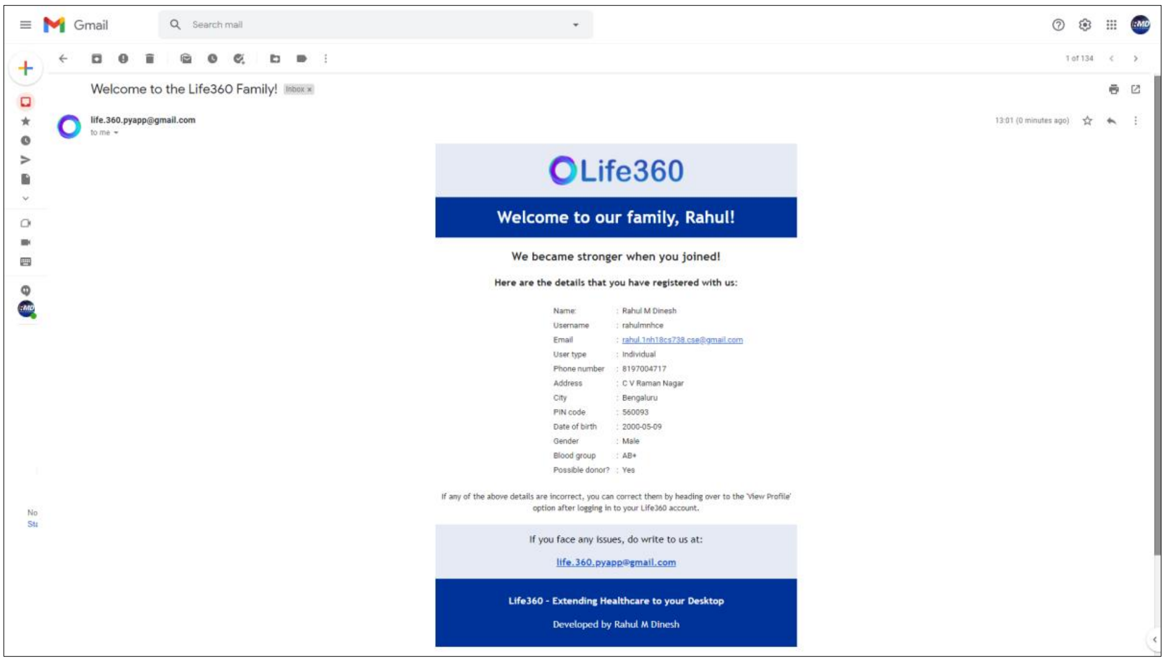
Name	:	Rahul M Dinesh
Username	:	rahulmnhce
Password	:	●●●●●●●●●●●●●
User type	:	Individual
Email	:	rahul.1nh18cs738.cse
Phone number	:	8197004717
Address	:	C V Raman Nagar
City	:	Bengaluru
Pincode	:	560093
Date of birth	:	2000-05-09
Gender	:	Male
Blood group	:	AB+
Possible donor?	:	Yes

Enter the OTP sent to your email.

OTP : 4447

Registration successful!
Check your email for further details!

Welcome mail and user details sent to user's email



Welcome to the Life360 Family!

life.360.pyapp@gmail.com
to me

Welcome to our family, Rahul!

We became stronger when you joined!

Here are the details that you have registered with us:

Name:	: Rahul M Dinesh
Username	: rahulmnhce
Email	: rahul.1nh18cs738.cse@gmail.com
User type	: Individual
Phone number	: 8197004717
Address	: C V Raman Nagar
City	: Bengaluru
PIN code	: 560093
Date of birth	: 2000-05-09
Gender	: Male
Blood group	: AB+
Possible donor?	: Yes

If any of the above details are incorrect, you can correct them by heading over to the 'View Profile' option after logging in to your Life360 account.

If you face any issues, do write to us at:

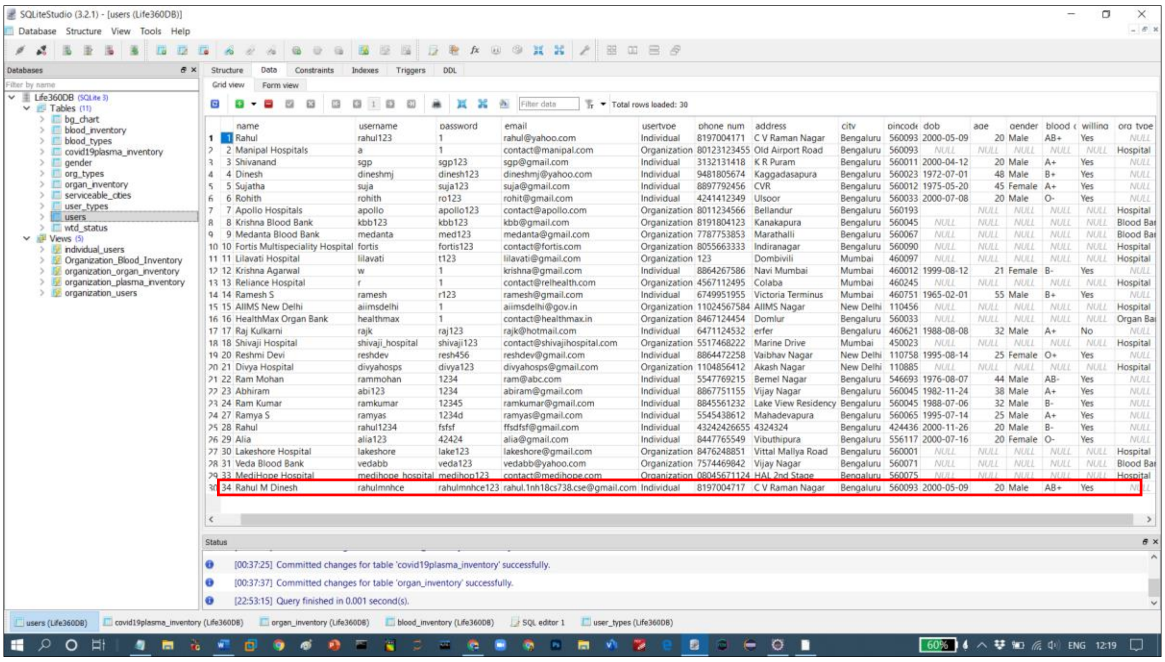
life.360.pyapp@gmail.com

Life360 - Extending Healthcare to your Desktop

Developed by Rahul M Dinesh

Back end

(SQLite Studio)

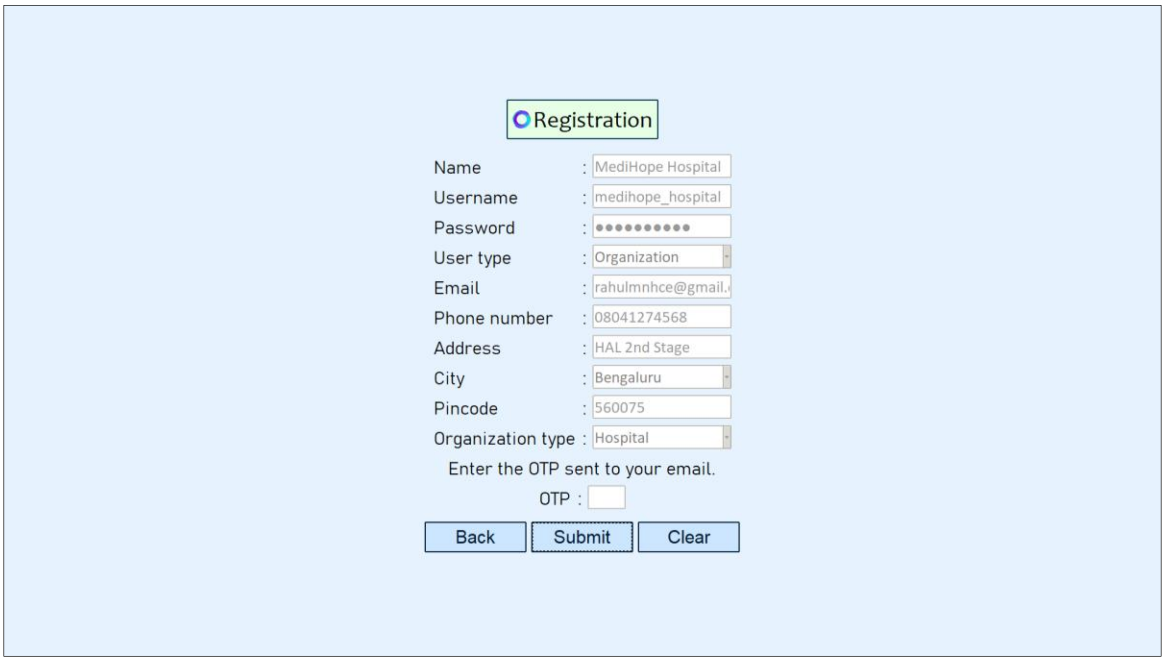

SQLiteStudio (3.2.1) - [users (Life360DB)]
Database Structure View Tools Help
Databases
Filter by name
Life360DB (SQLite 3)
Tables (11)
bg_chart
blood_inventory
blood_types
covid19plasma_inventory
gender
org_types
organ_inventory
serviceable_cities
user_types
users
wtd_status
Views (5)
individual_users
Organization_Blood_Inventory
organization_organ_inventory
organization_plasma_inventory
organization_users
Structure Data Constraints Indexes Triggers DDL
Grid view Form view
Filter data
Total rows loaded: 30
Status
users (Life360DB) covid19plasma_inventory (Life360DB) organ_inventory (Life360DB) blood_inventory (Life360DB) SQL editor 1 user_types (Life360DB)

		name	username	password	email	usertype	phone num	address	city	pincode	dob	age	gender	blood (	willing	org type
1	1	Rahul	rahul123	1	rahul@yahoo.com	Individual	8197004171	C V Raman Nagar	Bengaluru	560093	2000-05-09	20	Male	AB+	Yes	NULL
2	2	Manipal Hospitals	a	1	contact@manipal.com	Organization	80123123455	Old Airport Road	Bengaluru	560093	NULL	NULL	NULL	NULL	NULL	Hospital
3	3	Shivanand	sgp	sgp123	sgp@gmail.com	Individual	3132131418	K R Puram	Bengaluru	560011	2000-04-12	20	Male	A+	Yes	NULL
4	4	Dinesh	dineshmj	dinesh123	dineshmj@yahoo.com	Individual	9481805674	Kaggadasapura	Bengaluru	560023	1972-07-01	48	Male	B+	Yes	NULL
5	5	Sujatha	suja	suja123	suja@gmail.com	Individual	8897792456	CVR	Bengaluru	560012	1975-05-20	45	Female	A+	Yes	NULL
6	6	Rohith	rohith	ro123	rohit@gmail.com	Individual	4241412349	Ulsoor	Bengaluru	560033	2000-07-08	20	Male	O-	Yes	NULL
7	7	Apollo Hospitals	apollo	apollo123	contact@apollo.com	Organization	8011234566	Bellandur	Bengaluru	560193		NULL	NULL	NULL	NULL	Hospital
8	8	Krishna Blood Bank	kbb123	kbb123	kbb@gmail.com	Organization	8191804123	Kanakapura	Bengaluru	560045	NULL	NULL	NULL	NULL	NULL	Blood Ba
9	9	Medanta Blood Bank	medanta	med123	medanta@gmail.com	Organization	7787753853	Marathalli	Bengaluru	560067	NULL	NULL	NULL	NULL	NULL	Blood Ba
10	10	Fortis Multispeciality Hospital	fortis	fortis123	contact@fortis.com	Organization	8055663333	Indiranagar	Bengaluru	560090	NULL	NULL	NULL	NULL	NULL	Hospital
11	11	Lilavati Hospital	lilavati	t123	lilavati@gmail.com	Organization	123	Dombivili	Mumbai	460097	NULL	NULL	NULL	NULL	NULL	Hospital
12	12	Krishna Agarwal	w	1	krishna@gmail.com	Individual	8864267586	Navi Mumbai	Mumbai	460012	1999-08-12	21	Female	B-	Yes	NULL
13	13	Reliance Hospital	r	1	contact@relhealth.com	Organization	4567112495	Colaba	Mumbai	460245	NULL	NULL	NULL	NULL	NULL	Hospital
14	14	Ramesh S	ramesh	r123	ramesh@gmail.com	Individual	6749951955	Victoria Terminus	Mumbai	460751	1965-02-01	55	Male	B+	Yes	NULL
15	15	AIIMS New Delhi	aiimsdelhi	1	aiimsdelhi@gov.in	Organization	11024567584	AIIMS Nagar	New Delhi	110456	NULL	NULL	NULL	NULL	NULL	Hospital
16	16	HealthMax Organ Bank	healthmax	1	contact@healthmax.in	Organization	8467124454	Domlur	Bengaluru	560033	NULL	NULL	NULL	NULL	NULL	Organ Ba
17	17	Raj Kulkarni	rajk	raj123	rajk@hotmail.com	Individual	6471124532	erfer	Bengaluru	460621	1988-08-08	32	Male	A+	No	NULL
18	18	Shivaji Hospital	shivaji_hospital	shivaji123	contact@shivajihospital.com	Organization	5517468222	Marine Drive	Mumbai	450023	NULL	NULL	NULL	NULL	NULL	Hospital
19	20	Reshmi Devi	reshdev	resh456	reshdev@gmail.com	Individual	8864472258	Vaibhav Nagar	New Delhi	110758	1995-08-14	25	Female	O+	Yes	NULL
20	21	Divya Hospital	divyahosps	divya123	divyahosps@gmail.com	Organization	1104856412	Akash Nagar	New Delhi	110885	NULL	NULL	NULL	NULL	NULL	Hospital
21	22	Ram Mohan	rammohan	1234	ram@abc.com	Individual	5547769215	Bemel Nagar	Bengaluru	546693	1976-08-07	44	Male	AB-	Yes	NULL
22	23	Abhiram	abi123	1234	abiram@gmail.com	Individual	8867751155	Vijay Nagar	Bengaluru	560045	1982-11-24	38	Male	A+	Yes	NULL
23	24	Ram Kumar	ramkumar	12345	ramkumar@gmail.com	Individual	8845561232	Lake View Residency	Bengaluru	560045	1988-07-06	32	Male	B-	Yes	NULL
24	27	Ramya S	ramyas	1234d	ramyas@gmail.com	Individual	5545438612	Mahadevapura	Bengaluru	560065	1995-07-14	25	Male	A+	Yes	NULL
25	28	Rahul	rahul1234	fsfsf	ffsdfsf@gmail.com	Individual	43242426655	4324324	Bengaluru	424436	2000-11-26	20	Male	B-	Yes	NULL
26	29	Alia	alia123	42424	alia@gmail.com	Individual	8447765549	Vibuthipura	Bengaluru	556117	2000-07-16	20	Female	O-	Yes	NULL
27	30	Lakeshore Hospital	lakeshore	lake123	lakeshore@gmail.com	Organization	8476248851	Vittal Mallya Road	Bengaluru	560001	NULL	NULL	NULL	NULL	NULL	Hospital
28	31	Veda Blood Bank	vedabb	veda123	vedabb@yahoo.com	Organization	7574469842	Vijay Nagar	Bengaluru	560071	NULL	NULL	NULL	NULL	NULL	Blood Ba
29	33	MediHope Hospital	medihope_hospital	medihop123	contact@medihope.com	Organization	08045671124	HAL 2nd Stage	Bengaluru	560075	NULL	NULL	NULL	NULL	NULL	Hospital
30	34	Rahul M Dinesh	rahulmnhce	rahulmnhce123	rahul.1nh18cs738.cse@gmail.com	Individual	8197004717	C V Raman Nagar	Bengaluru	560093	2000-05-09	20	Male	AB+	Yes	NULL

```
[00:37:25] Committed changes for table 'covid19plasma_inventory' successfully.
[00:37:37] Committed changes for table 'organ_inventory' successfully.
[22:53:15] Query finished in 0.001 second(s).
```

8.3 REGISTERING A NEW USER (ORGANIZATION)

Front end

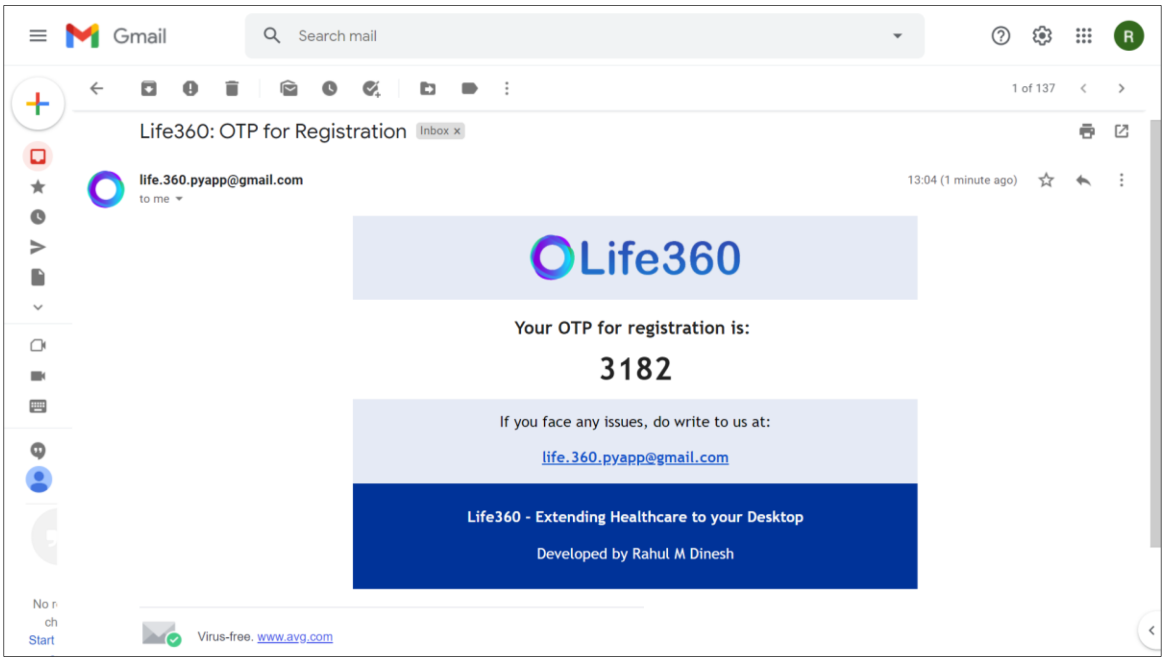


Name	:	MediHope Hospital
Username	:	medihope_hospital
Password	:	••••••••••
User type	:	Organization
Email	:	rahulmnhce@gmail.
Phone number	:	08041274568
Address	:	HAL 2nd Stage
City	:	Bengaluru
Pincode	:	560075
Organization type	:	Hospital

Enter the OTP sent to your email.

