# COVID-19 VACCINATION MONITORING SYSTEM

**A PROJECT REPORT**

***Submitted by***

**ISHAN THAKUR [Reg No: RA1911003030068]**
**AKHIL ABHILASH [Reg No: RA1911003030086]**
**HARSHIT TYAGI [Reg No: RA1911003030093]**

*Under the guidance of*

**Mrs. Bharti Vidhury**

(Professor, Department of Computer Sciene & Engineering)

***in partial fulfillment for the award of the degree***

***of***

**BACHELOR OF TECHNOLOGY**

in

**COMPUTER SCIENCE ENGINEERING**

of

**FACULTY OF ENGINEERING AND TECHNOLOGY**


SRM INSTITUTE OF SCIENCE TECHNOLOGY, Delhi-NCR

**NOV 2022**

# SRM INSTITUTE OF SCIENCE TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

## BONAFIDE CERTIFICATE

Certified that this project report titled "**COVID-19 VACCINATION MONITORING SYSTEM**" is the bonafide work of **AKHIL ABHILASH , HARSHIT TYAGI , ISHAN THAKUR** , who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

**SIGNATURE**

Mrs. Bharti Vidhury
**GUIDE**
Professor
Dept. of Computer Sciene & Engineering

**SIGNATURE**

Dr. R.P.Mahapatra
**HEAD OF THE DEPARTMENT**
Dept. of Computer Science Engineering

Signature of the Internal Examiner

Signature of the External Examiner

# ABSTRACT

CVMS is designed to simplify the COVID - 19 Vaccination process that's running throughout the world and help to boost the efficiency and focuses on developing a software which will help people to keep track of their COVID-19 Vaccination Date, Time Slot, Place and will also remind them to complete the process. Software will also keep count on the total number and type of vaccines distributed. CVMS system will ensure that the vaccination process will be done in a systematic and efficient manner. CVMS development team is committed to privacy and being transparent about government requests for customer data globally. The CVMS system is developed using the Iterative model. Iterative process starts with a simple implementation of a subset of the software requirements and iteratively enhances the evolving versions until the full system is implemented. At each iteration, design modifications are made and new functional capabilities are added. The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental).

# ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my guide, Mrs. Bharti Vidhury her valuable guidance, consistent encouragement, personal caring, timely help and providing me with an excellent atmosphere for doing research. All through the work, in spite of her busy schedule, she has extended cheerful and cordial support to us for completing this project work.

**Author**

# TABLE OF CONTENTS

# ABBREVIATIONS

| | |
|---|---|
| **CVMS** | COVID-19 Vaccination Monitoring System |
| **UX** | User Experience |
| **IDE** | Integrated Development Environment |
| **WBS** | Work Breakdown Structure |
| **MS** | Microsoft Office |
| **OOP** | Object Oriented Programming |
| **ER** | Entity Relationship |
| **DFD** | Data Flow Diagram |
| **AUIDN** | Aadhar Unique Identification Numbers |
| **UML** | Unified Modelling Language |
| **BPMN** | Business Process Modelling Notation |
| **SDLC** | Software Development Life Cycle |
| **GUI** | Graphical User Interface |
| **HTTP** | Hyper Text Transfer Protocol |
| **HTML** | Hyper Text Markup Language |
| **XML** | Extensible Markup Language |
| **INR** | Indian Rupee |
| **QR** | Quick Response |

# CHAPTER 1

# INTRODUCTION

## 1.1 Outline:

CVMS project is exclusively designed for the government and respected citizens to simplify the COVID - 19 Vaccination process that's running throughout the world. We are so busy in our daily lives right now that it becomes impossible to keep track of things. To address this issue, this project will help the government to automate the patient selection process and will boost the efficiency and focuses on developing a software which will help people to keep track of their COVID-19 Vaccination Date, Time Slot, Place and will also remind them to complete the process. Software will also keep count on the total number and type of vaccines distributed.