OTP :

OTP for Registration sent to user's email



Life360: OTP for Registration

life.360.pyapp@gmail.com
to me

13:04 (1 minute ago)

Life360

Your OTP for registration is:

3182

If you face any issues, do write to us at:

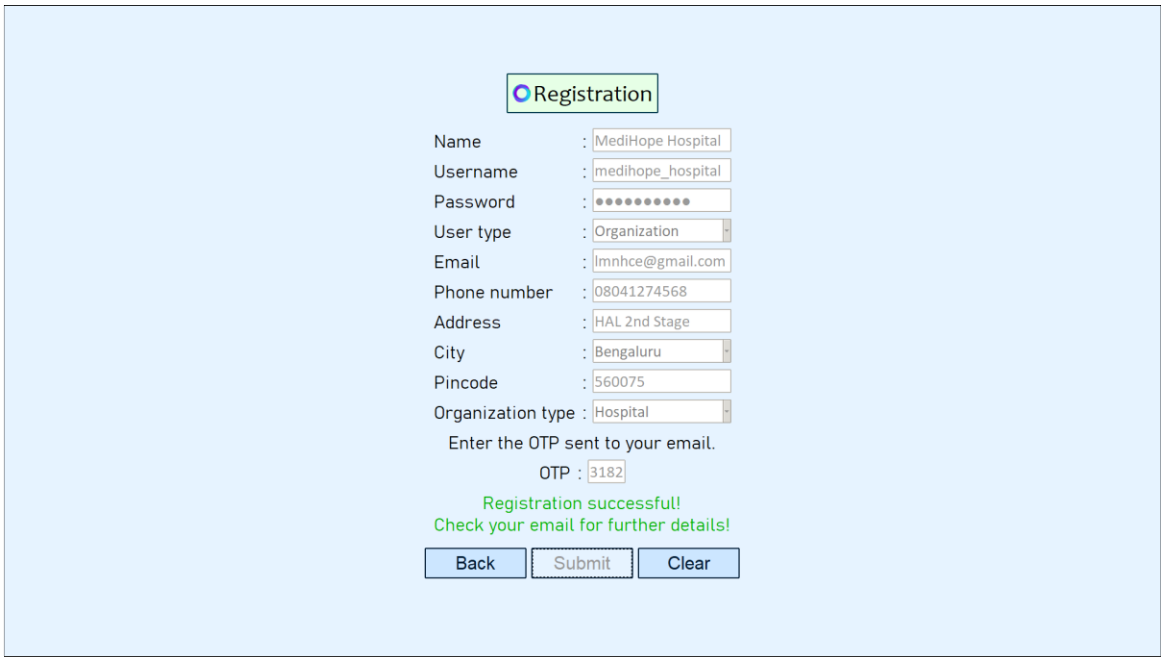
life.360.pyapp@gmail.com

Life360 - Extending Healthcare to your Desktop

Developed by Rahul M Dinesh

Virus-free. www.avg.com

User successfully registered


Registration

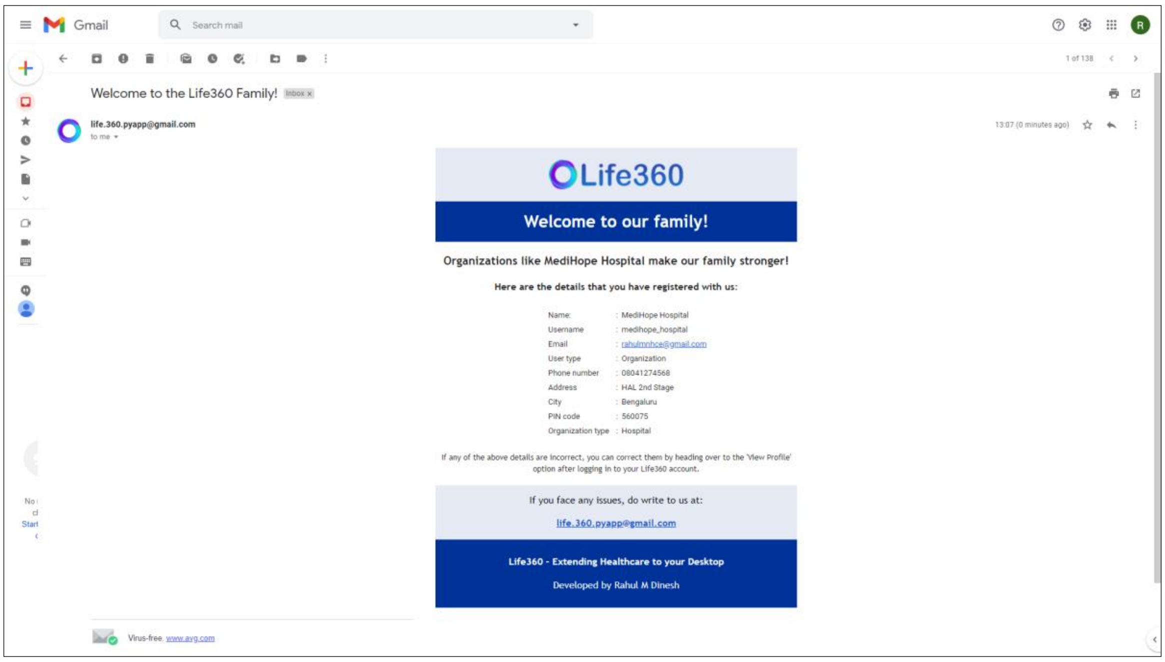
Name	:	MediHope Hospital
Username	:	medihope_hospital
Password	:	••••••••••
User type	:	Organization
Email	:	lmnhce@gmail.com
Phone number	:	08041274568
Address	:	HAL 2nd Stage
City	:	Bengaluru
Pincode	:	560075
Organization type	:	Hospital

Enter the OTP sent to your email.

OTP : 3182

Registration successful!
Check your email for further details!

Welcome mail and organization details sent to user's email



Welcome to the Life360 Family! Inbox

life.360.pyapp@gmail.com
to me

13:07 (0 minutes ago)

Life360

Welcome to our family!

Organizations like MediHope Hospital make our family stronger!

Here are the details that you have registered with us:

Name:	:	MediHope Hospital
Username	:	medihope_hospital
Email	:	rahulmnhce@gmail.com
User type	:	Organization
Phone number	:	08041274568
Address	:	HAL 2nd Stage
City	:	Bengaluru
PIN code	:	560075
Organization type	:	Hospital

If any of the above details are incorrect, you can correct them by heading over to the 'View Profile' option after logging in to your Life360 account.

If you face any issues, do write to us at:

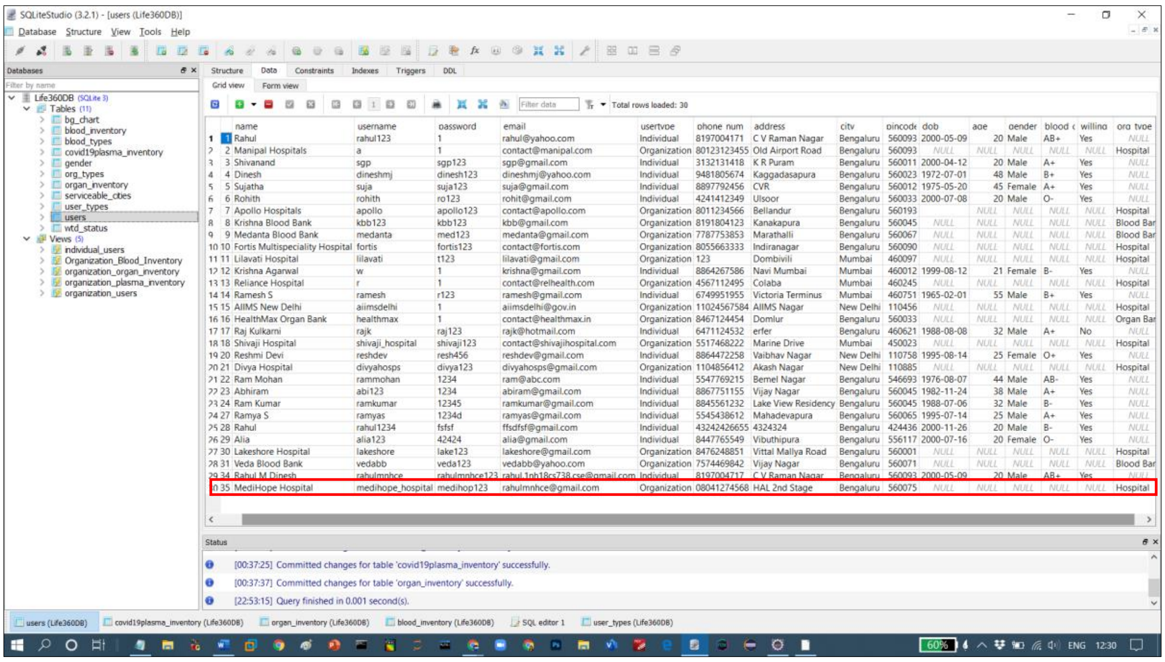
life.360.pyapp@gmail.com

Life360 - Extending Healthcare to your Desktop

Developed by Rahul M Dinesh

Virus-free. www.avg.com

Back end



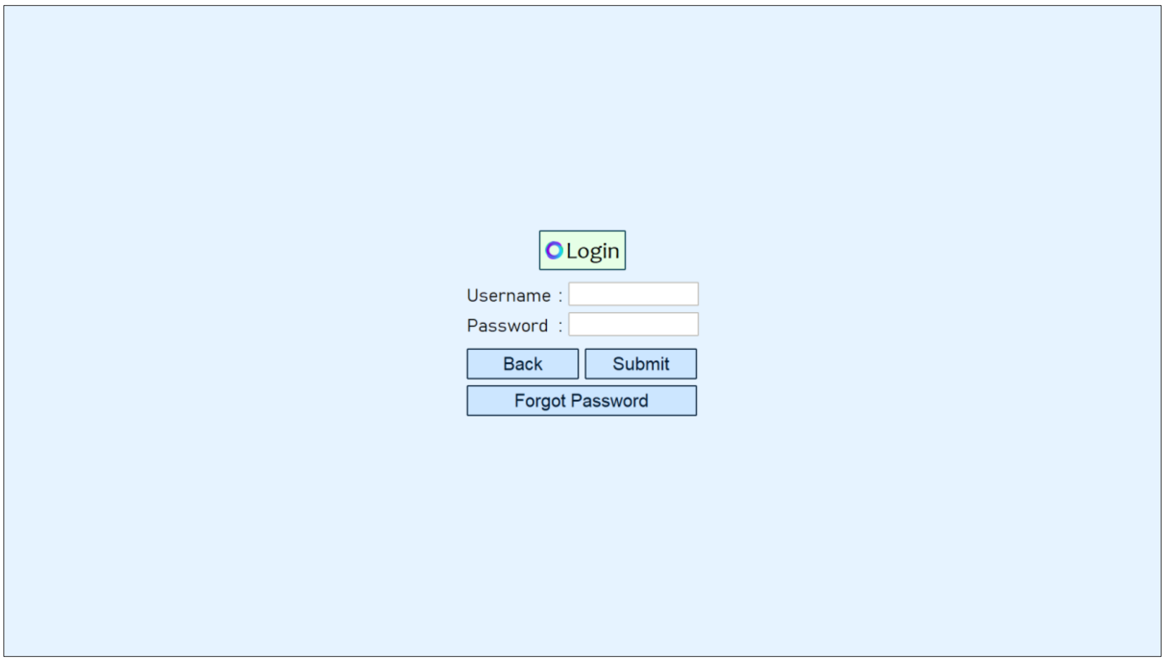
Total rows loaded: 30

		name	username	password	email	usertype	phone num	address	city	pincode	dob	age	gender	blood	willing	org type
1	1	Rahul	rahul123	1	rahul@yahoo.com	Individual	8197004171	C V Raman Nagar	Bengaluru	560093	2000-05-09	20	Male	AB+	Yes	NULL
2	2	Manipal Hospitals	a	1	contact@manipal.com	Organization	80123123455	Old Airport Road	Bengaluru	560093	NULL	NULL	NULL	NULL	NULL	Hospital
3	3	Shivanand	sgp	sgp123	sgp@gmail.com	Individual	3132131418	K R Puram	Bengaluru	560011	2000-04-12	20	Male	A+	Yes	NULL
4	4	Dinesh	dineshmj	dinesh123	dineshmj@yahoo.com	Individual	9481805674	Kaggadasapura	Bengaluru	560023	1972-07-01	48	Male	B+	Yes	NULL
5	5	Sujatha	suja	suja123	suja@gmail.com	Individual	8897792456	CVR	Bengaluru	560012	1975-05-20	45	Female	A+	Yes	NULL
6	6	Rohith	rohith	ro123	rohit@gmail.com	Individual	4241412349	Ulsoor	Bengaluru	560033	2000-07-08	20	Male	O-	Yes	NULL
7	7	Apollo Hospitals	apollo	apollo123	contact@apollo.com	Organization	8011234566	Bellandur	Bengaluru	560193		NULL	NULL	NULL	NULL	Hospital
8	8	Krishna Blood Bank	kbb123	kbb123	kbb@gmail.com	Organization	8191804123	Kanakapura	Bengaluru	560045	NULL	NULL	NULL	NULL	NULL	Blood Bar
9	9	Medanta Blood Bank	medanta	med123	medanta@gmail.com	Organization	7787753853	Marathalli	Bengaluru	560067	NULL	NULL	NULL	NULL	NULL	Blood Bar
10	10	Fortis Multispeciality Hospital	fortis	fortis123	contact@fortis.com	Organization	8055663333	Indiranagar	Bengaluru	560090	NULL	NULL	NULL	NULL	NULL	Hospital
11	11	Lilavati Hospital	lilavati	t123	lilavati@gmail.com	Organization	123	Dombivili	Mumbai	460097	NULL	NULL	NULL	NULL	NULL	Hospital
12	12	Krishna Agarwal	w	1	krishna@gmail.com	Individual	8864267586	Navi Mumbai	Mumbai	460012	1999-08-12	21	Female	B-	Yes	NULL
13	13	Reliance Hospital	r	1	contact@relhealth.com	Organization	4567112495	Colaba	Mumbai	460245	NULL	NULL	NULL	NULL	NULL	Hospital
14	14	Ramesh S	ramesh	r123	ramesh@gmail.com	Individual	6749951955	Victoria Terminus	Mumbai	460751	1965-02-01	55	Male	B+	Yes	NULL
15	15	AIIMS New Delhi	aiimsdelhi	1	aiimsdelhi@gov.in	Organization	11024567584	AIIMS Nagar	New Delhi	110456	NULL	NULL	NULL	NULL	NULL	Hospital
16	16	HealthMax Organ Bank	healthmax	1	contact@healthmax.in	Organization	8467124454	Domlur	Bengaluru	560033	NULL	NULL	NULL	NULL	NULL	Organ Bar
17	17	Raj Kulkarni	rajk	raj123	rajk@hotmail.com	Individual	6471124532	erfer	Bengaluru	460621	1988-08-08	32	Male	A+	No	NULL
18	18	Shivaji Hospital	shivaji_hospital	shivaji123	contact@shivajihospital.com	Organization	5517468222	Marine Drive	Mumbai	450023	NULL	NULL	NULL	NULL	NULL	Hospital
19	20	Reshmi Devi	reshdev	resh456	reshdev@gmail.com	Individual	8864472258	Vaibhav Nagar	New Delhi	110758	1995-08-14	25	Female	O+	Yes	NULL
20	21	Divya Hospital	divyahosps	divya123	divyahosps@gmail.com	Organization	1104856412	Akash Nagar	New Delhi	110885	NULL	NULL	NULL	NULL	NULL	Hospital
21	22	Ram Mohan	rammohan	1234	ram@abc.com	Individual	5547769215	Bemel Nagar	Bengaluru	546693	1976-08-07	44	Male	AB-	Yes	NULL
22	23	Abhiram	abi123	1234	abiram@gmail.com	Individual	8867751155	Vijay Nagar	Bengaluru	560045	1982-11-24	38	Male	A+	Yes	NULL
23	24	Ram Kumar	ramkumar	12345	ramkumar@gmail.com	Individual	8845561232	Lake View Residency	Bengaluru	560045	1988-07-06	32	Male	B-	Yes	NULL
24	27	Ramya S	ramyas	1234d	ramyas@gmail.com	Individual	5545438612	Mahadevapura	Bengaluru	560065	1995-07-14	25	Male	A+	Yes	NULL
25	28	Rahul	rahul1234	fsfsf	ffsdfsf@gmail.com	Individual	43242426655	4324324	Bengaluru	424436	2000-11-26	20	Male	B-	Yes	NULL
26	29	Alia	alia123	42424	alia@gmail.com	Individual	8447765549	Vibuthipura	Bengaluru	556117	2000-07-16	20	Female	O-	Yes	NULL
27	30	Lakeshore Hospital	lakeshore	lake123	lakeshore@gmail.com	Organization	8476248851	Vittal Mallya Road	Bengaluru	560001	NULL	NULL	NULL	NULL	NULL	Hospital
28	31	Veda Blood Bank	vedabb	veda123	vedabb@yahoo.com	Organization	7574469842	Vijay Nagar	Bengaluru	560071	NULL	NULL	NULL	NULL	NULL	Blood Bar
29	34	Rahul M Dinesh	rahulmnhce	rahulmnhce123	rahul.1nh18cs738.cse@gmail.com	Individual	8197004717	C V Raman Nagar	Bengaluru	560093	2000-05-09	20	Male	AB+	Yes	NULL
30	35	MediHope Hospital	medihope_hospital	medihop123	rahulmnhce@gmail.com	Organization	08041274568	HAL 2nd Stage	Bengaluru	560075	NULL	NULL	NULL	NULL	NULL	Hospital

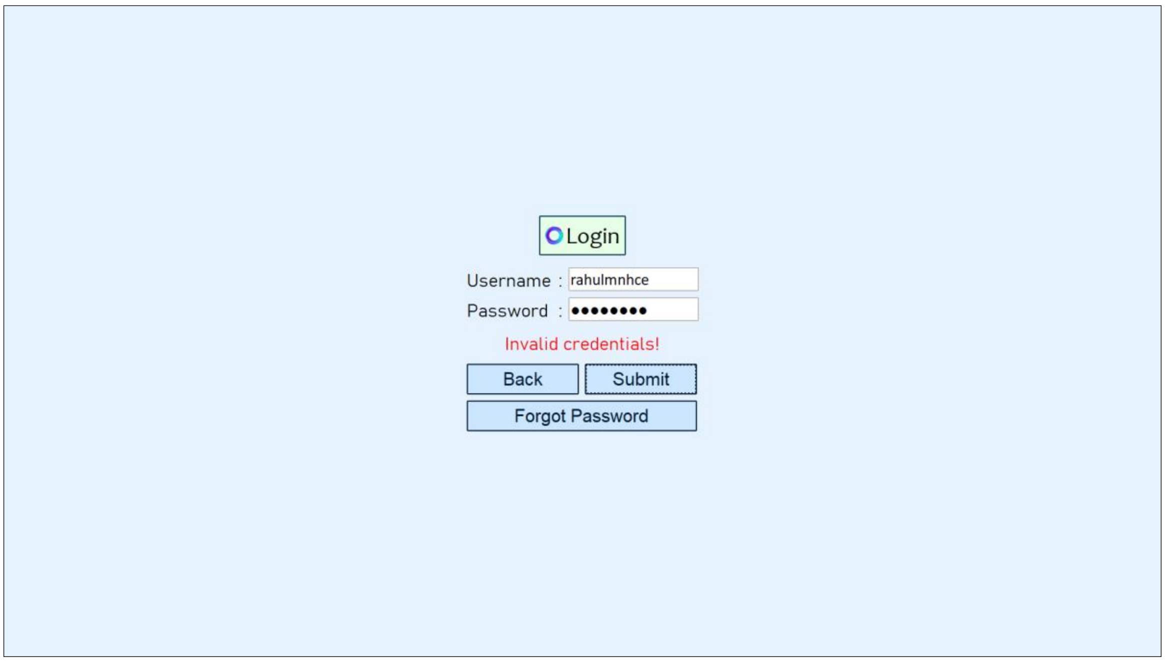
```
[00:37:25] Committed changes for table 'covid19plasma_inventory' successfully.
[00:37:37] Committed changes for table 'organ_inventory' successfully.
[22:53:15] Query finished in 0.001 second(s).
```

8.4 LOGGING IN (INDIVIDUAL)

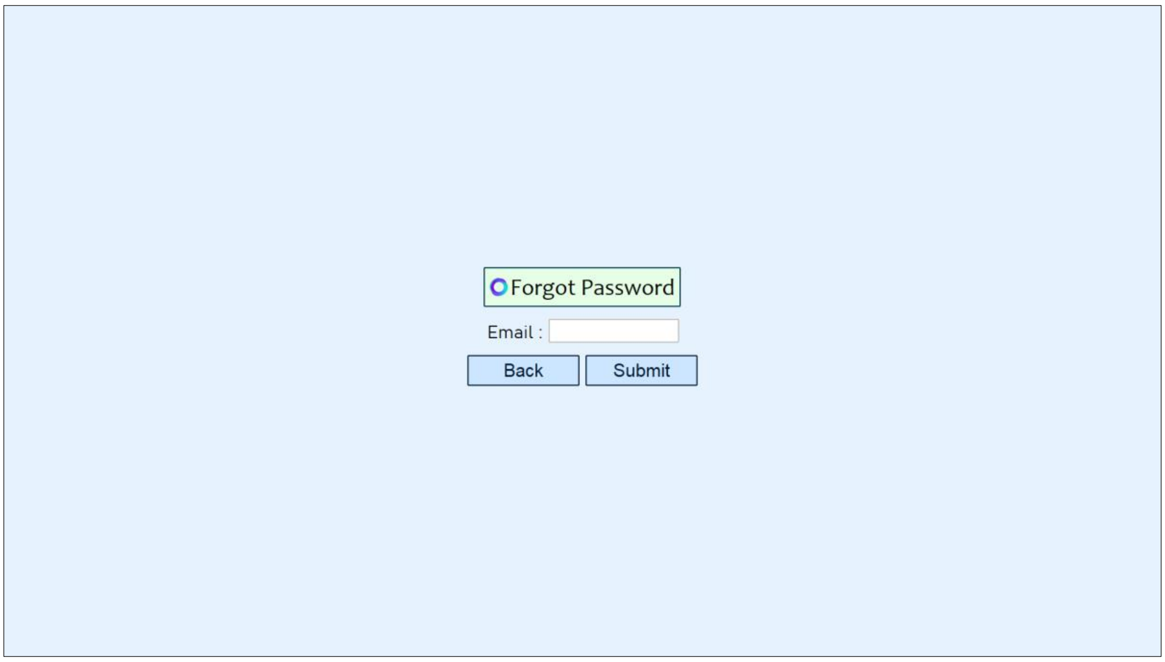
Login screen



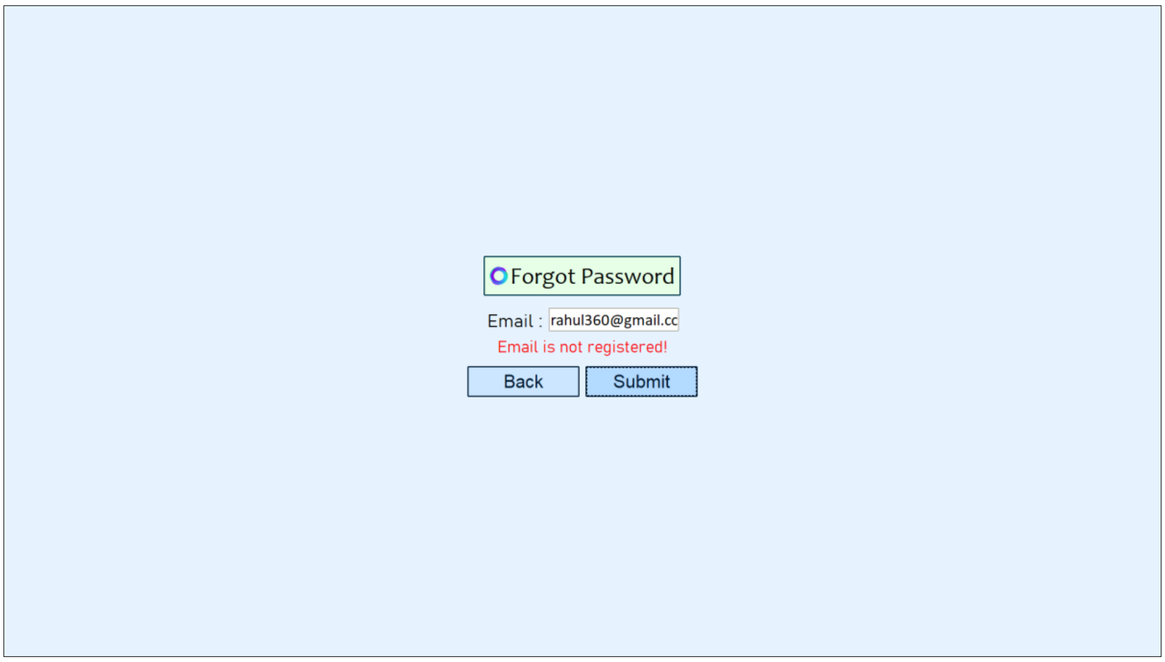
Logging in with invalid credentials



If a user forgets his/her password, then he/she can reset their password



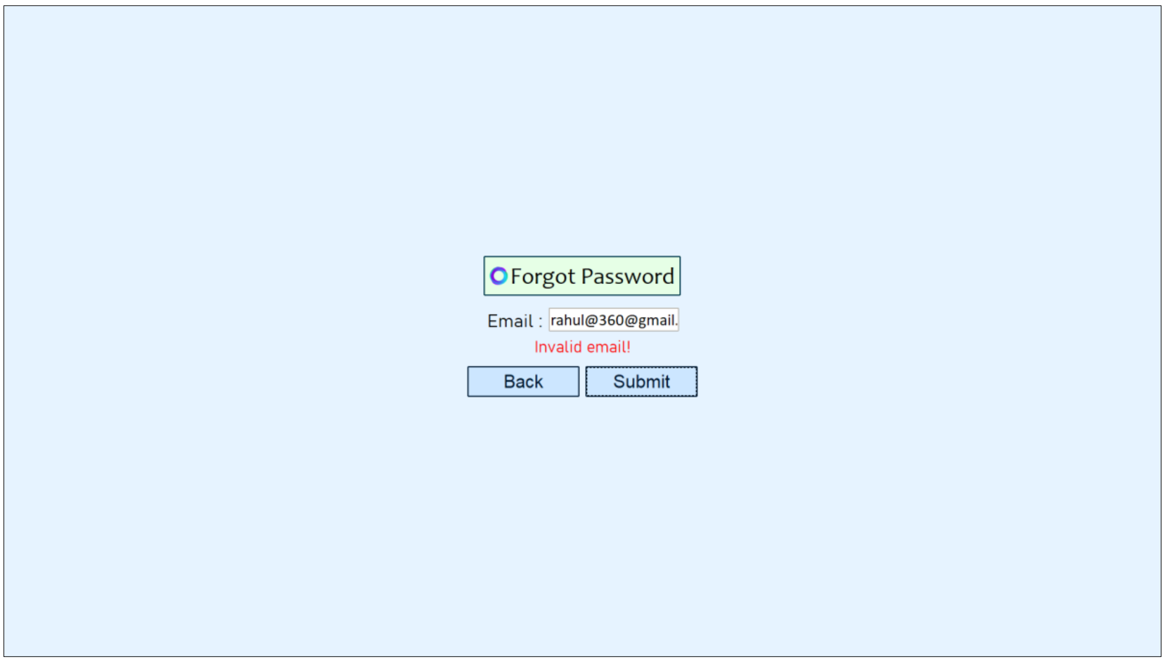
If an unregistered email address is entered



rahul360@gmail.cc

Email is not registered!

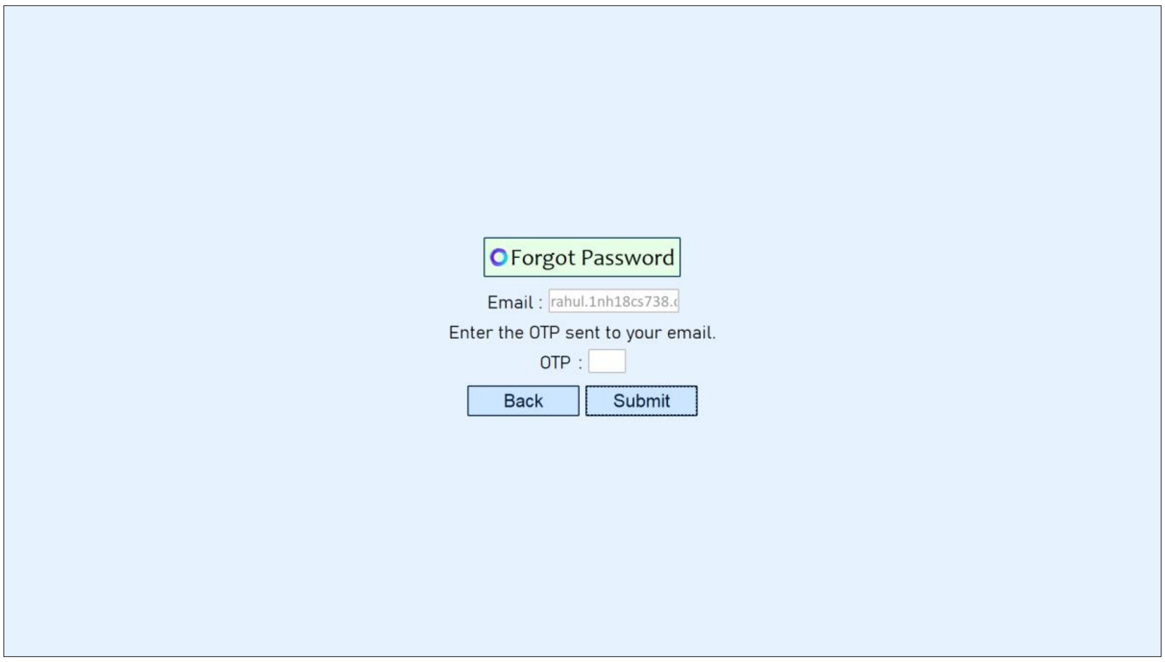
If an invalid email address is entered



rahul@360@gmail.

Invalid email!