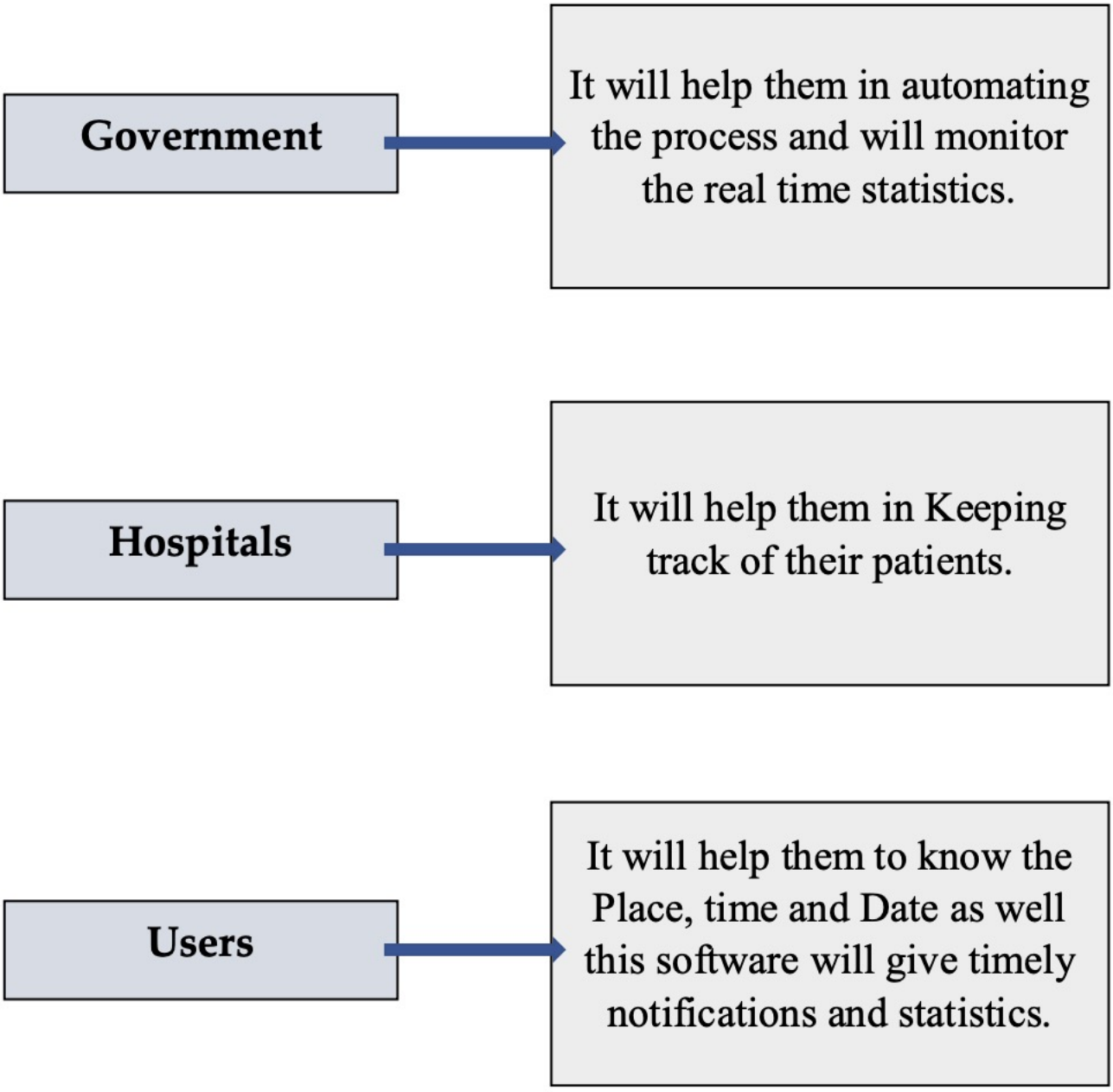
## 1.2 Strategy:

Our strategy is to project best in industry customer service, and the current situation does not reflect this. The new CVMS system will ensure that the vaccination process will be done in a systematic and efficient manner. CVMS development team is committed to privacy and being transparent about government requests for customer data globally.

# CHAPTER 2

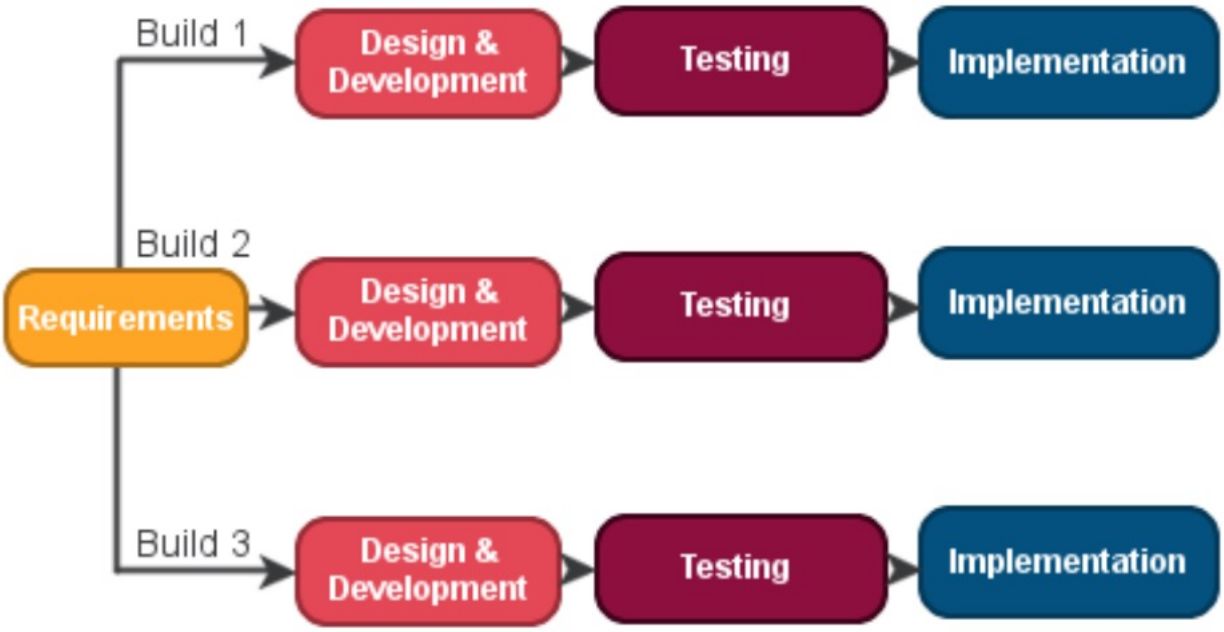
## STAKEHOLDER AND PROCESS MODEL OVERVIEW

### 2.1 Stakeholder:



## 2.2 Process Model:

The CVMS system will be processed using the Iterative model. Iterative process starts with a simple implementation of a subset of the software requirements and iteratively enhances the evolving versions until the full system is implemented. At each iteration, design modifications are made and new functional capabilities are added. The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental).



# CHAPTER 3

# IDENTIFYING THE REQUIREMENTS

## 3.1 System Requirement:

- The System will provide security to user's data, user friendly experience and easy to access general data.
- The System will be capable of providing the total number of vaccinated people.
- The government will be able to access all the information from the system
- The Subsystem #1 will be capable of taking reviews from the people who are vaccinated.
- The Subsystem #2 will be for donations for people who can't afford hospital bills and medicines for COVID-19 treatment.

## 3.2 Software Requirement:

- **Developer:** Using windows 10 having adequate processing power to support the project.
- **User:** Requires basic software capacity.

## 3.3 Functional Requirement:

- **Government:** It would provide them the ability to do the tasks without any human errors as the system is purely automated. All the data of the patients can be stored in one place which will be easier to access and govern.

- **Hospitals:** It would help them to keep track of the patients who have to be vaccinated and who have been vaccinated. It would also keep records of the doses of the vaccinations given on a digital platform.

- **Users:** It would give them timely notifications for the date and time, notify about the place and procedure of how the vaccination process will take place.

## 3.4 Non Functional Requirement:

- **Portability:** The system can be used on any of the operating systems- android, iOS, windows, mac.
- **Security:** All the user data will be encrypted and stored on cloud. The user's personal data can only be accessed by the user and the government.
- **Flexibility:** The system is user friendly and easy to access.
- **Scalability:** The system can easily store data of mass
- **Performance:** The statistics will be updated on the hourly basis so that the user can get access to the latest information about the vaccination.
- **Maintainability:** In case of an unexpected server crash, the restoring of the servers can be done at a considerable pace.

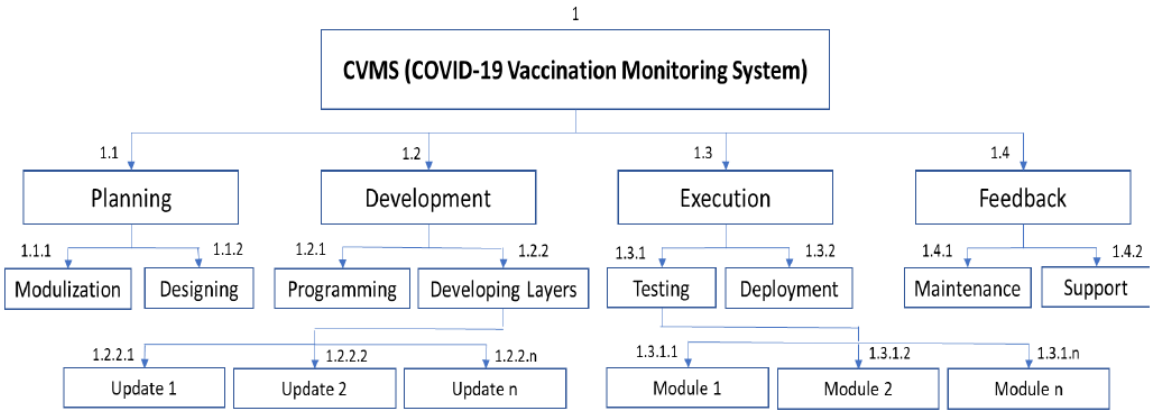
# CHAPTER 4

# WORK BREAKDOWN STRUCTURE

## 4.1 Approach:

A Work Breakdown Structure (WBS) is a deliverable-oriented hierarchical decomposition of the work to be executed by the project team to accomplish the project objectives and create the required deliverables.

The project is being divided into small modules until it is not further divisible. The root of the tree of work breakdown structure is labelled by the project name. For constructing a work breakdown structure, each node is recursively divided into smaller sub-modules, until the leaf level. the modules become undividable and independent. It follows a Top-Down approach.