An OTP will be sent to the registered email address

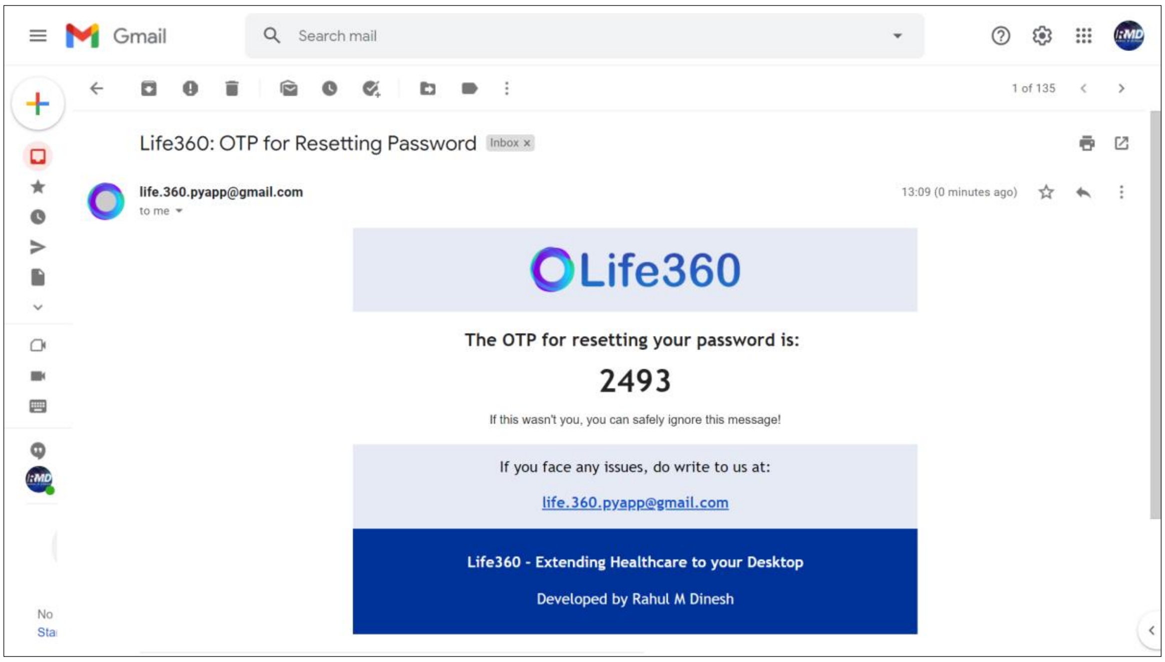


Email : rahul.1nh18cs738.

Enter the OTP sent to your email.

OTP :

OTP for resetting user password received at user's registered email address



Life360: OTP for Resetting Password

life.360.pyapp@gmail.com

13:09 (0 minutes ago)

Life360

The OTP for resetting your password is:

2493

If this wasn't you, you can safely ignore this message!

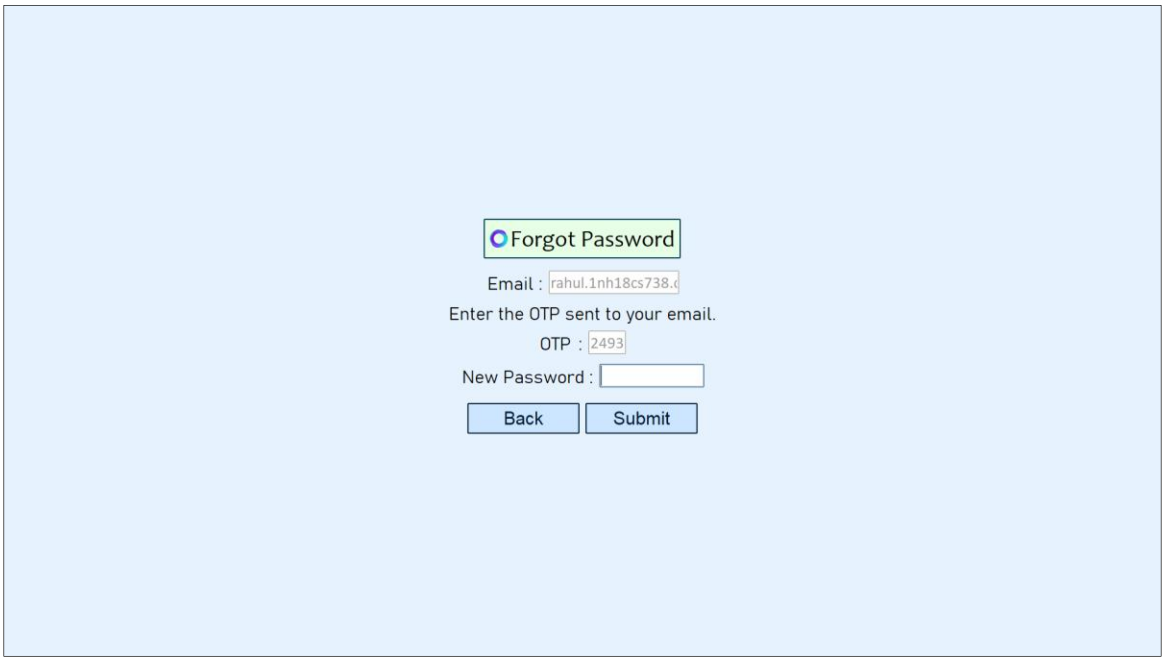
If you face any issues, do write to us at:

life.360.pyapp@gmail.com

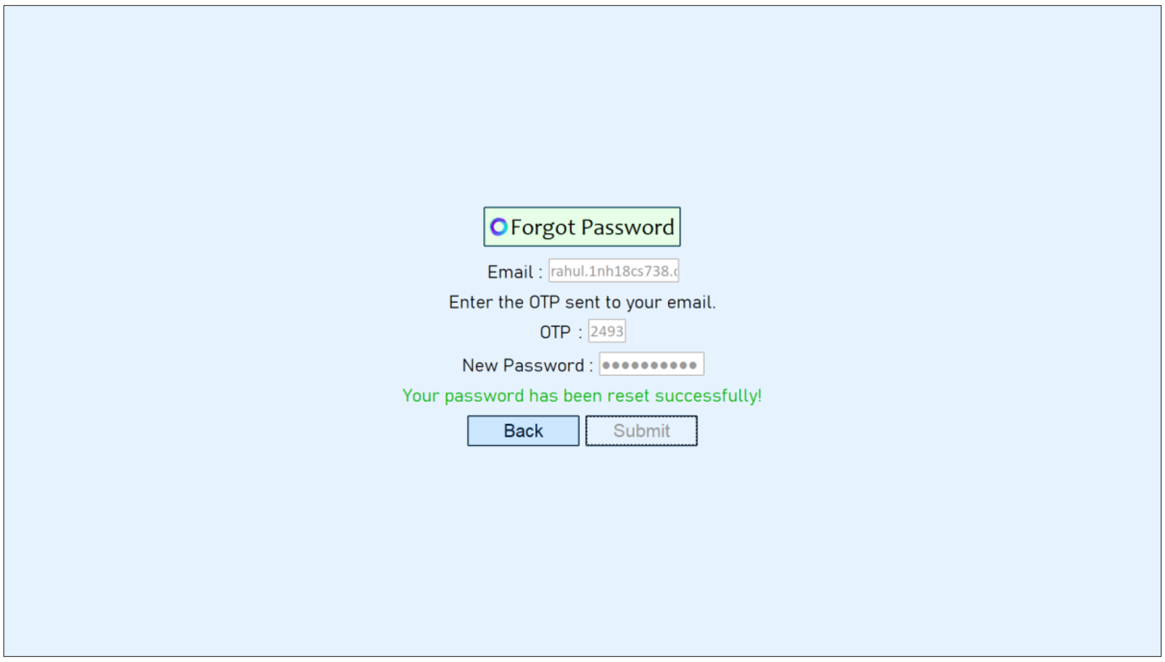
Life360 - Extending Healthcare to your Desktop

Developed by Rahul M Dinesh

If user enters the correct OTP, then he/she can enter a new password for their account



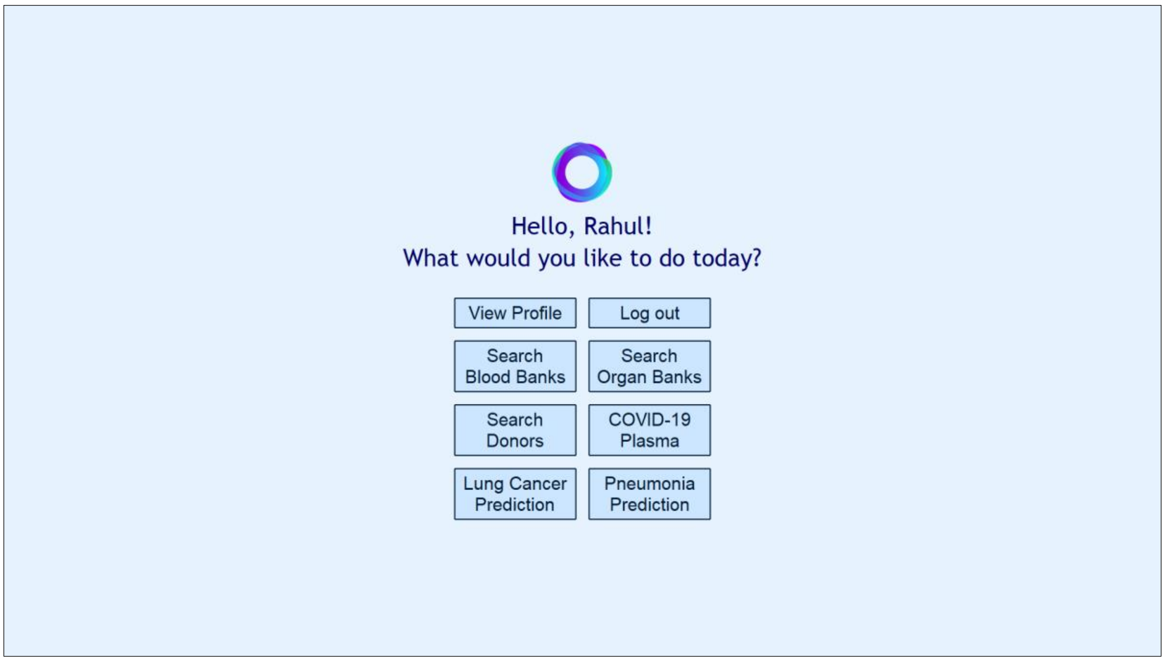
Upon successful resetting of user’s password


Forgot Password
Email : rahul.1nh18cs738.
Enter the OTP sent to your email.
OTP : 2493
New Password : ●●●●●●●●●●
Back
Submit

Your password has been reset successfully!

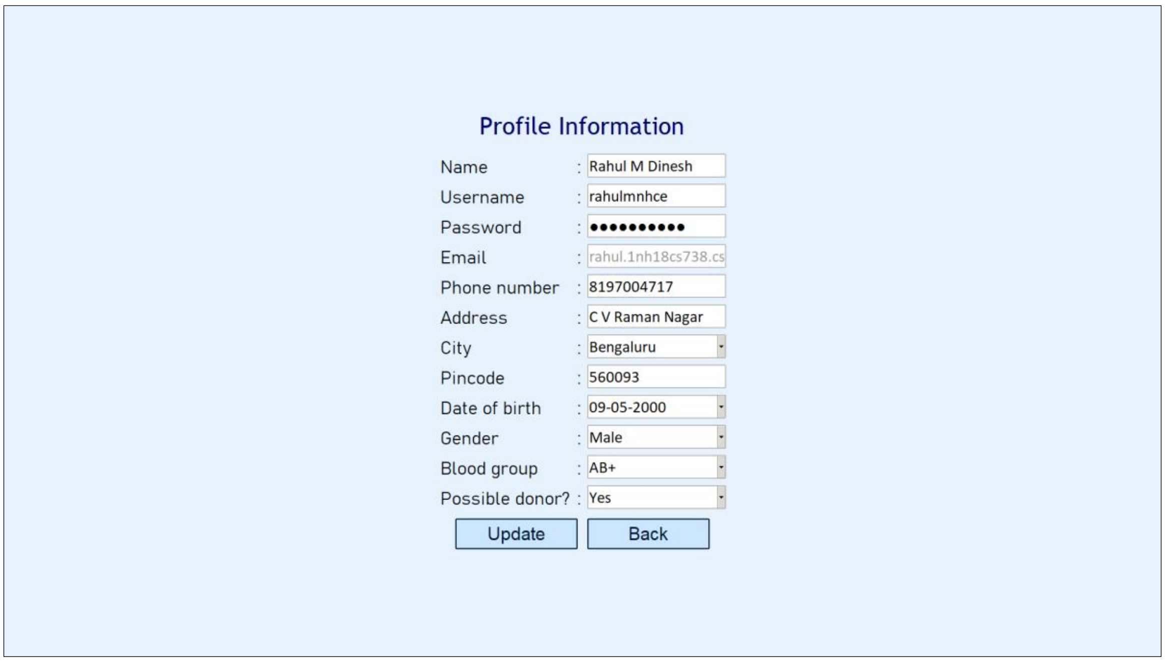
Upon successful logging in

(Individual user type UI)



8.5 USER UI (INDIVIDUAL)

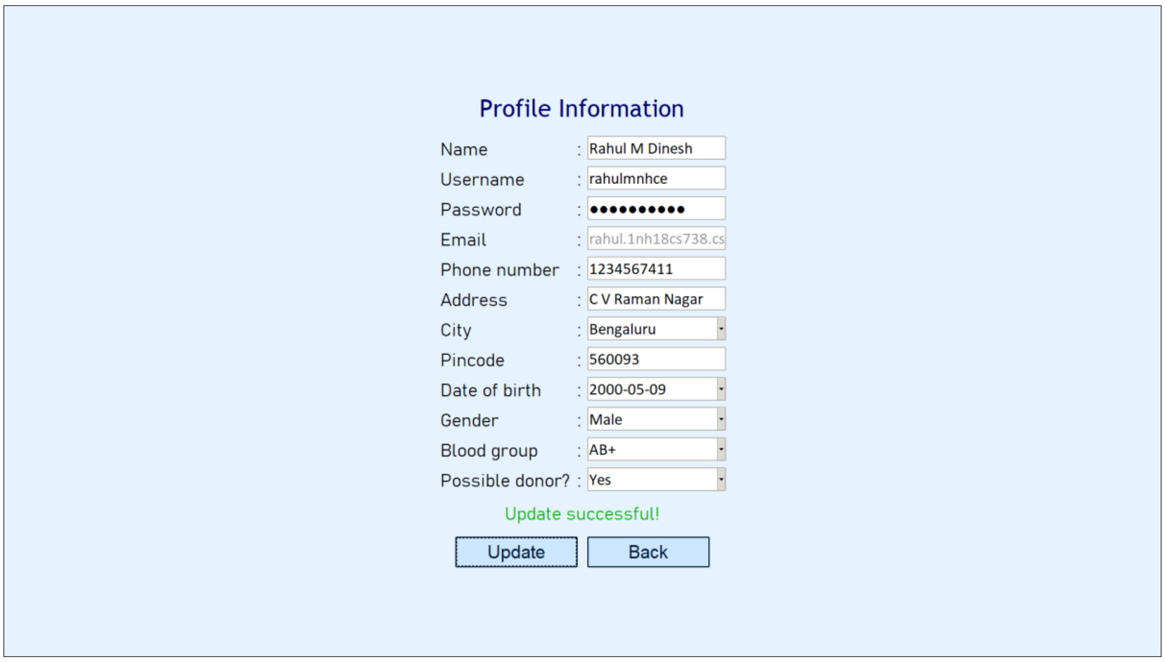
View Profile



Profile Information

Name	:	Rahul M Dinesh
Username	:	rahulmnhce
Password	:	●●●●●●●●●●●
Email	:	rahul.1nh18cs738.cs
Phone number	:	8197004717
Address	:	C V Raman Nagar
City	:	Bengaluru
Pincode	:	560093
Date of birth	:	09-05-2000
Gender	:	Male
Blood group	:	AB+
Possible donor?	:	Yes

Updating Profile Information (with valid data)

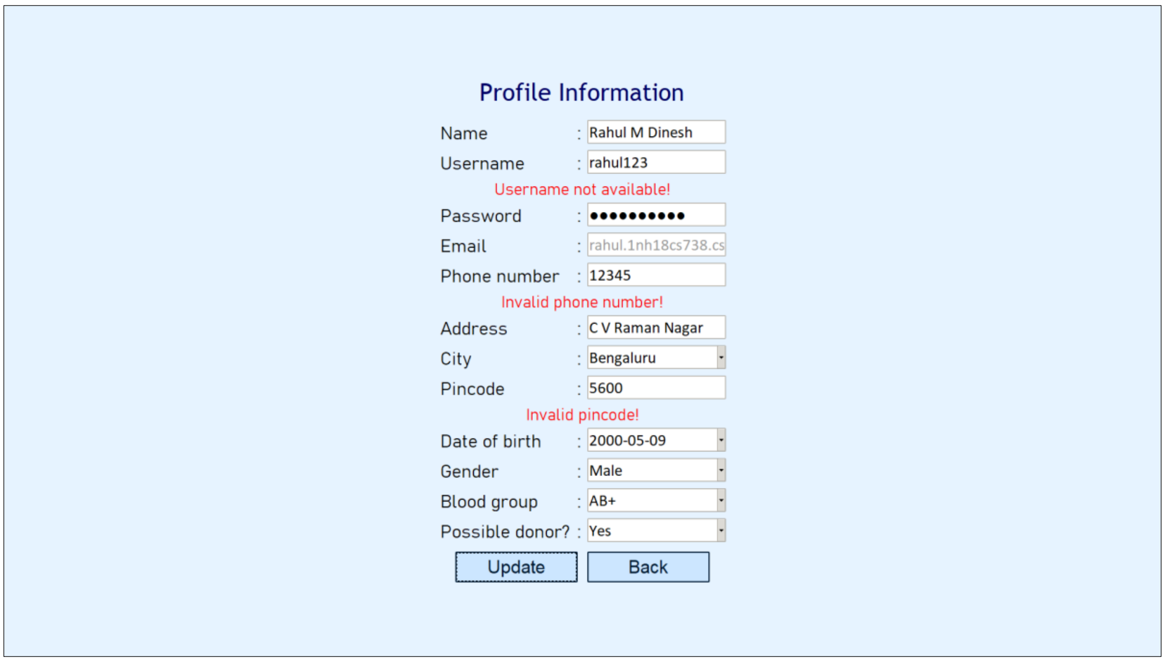


Profile Information

Name	: Rahul M Dinesh
Username	: rahulmnhce
Password	: ●●●●●●●●●●
Email	: rahul.1nh18cs738.cs
Phone number	: 1234567411
Address	: C V Raman Nagar
City	: Bengaluru
Pincode	: 560093
Date of birth	: 2000-05-09
Gender	: Male
Blood group	: AB+
Possible donor?	: Yes

Update successful!