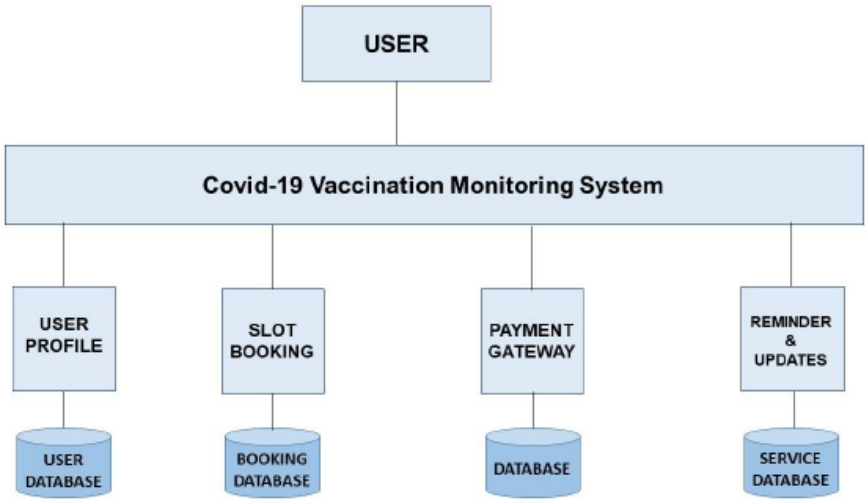


# CHAPTER 5

# ARCHITECTURE DESIGN

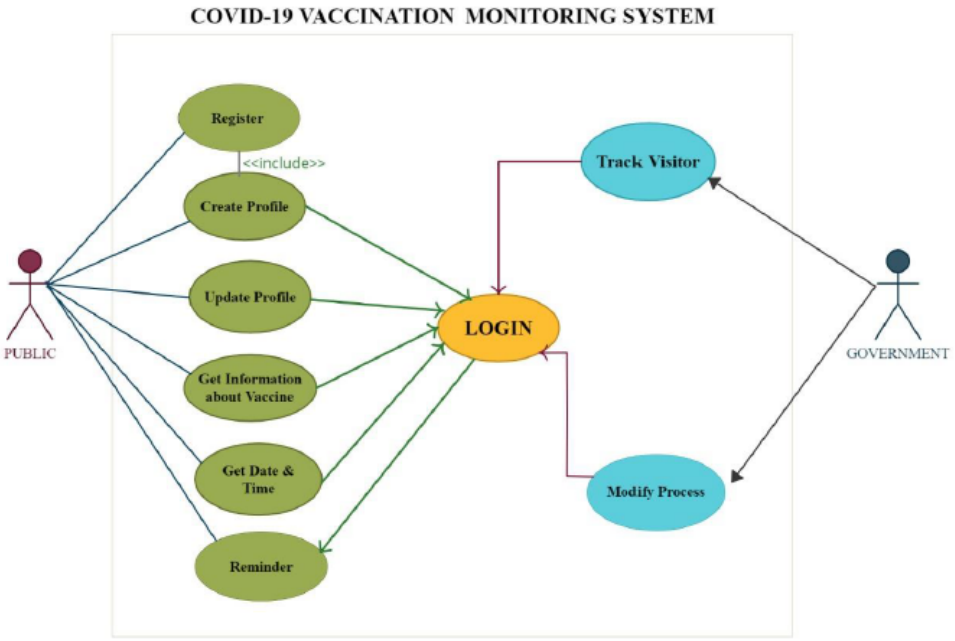
## 5.1 System Architecture:

An architectural diagram is a diagram of a system that is used to abstract the overall outline of the software system and the relationships, constraints, and boundaries between components. It is an important tool as it provides an overall view of the physical deployment of the software system and its evolution roadmap.



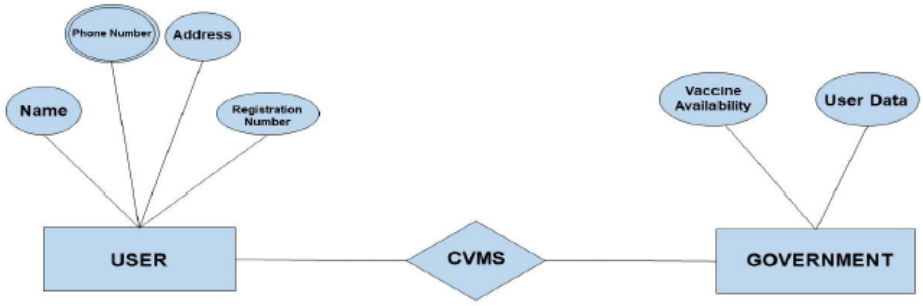

## 5.2 Use Case:

Use case diagram can summarize the details of your system's users (also known as actors) and their interactions with the system.




## 5.3 Entity Relation:

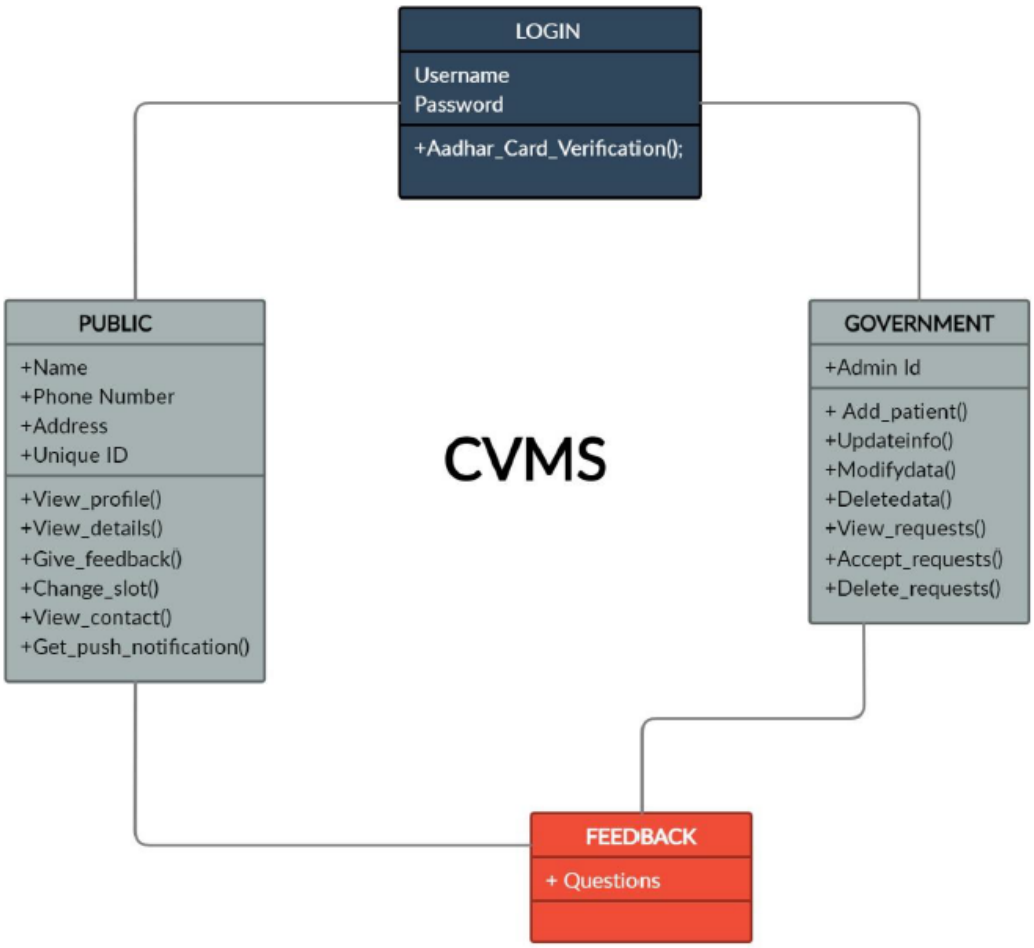
Illustrates how "entities" such as people, objects or concepts relate to each other within a system.




## 5.4 Dividing into Classes:

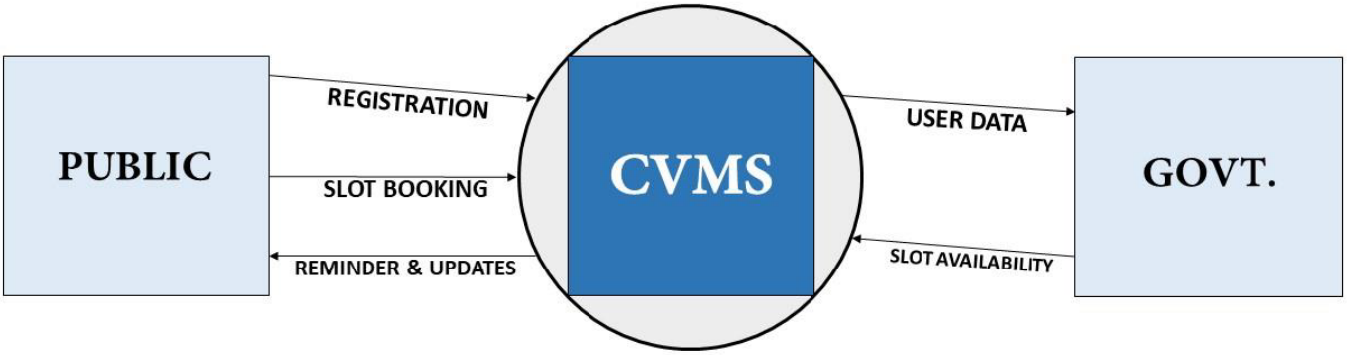
Classe are the main building blocks of every object-oriented method. The class diagram can be used to show the classes, relationships, interface, association, and collaboration. UML is standardized in class diagrams. Since classes are the building block of an application that is based on OOPs, so as the class diagram has an appropriate structure to represent the classes, inheritance, relationships, and everything that OOPs have in

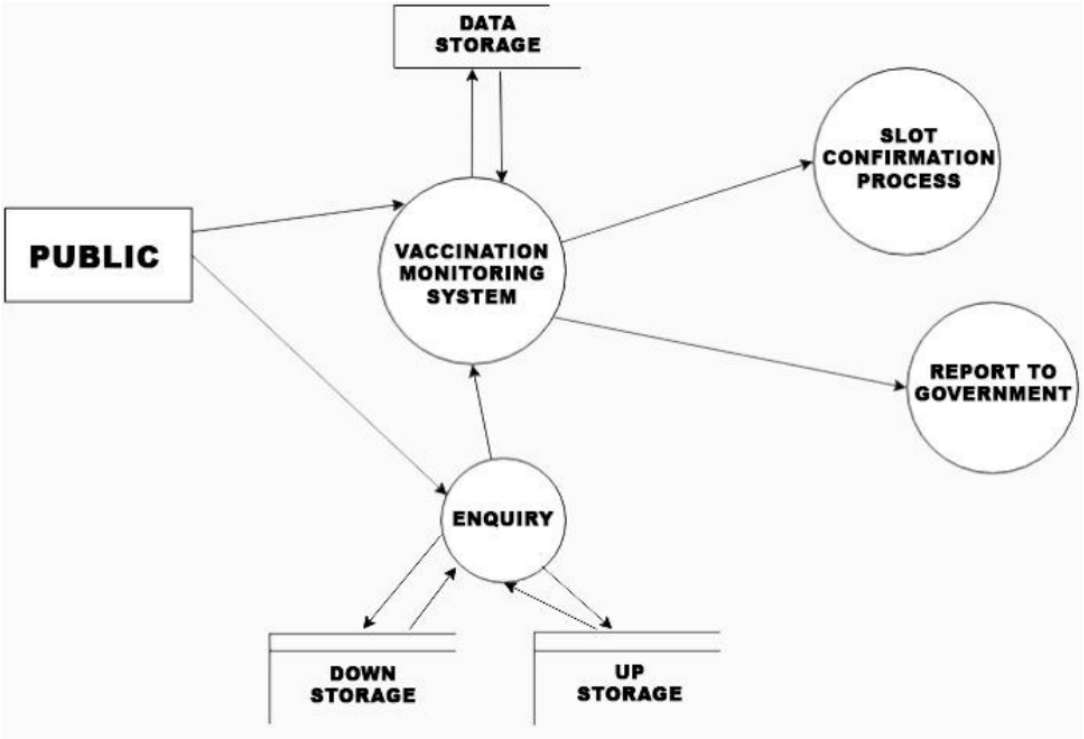
their context. It describes various kinds of objects and the static relationship between them.