Updating Profile Information (with invalid data)



Profile Information

Name	: Rahul M Dinesh
Username	: rahul123

Username not available!

Password	: ●●●●●●●●●●
Email	: rahul.1nh18cs738.cs
Phone number	: 12345

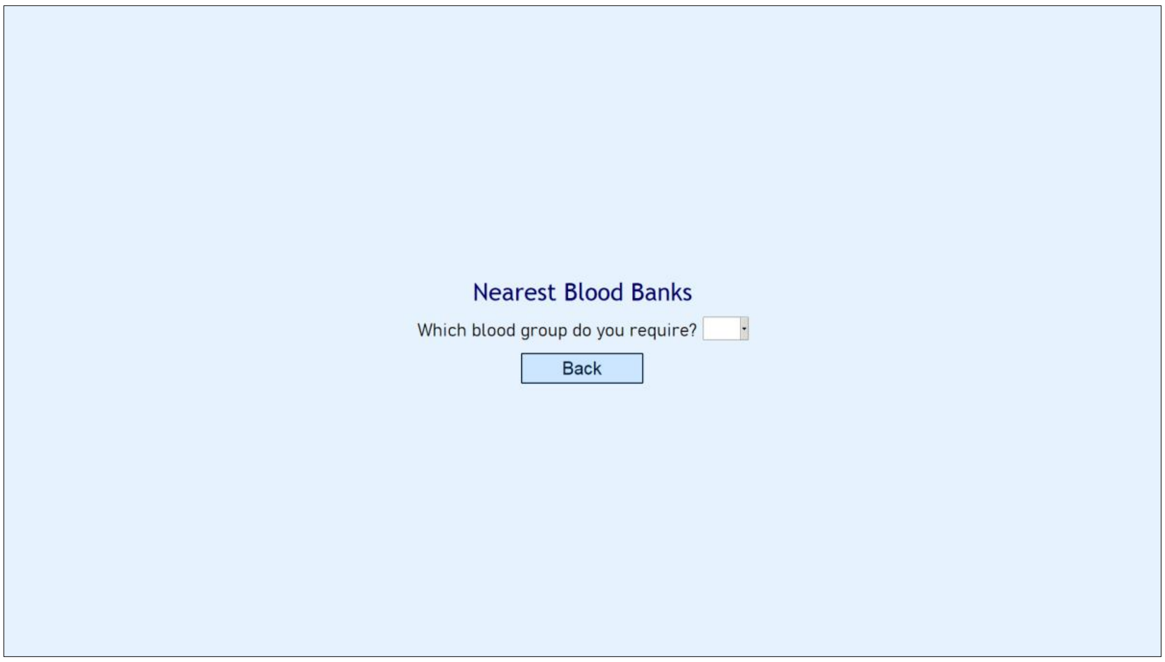
Invalid phone number!

Address	: C V Raman Nagar
City	: Bengaluru
Pincode	: 5600

Invalid pincode!

Date of birth	: 2000-05-09
Gender	: Male
Blood group	: AB+
Possible donor?	: Yes

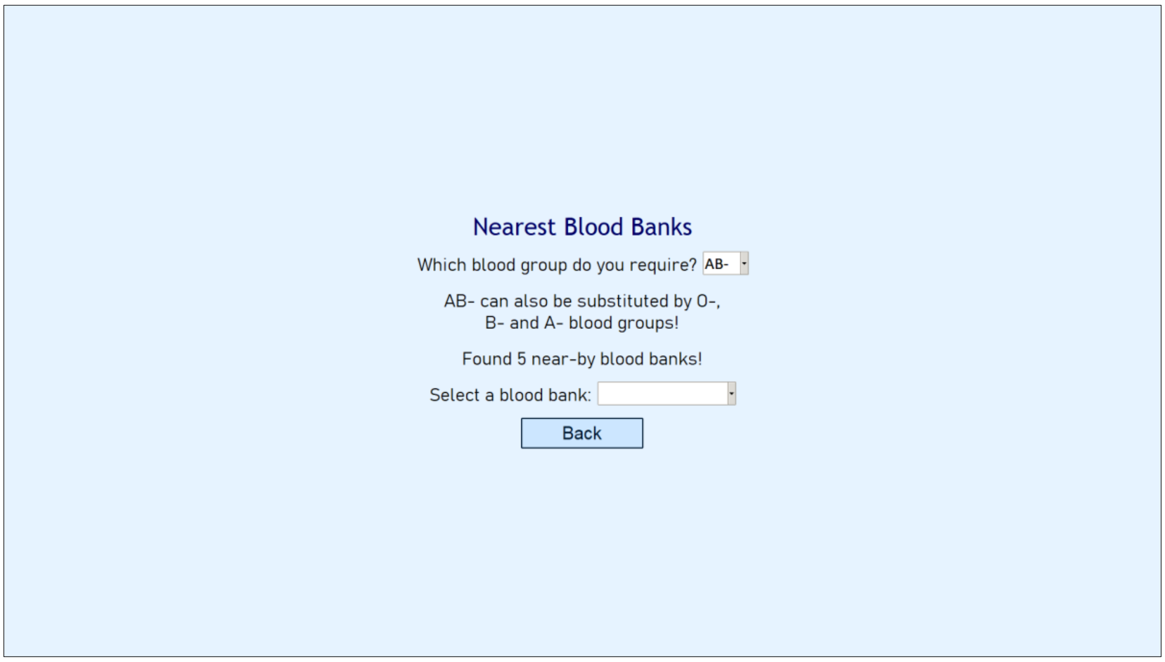
Searching Nearest Blood Banks



Nearest Blood Banks

Which blood group do you require?

Selection of blood groups show the compatible blood groups



Nearest Blood Banks

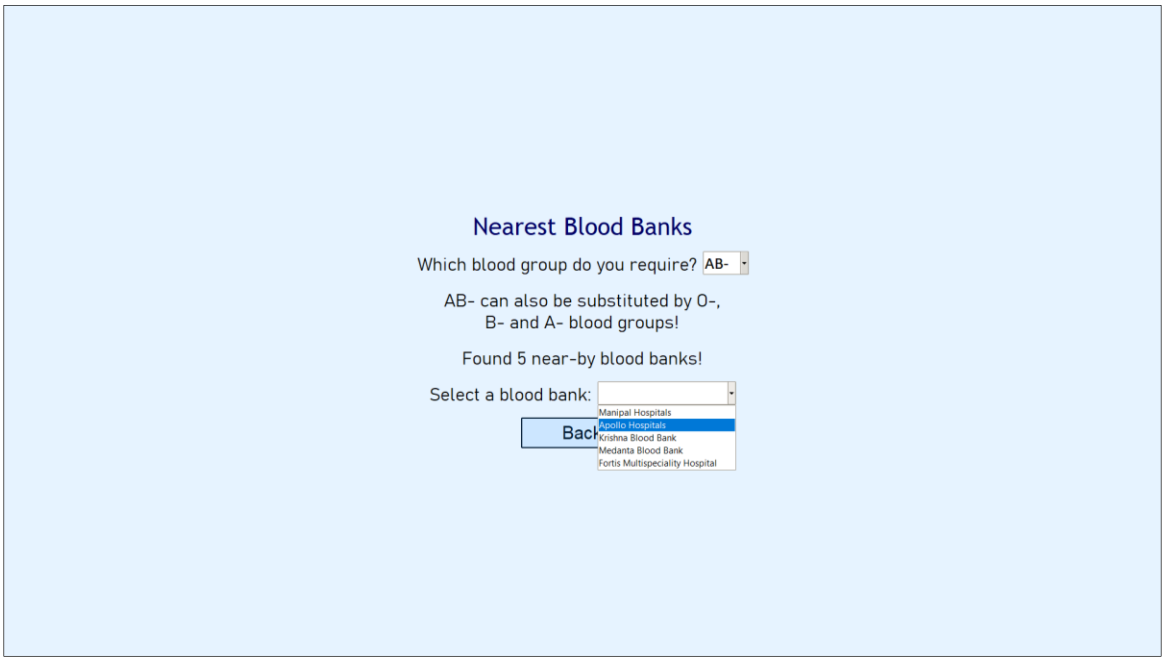
Which blood group do you require? AB-

AB- can also be substituted by O-, B- and A- blood groups!

Found 5 near-by blood banks!

Select a blood bank:

Combobox showing the list of hospitals after querying the database



Nearest Blood Banks

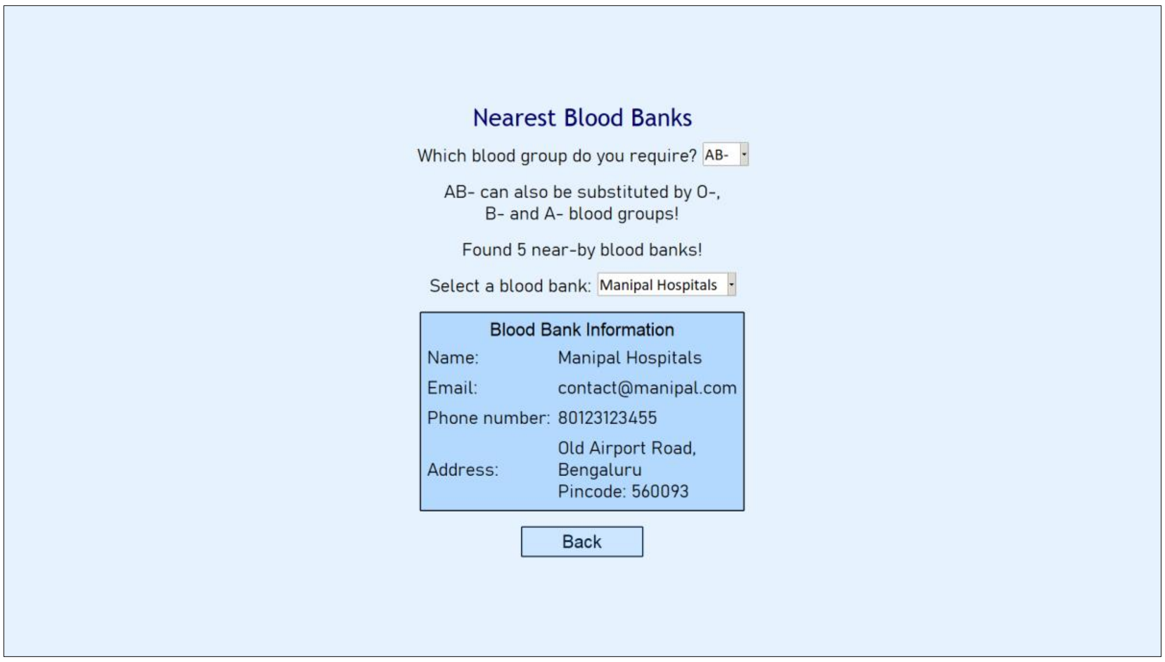
Which blood group do you require? AB-

AB- can also be substituted by O-,
B- and A- blood groups!

Found 5 near-by blood banks!

Select a blood bank:

Upon selecting a blood bank, the information regarding the blood bank is displayed



Nearest Blood Banks

Which blood group do you require? AB-

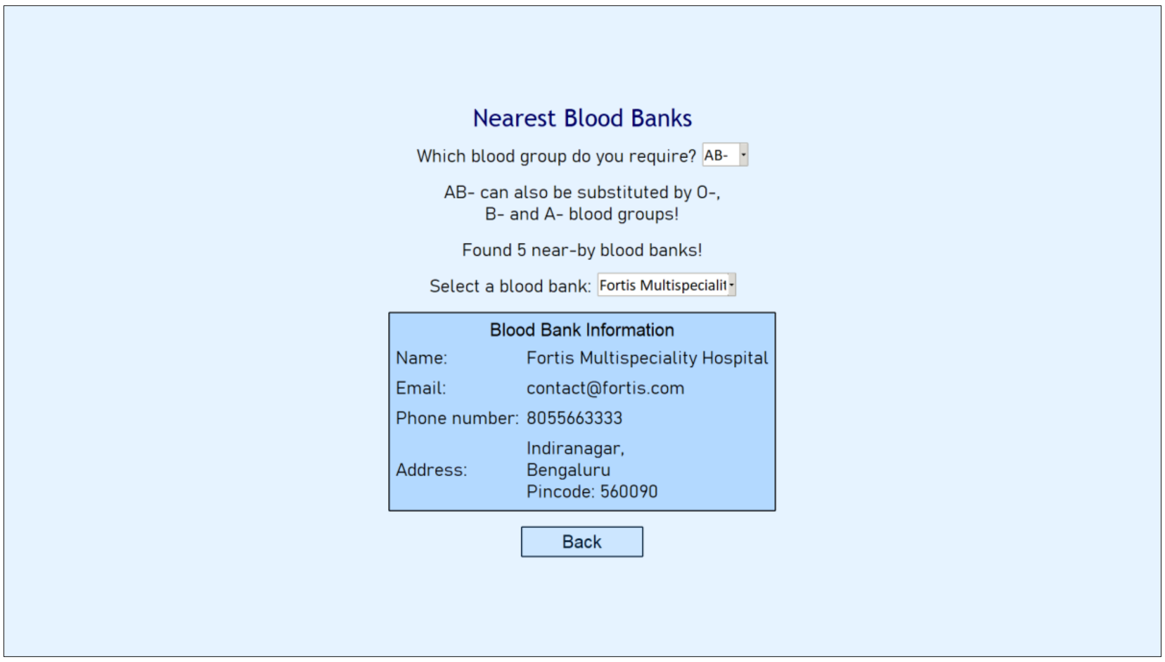
AB- can also be substituted by O-,
B- and A- blood groups!

Found 5 near-by blood banks!

Select a blood bank: Manipal Hospitals

Blood Bank Information

Name:	Manipal Hospitals
Email:	contact@manipal.com
Phone number:	80123123455
Address:	Old Airport Road, Bengaluru Pincode: 560093



Nearest Blood Banks

Which blood group do you require? AB-

AB- can also be substituted by O-, B- and A- blood groups!