## 5.5 Data Flow:

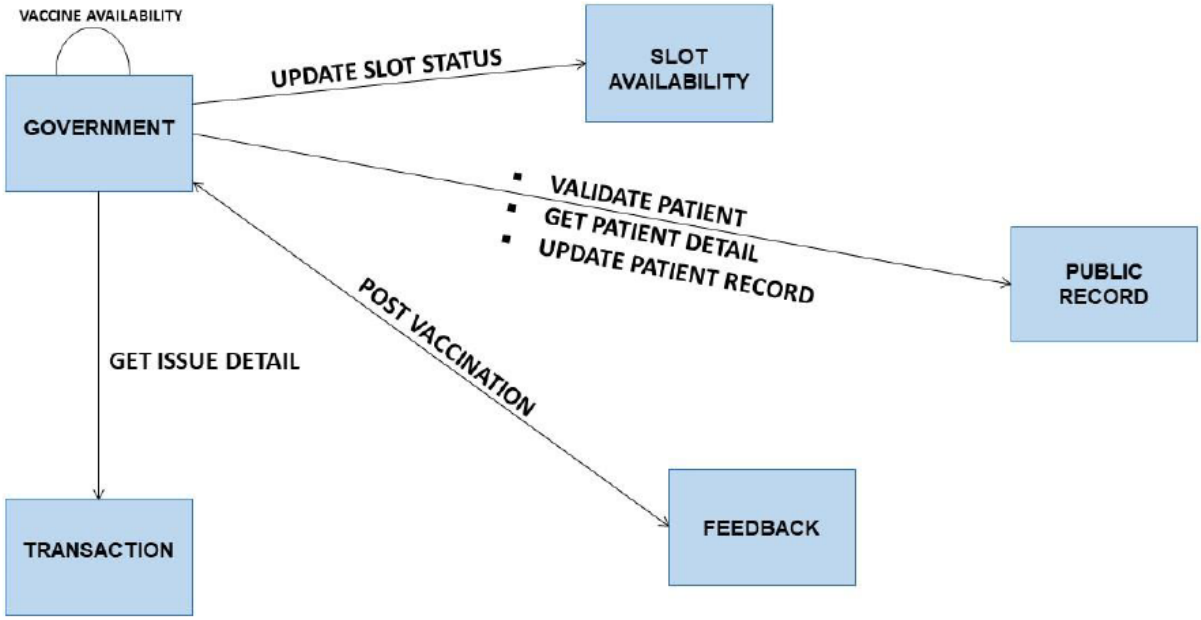
To analyse the flow of data through a system or a process. It also gives insight into the inputs and outputs of each entity and the process itself.






## 5.6 Communication Diagram:

Instead of showing the flow of messages, it depicts the architecture of the object residing in the system as it is based on object-oriented programming. Used to portray the object's architecture in the system.