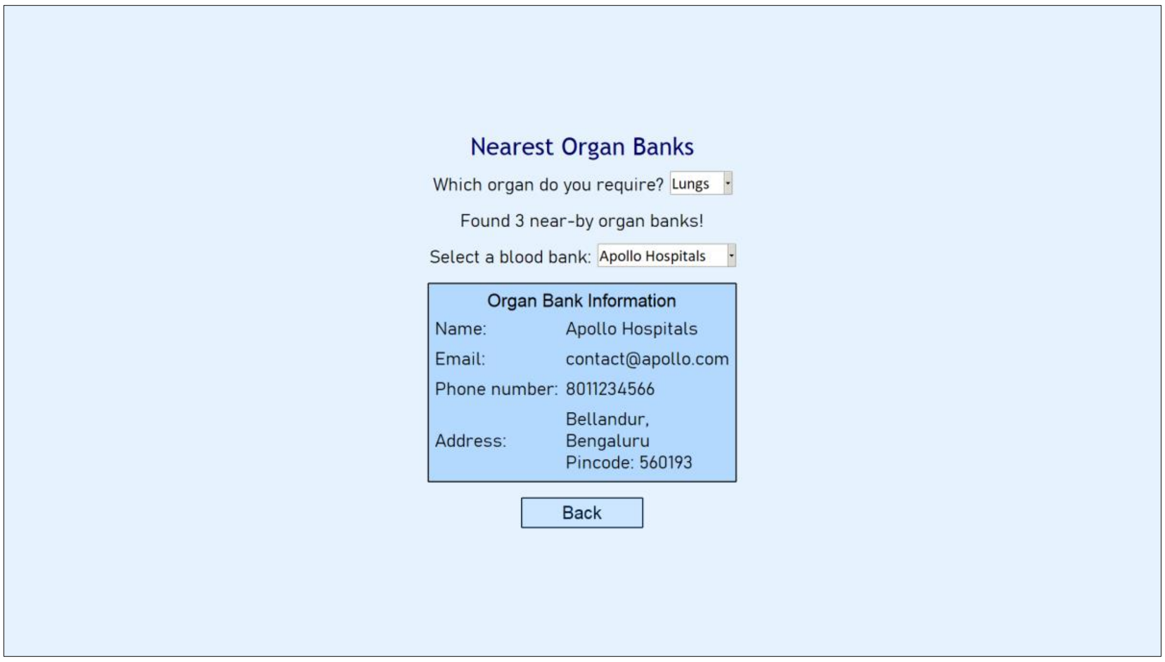
Found 5 near-by blood banks!

Select a blood bank: Fortis Multispecialit

Blood Bank Information

Name:	Fortis Multispeciality Hospital
Email:	contact@fortis.com
Phone number:	8055663333
Address:	Indiranagar, Bengaluru Pincode: 560090

Searching Nearest Organ Banks



Nearest Organ Banks

Which organ do you require? Lungs

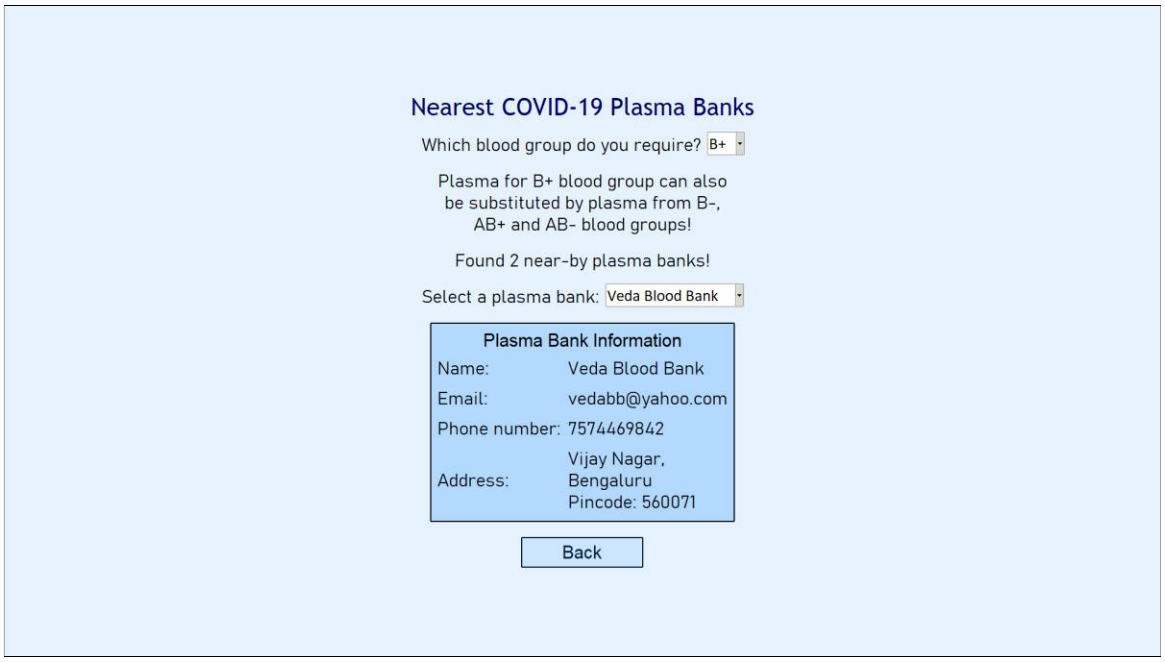
Found 3 near-by organ banks!

Select a blood bank: Apollo Hospitals

Organ Bank Information

Name:	Apollo Hospitals
Email:	contact@apollo.com
Phone number:	8011234566
Address:	Bellandur, Bengaluru Pincode: 560193

Searching Nearest Plasma Banks


Nearest COVID-19 Plasma Banks

Which blood group do you require? B+

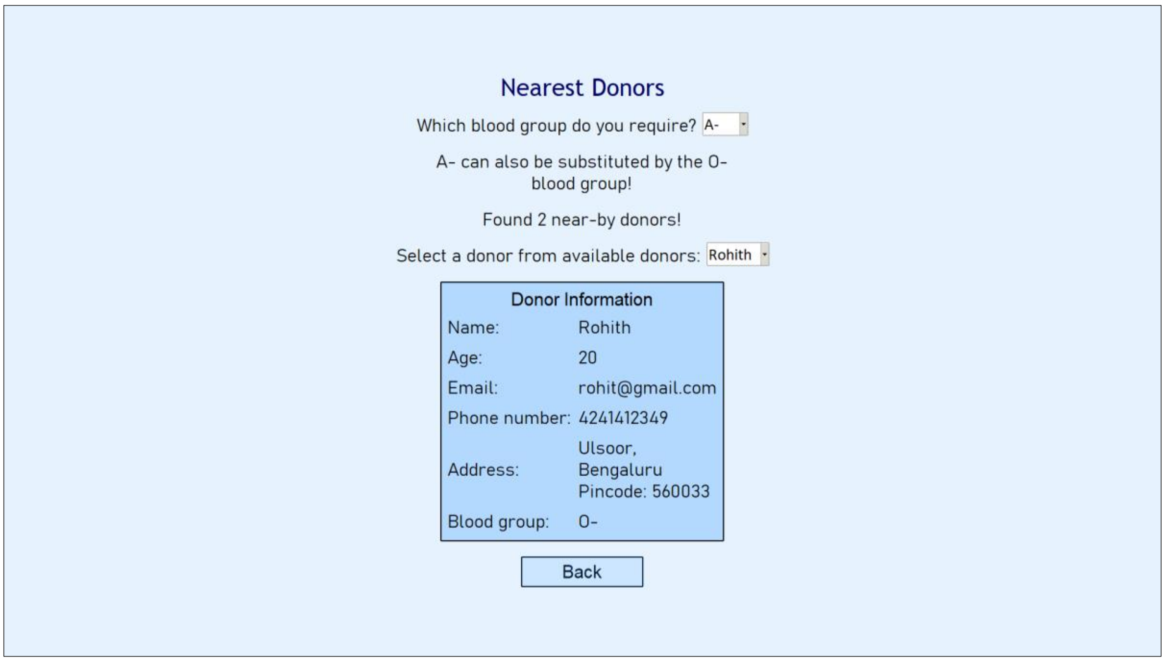
Plasma for B+ blood group can also be substituted by plasma from B-, AB+ and AB- blood groups!

Found 2 near-by plasma banks!

Select a plasma bank: Veda Blood Bank

Plasma Bank Information	
Name:	Veda Blood Bank
Email:	vedabb@yahoo.com
Phone number:	7574469842
Address:	Vijay Nagar, Bengaluru Pincode: 560071

Searching Near-by Blood Donors


Nearest Donors

Which blood group do you require? A-

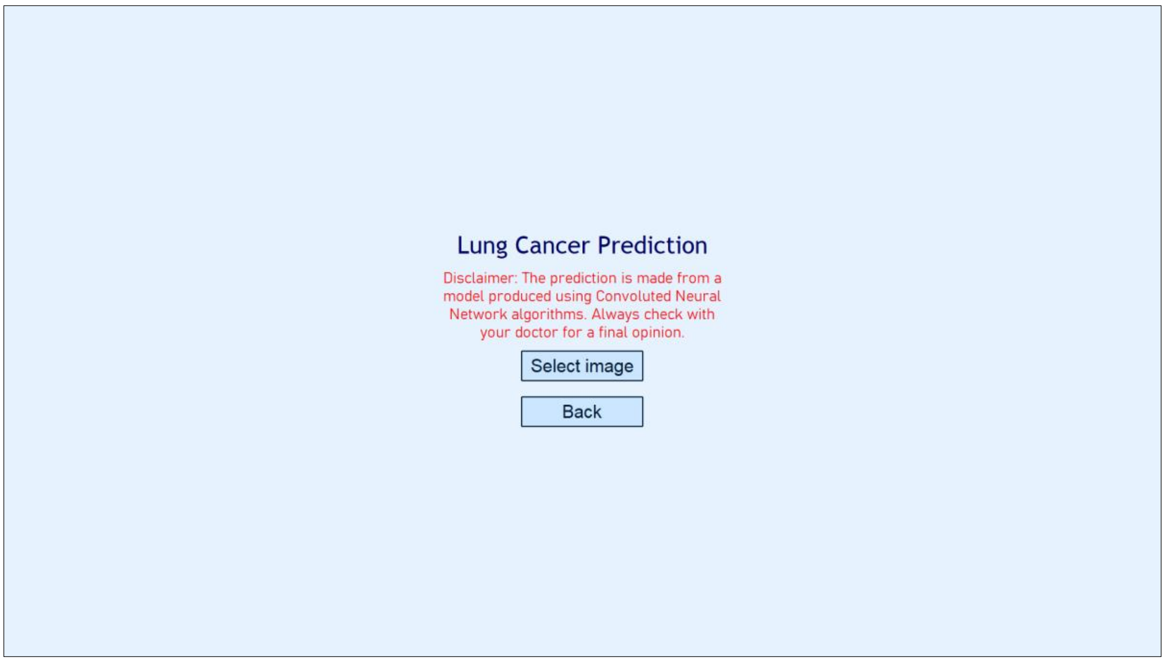
A- can also be substituted by the O- blood group!

Found 2 near-by donors!

Select a donor from available donors: Rohith

Donor Information	
Name:	Rohith
Age:	20
Email:	rohit@gmail.com
Phone number:	4241412349
Address:	Ulsoor, Bengaluru Pincode: 560033
Blood group:	O-

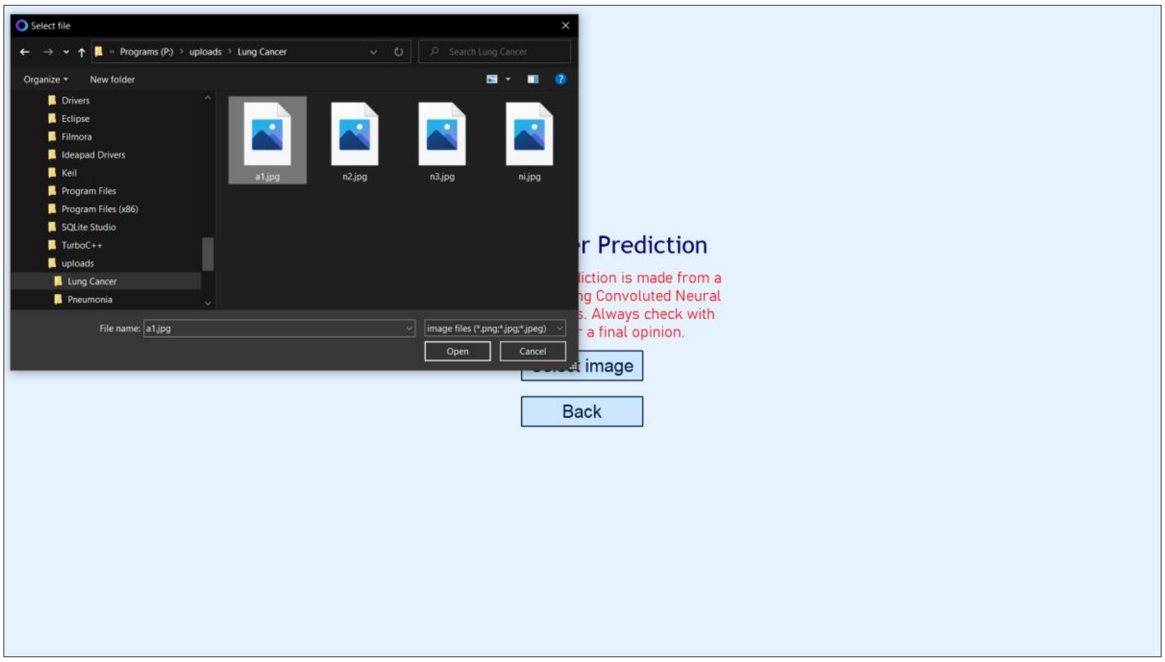
Lung Cancer detection using CNN Prediction Model



Lung Cancer Prediction

Disclaimer: The prediction is made from a model produced using Convoluted Neural Network algorithms. Always check with your doctor for a final opinion.

File dialog box for selecting a valid image file



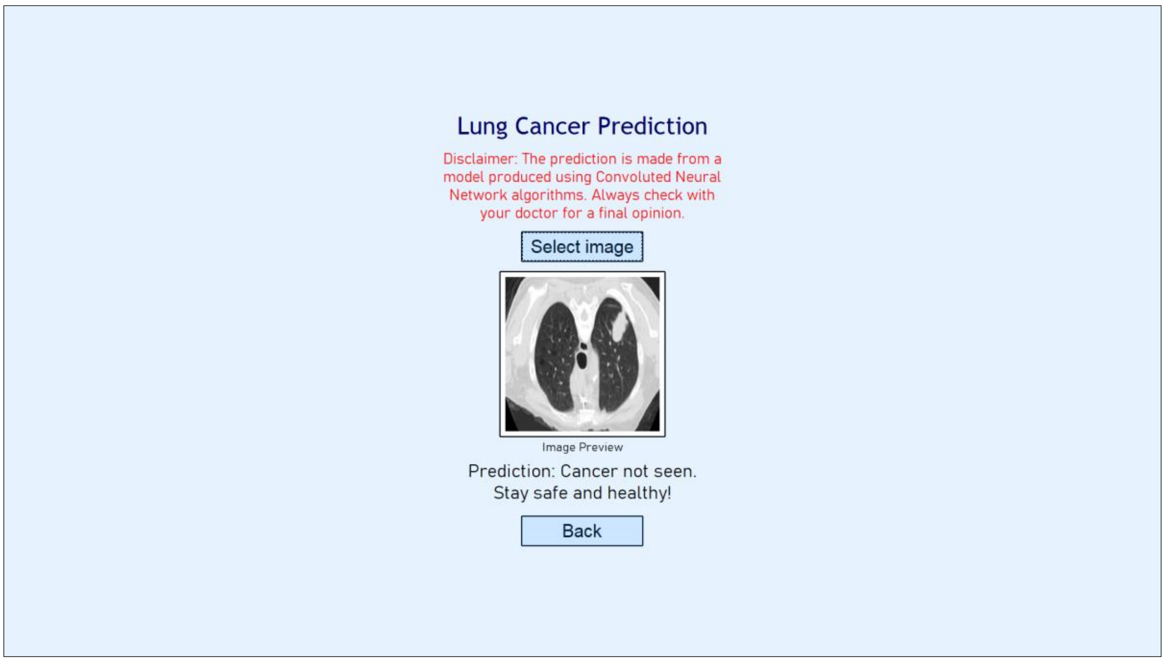
Select file

Programs (P:) › uploads › Lung Cancer

File name: a1.jpg

image files (*.png;*.jpg;*.jpeg)

Lung Cancer Prediction Result (Normal)

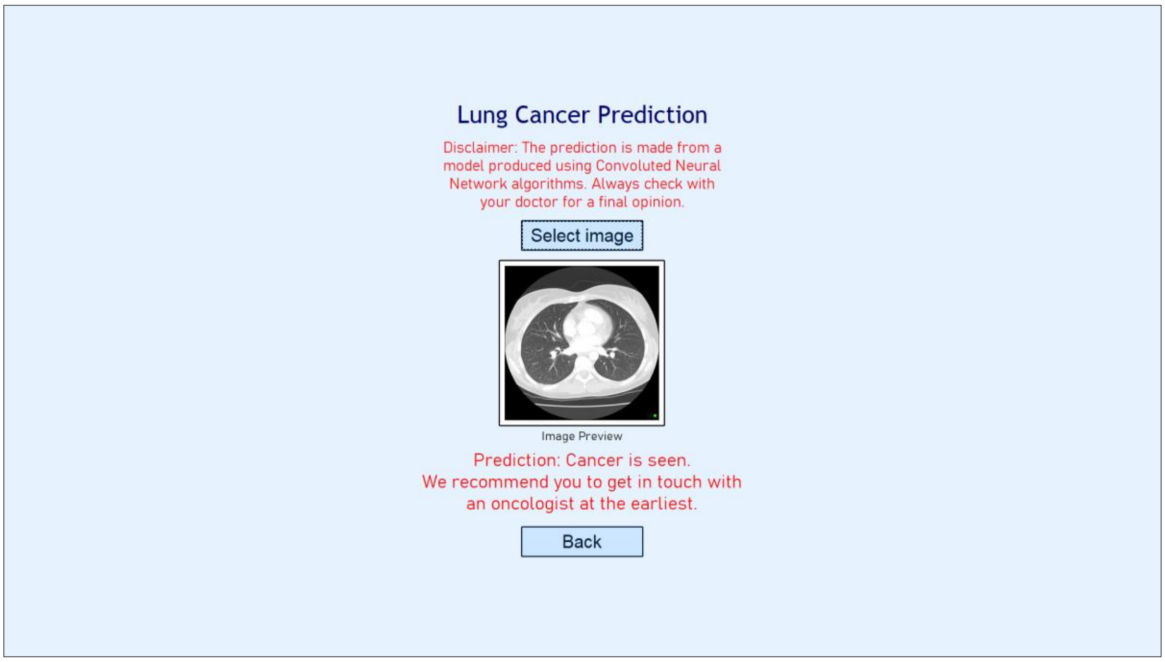


Lung Cancer Prediction

Disclaimer: The prediction is made from a model produced using Convoluted Neural Network algorithms. Always check with your doctor for a final opinion.

Prediction: Cancer not seen.
Stay safe and healthy!

Lung Cancer Prediction Result (Abnormal)

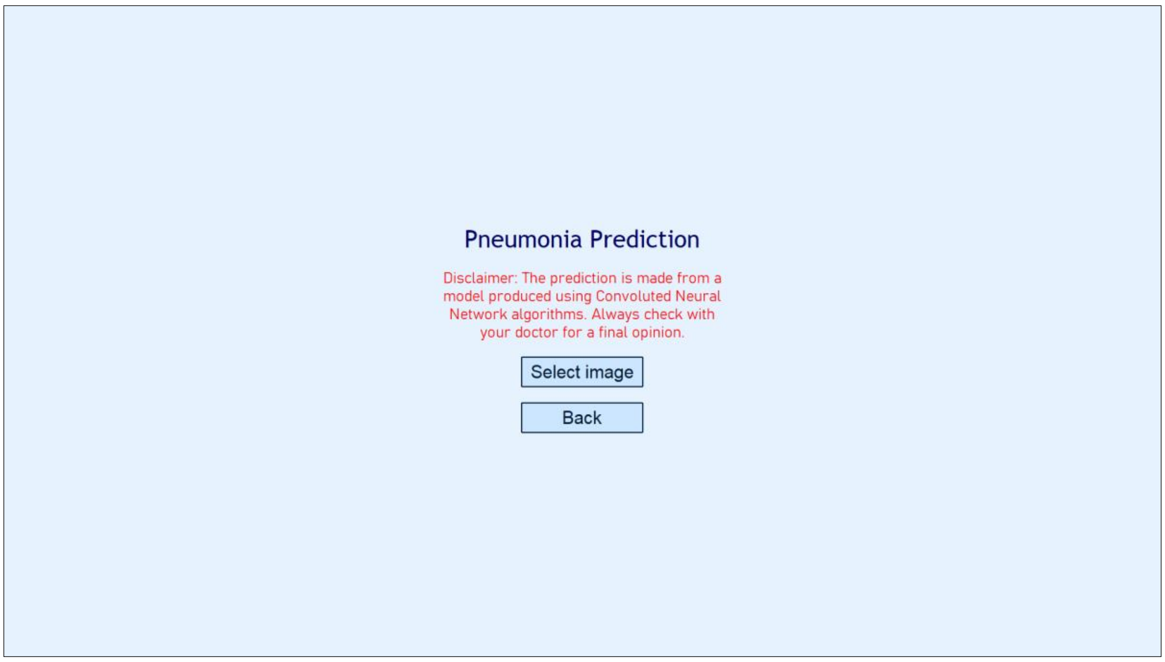


Lung Cancer Prediction

Disclaimer: The prediction is made from a model produced using Convoluted Neural Network algorithms. Always check with your doctor for a final opinion.

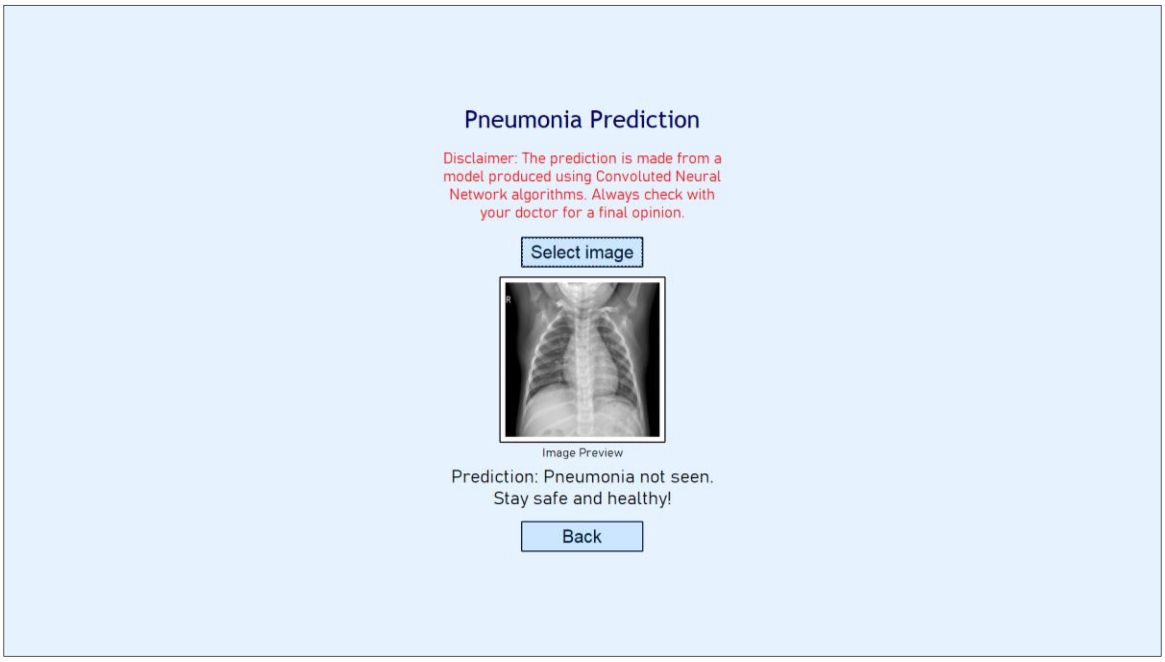
Prediction: Cancer is seen.
We recommend you to get in touch with an oncologist at the earliest.

Lung Cancer detection using CNN Prediction Model


Pneumonia Prediction

Disclaimer: The prediction is made from a model produced using Convoluted Neural Network algorithms. Always check with your doctor for a final opinion.

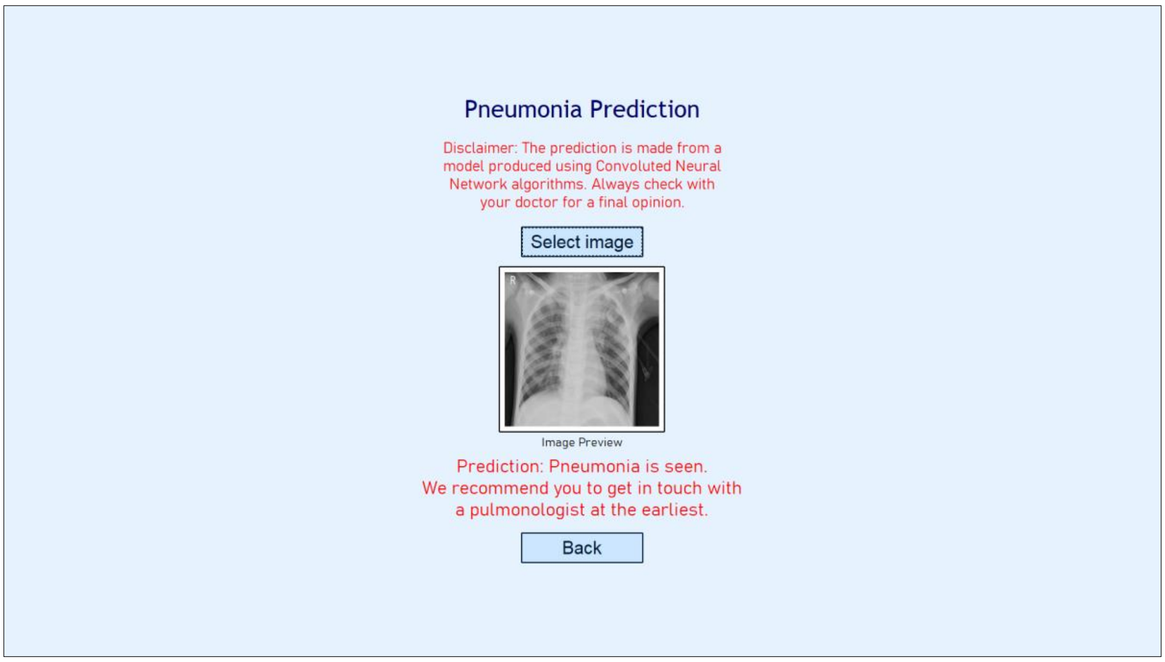
Pneumonia Prediction Result (Normal)


Pneumonia Prediction

Disclaimer: The prediction is made from a model produced using Convoluted Neural Network algorithms. Always check with your doctor for a final opinion.

Prediction: Pneumonia not seen.
Stay safe and healthy!

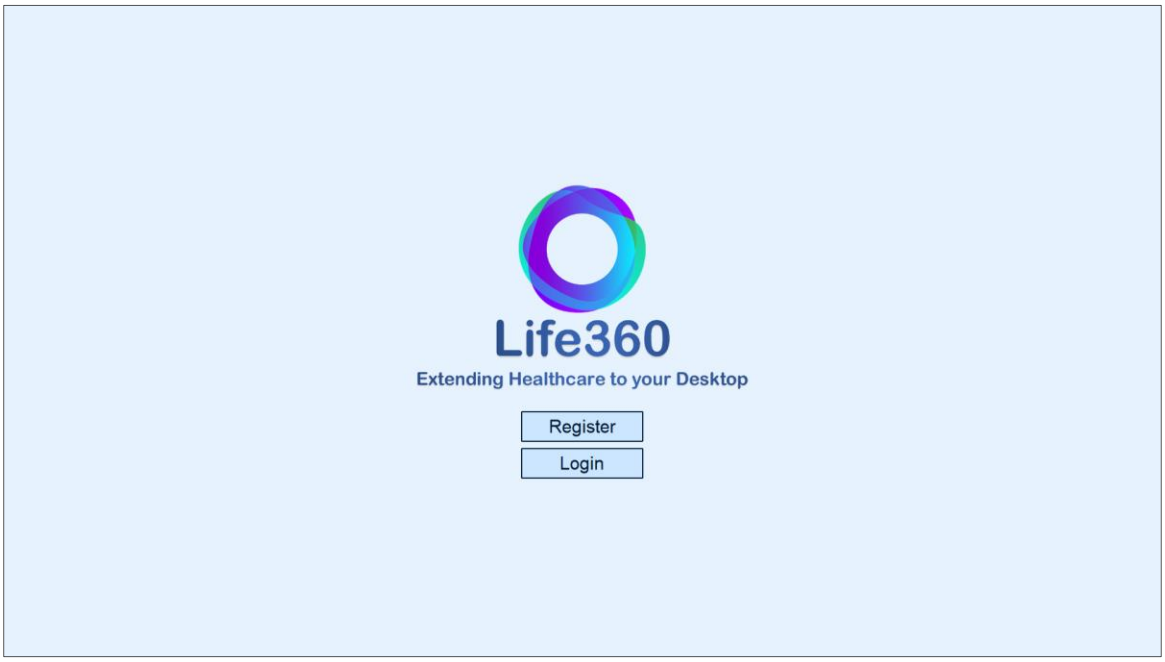
Pneumonia Prediction Result (Abnormal)


Pneumonia Prediction

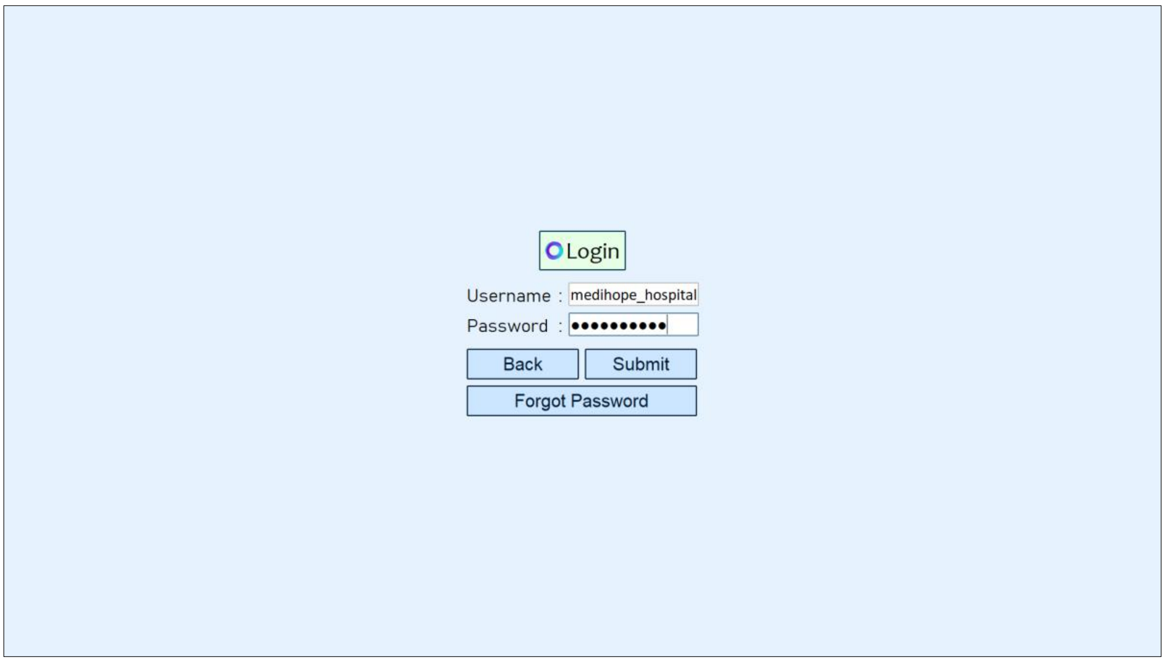
Disclaimer: The prediction is made from a model produced using Convoluted Neural Network algorithms. Always check with your doctor for a final opinion.

Prediction: Pneumonia is seen. We recommend you to get in touch with a pulmonologist at the earliest.