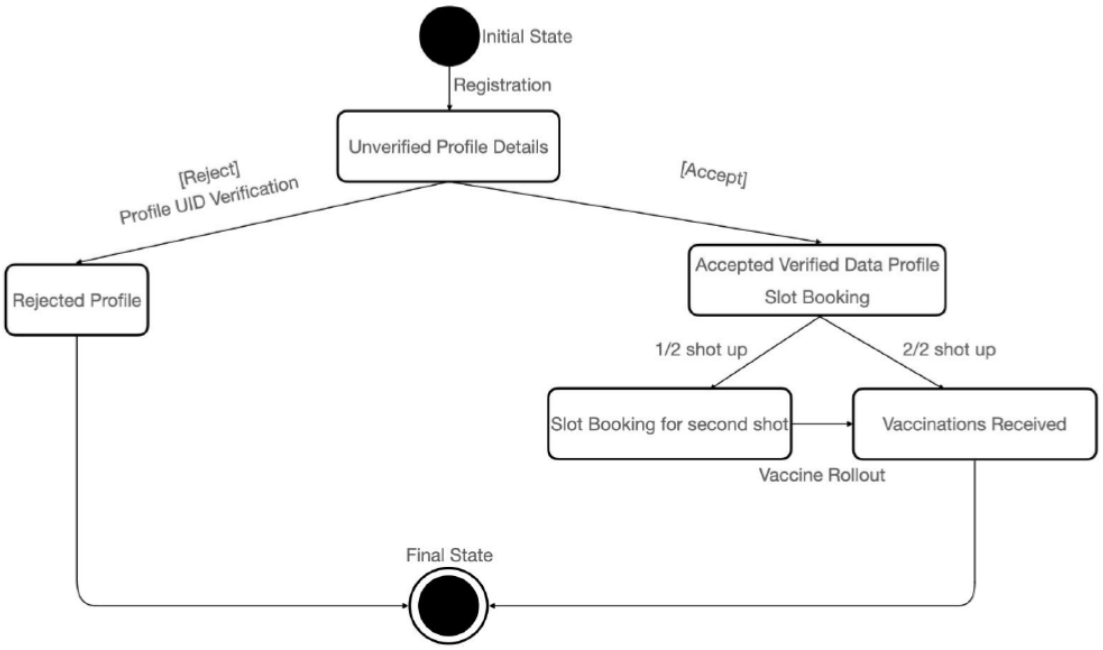


# CHAPTER 6

# BEHAVIORAL PATTERN

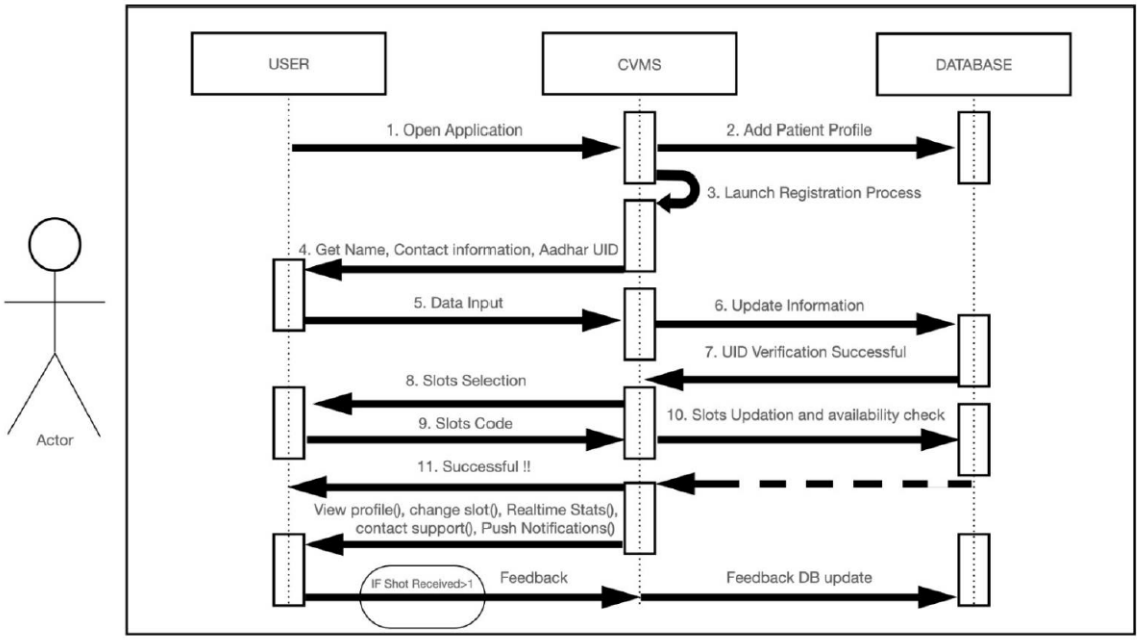
## 6.1 Software State:

A state diagram is used to represent the condition of the system or part of the system at finite instances of time. It's a behavioural diagram and it represents the behaviour using finite state transitions.