After Logging out, application returns to Launch screen

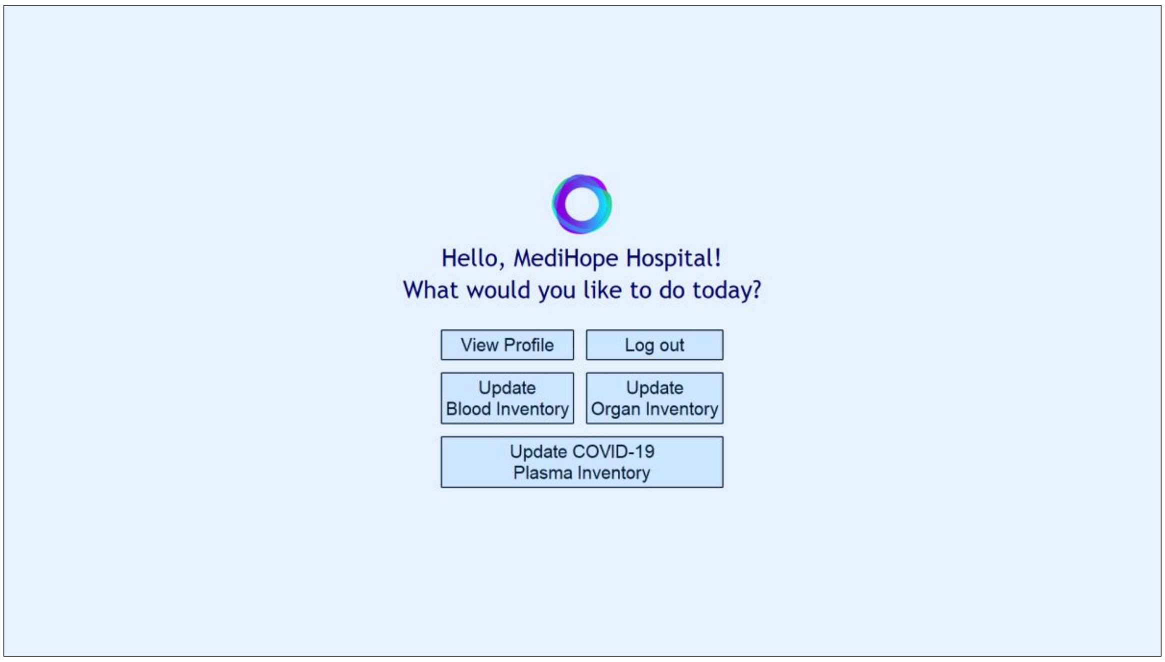


8.6 LOGGING IN (ORGANIZATION)



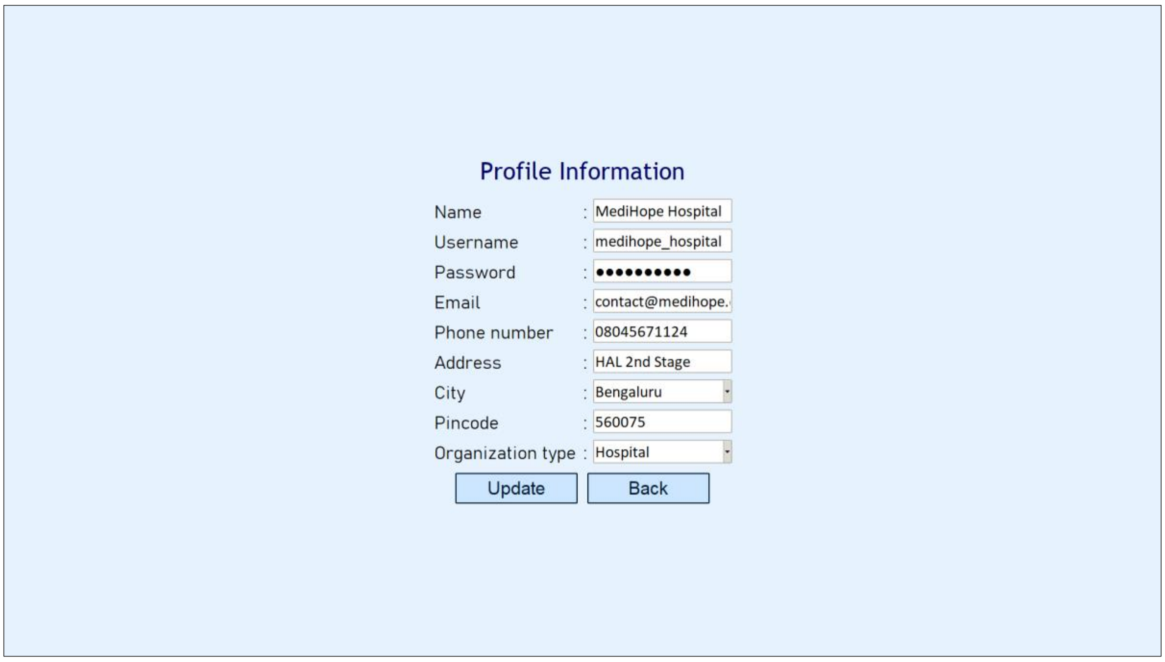
Upon successful logging in

(Organization user type UI)


Hello, MediHope Hospital!
What would you like to do today?

8.7 USER UI (ORGANIZATION)

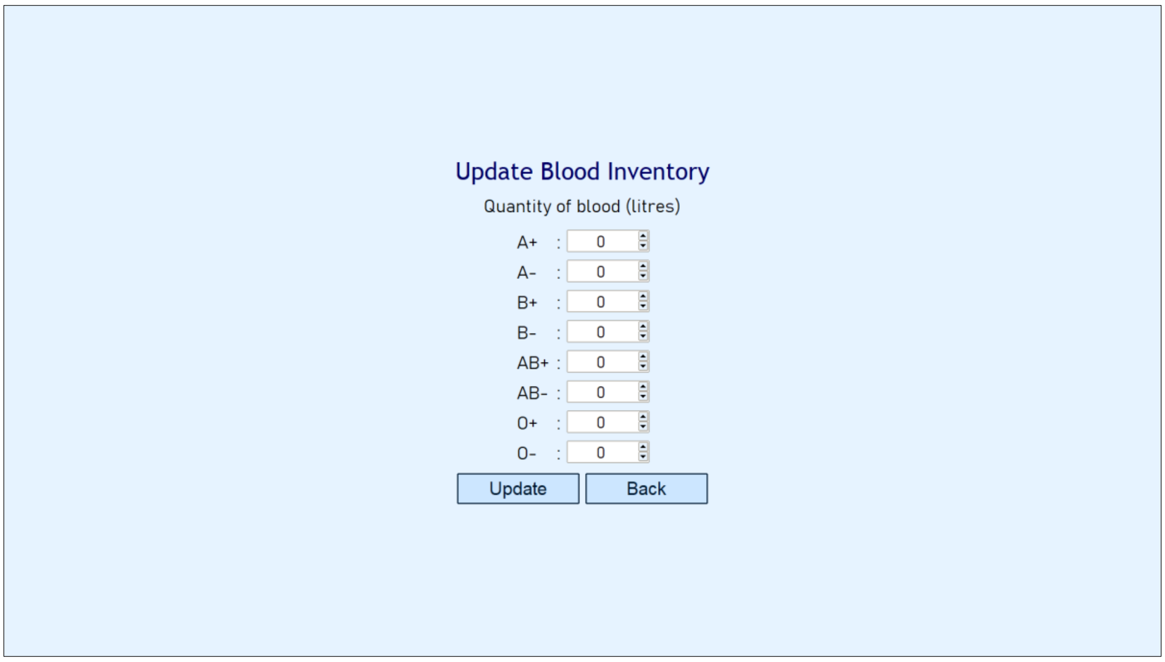
View/Update Profile



Profile Information

Name :	MediHope Hospital
Username :	medihope_hospital
Password :	••••••••••
Email :	contact@medihope.
Phone number :	08045671124
Address :	HAL 2nd Stage
City :	Bengaluru
Pincode :	560075
Organization type :	Hospital

Updating Blood Inventory (new user has values set to 0)

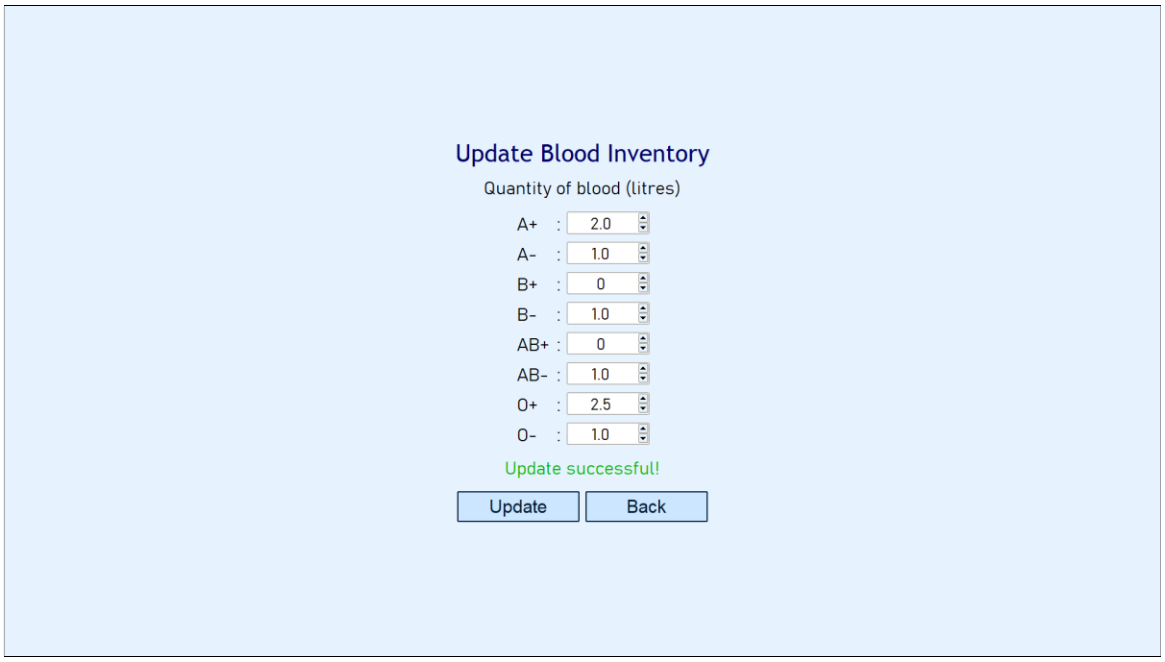


Update Blood Inventory

Quantity of blood (litres)

A+ :	0
A- :	0
B+ :	0
B- :	0
AB+ :	0
AB- :	0
O+ :	0
O- :	0

Updating Blood Inventory (with values)



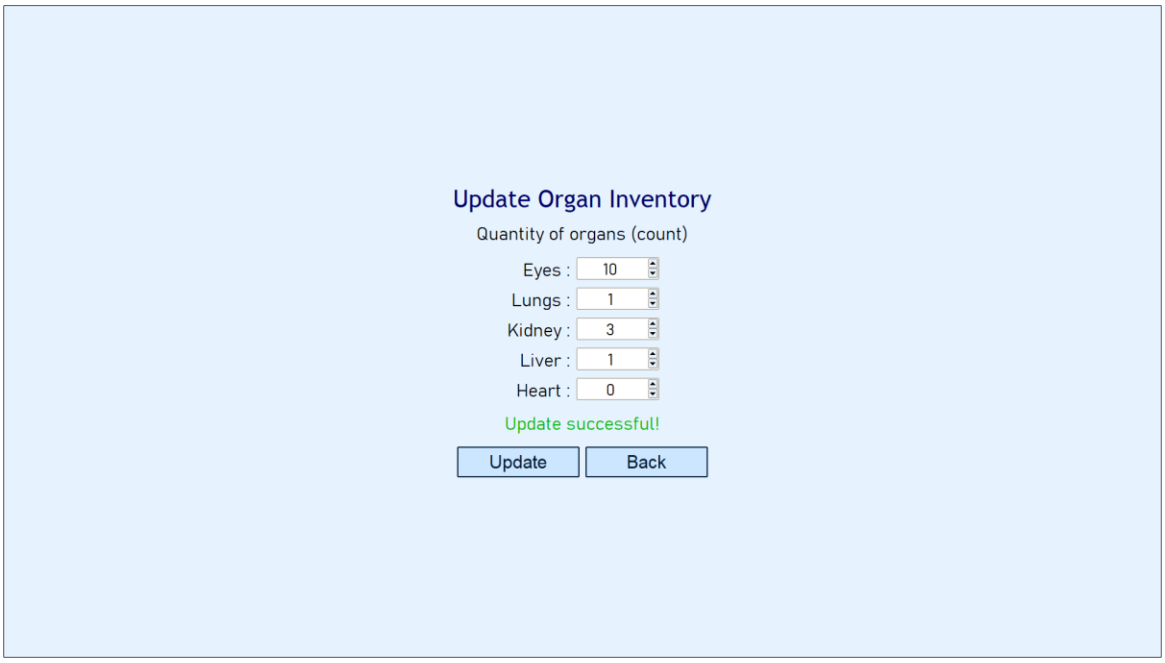
Update Blood Inventory

Quantity of blood (litres)

A+ :	2.0
A- :	1.0
B+ :	0
B- :	1.0
AB+ :	0
AB- :	1.0
O+ :	2.5
O- :	1.0

Update successful!

Updating Organ Inventory (with values)



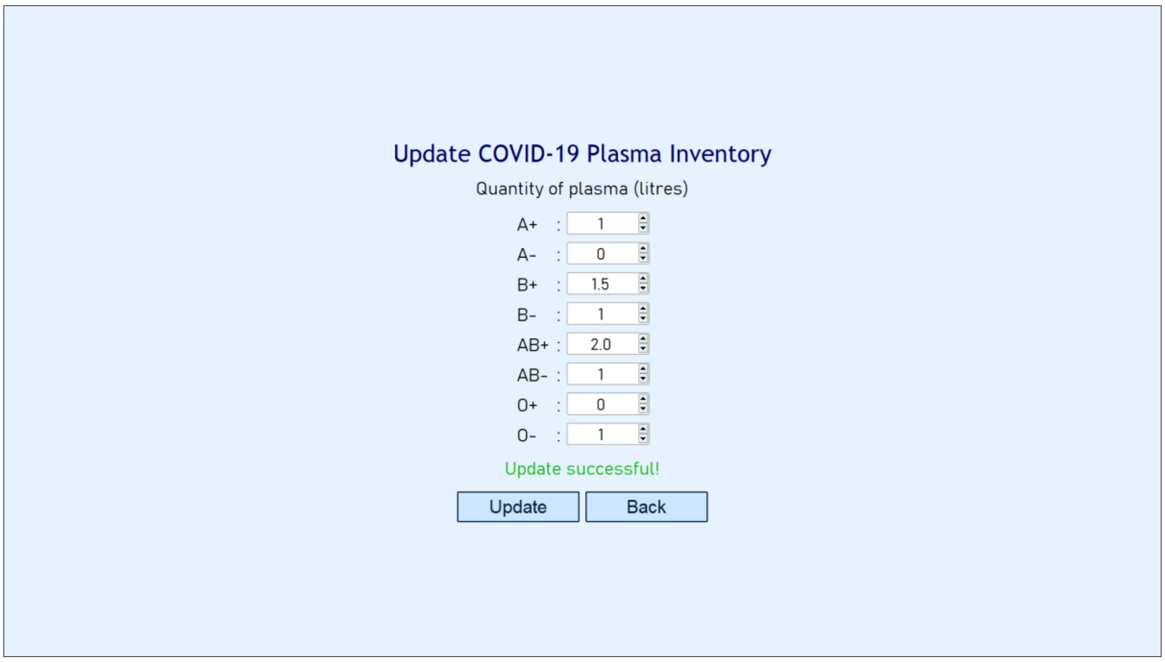
Update Organ Inventory

Quantity of organs (count)

Eyes :	10
Lungs :	1
Kidney :	3
Liver :	1
Heart :	0

Update successful!

Updating Plasma Inventory (with values)



Update COVID-19 Plasma Inventory

Quantity of plasma (litres)

A+	:	1
A-	:	0
B+	:	1.5
B-	:	1
AB+	:	2.0
AB-	:	1
O+	:	0
O-	:	1

Update successful!

CHAPTER 9

CONCLUSION

The mini project has successfully accomplished the goals it had set out in the objectives and design sections of this report.

The Python application developed in this mini project has been able to provide a simple and easy to use graphical user interface that can potentially save lives.

Separate portals have been made for the both type of users, i.e. individual users and organization users.

The individual user type UI has successfully implemented several modules. An individual user can opt to volunteer as a potential blood donor at the time of registration. Once logged in, the user can update his/her profile information which would be shown during a donor search by other users. The user can additionally search for near-by blood banks, organ banks, COVID-19 plasma banks and donors who have been registered on the application. The user can also upload CT scan of lungs and/or X-ray scans of chests to predict whether the scans have any abnormalities. This has been achieved by using model devised from machine learning algorithms, specifically, convolutional neural network algorithms.

The organization user type UI has also successfully implemented several modules. Organizations like hospitals, blood/plasma banks and organ banks can now update their blood inventory/ organ inventory/ plasma inventory on a real time basis and changes will automatically be reflected to any individual users who are searching for near-by blood/ organ/ COVID-19 plasma banks.

The application has also appropriately exploited the power of Python programming through the use of its various data types, built-in and imported libraries and packages.

In the future, I aim to extend this mini project by having the database on a server/cloud so that multiple users from different places can use the application. I also plan to add additional functionalities that would enhance the user experience.

REFERENCES

[1] https://www.programiz.com/

[2] https://www.geeksforgeeks.org/

[3] https://www.javatpoint.com/

[4] https://anzeljg.github.io/

[5] https://www.tutorialspoint.com/

[6] https://docs.python.org/

[7] https://realpython.com/

[8] https://www.edureka.co/

[9] https://pythonprogramming.net/

[10] https://www.tutorialsteacher.com/