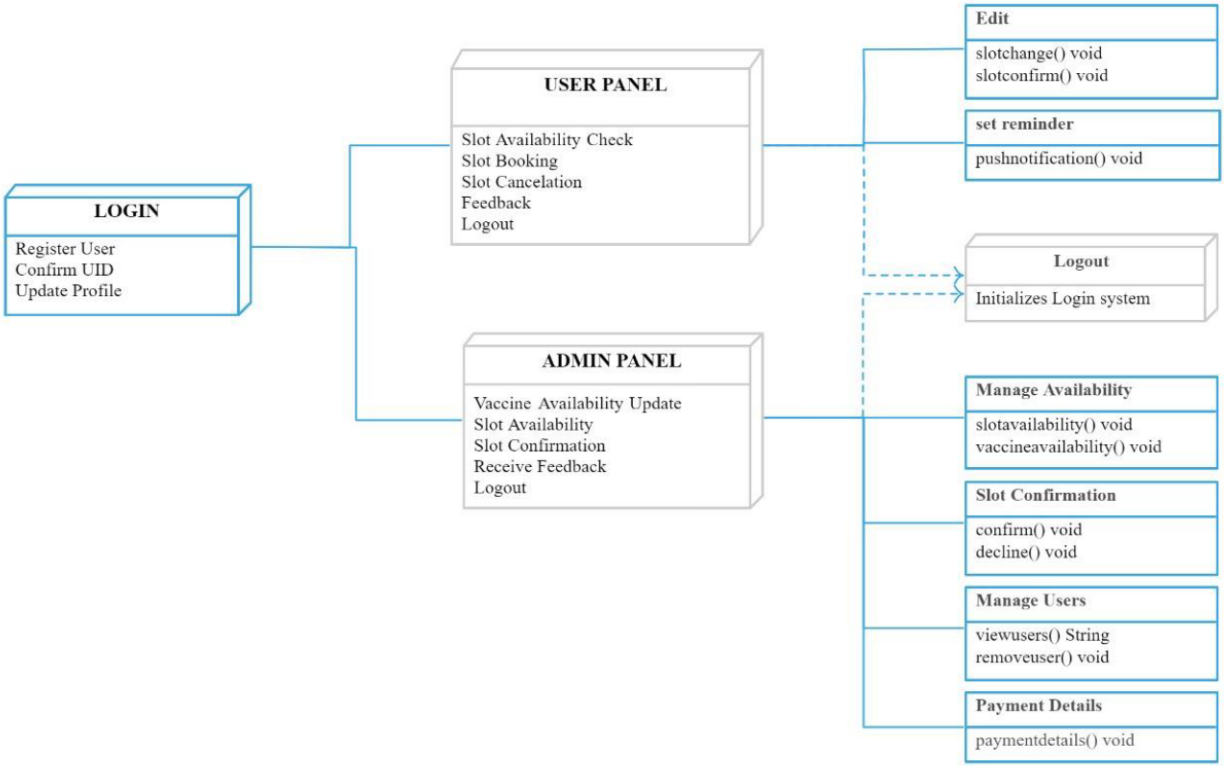
## 6.2 Event Scenarios:

Depicts the interaction between objects in a sequential order i.e., the order in which these interactions take place.



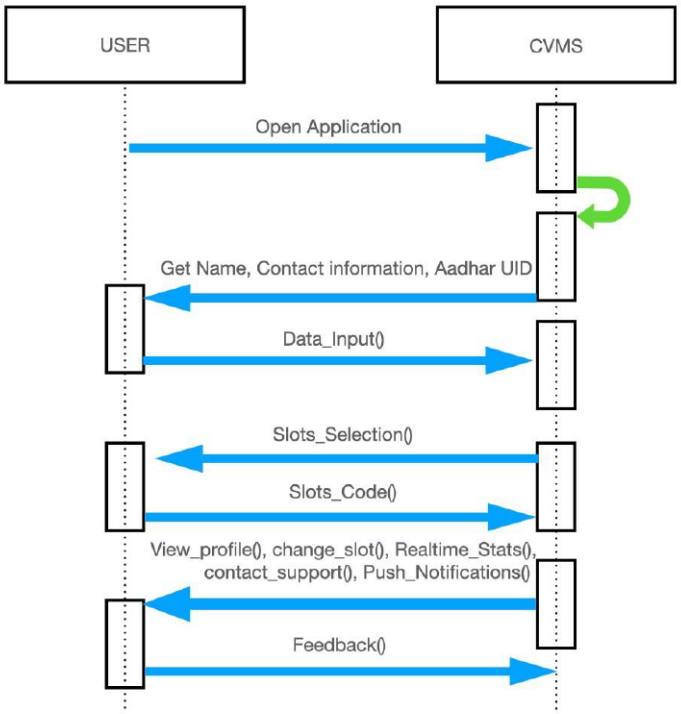

## 6.3 Deployment Diagramming:

In the context of the Unified Modelling Language (UML), a deployment diagram falls under the structural diagramming family because it describes an aspect of the system itself. It describes the physical deployment of information generated by the software program on hardware components.




## 6.4 Front End:

A Front-End diagram is a type of interaction diagram because it describes how a group of objects works together.



# CHAPTER 7

# MODULE IMPLEMENTATION

CVMS Project is coded in Python Language so Installing Tkinter Module provides a fast and easy way to create GUI applications

## 7.1 M1: Main Screen and New User Registration/Login

```
#import modules
from tkinter import *
import os
# (Design) window for registration
def register():
   global register_screen
   register_screen = Toplevel(main_screen)
   register_screen.title("Register")
   register_screen.geometry("300x400")
   global username
   global password
   global phone
   global address
   global aadharuid
   global username_entry
```

```
    global password_entry
    global phone_entry
    global address_entry
    global aadharuid_entry
    username = StringVar()
    password = StringVar()
    phone = StringVar()
    address = StringVar()
    aadharuid = StringVar()

    Label(register_screen, text="Please enter details below").pack()
    Label(register_screen, text="").pack()

    username_lable = Label(register_screen, text="Username")
    username_lable.pack()
    username_entry = Entry(register_screen, textvariable=username)
    username_entry.pack()

    password_lable = Label(register_screen, text="Password")
    password_lable.pack()
    password_entry = Entry(register_screen, textvariable=password, show='*')
    password_entry.pack()

    phone_lable = Label(register_screen, text="Phone Number")
    phone_lable.pack()
    phone_entry = Entry(register_screen, textvariable=phone)
    phone_entry.pack()

    address_lable = Label(register_screen, text="Address")
    address_lable.pack()
    address_entry = Entry(register_screen, textvariable=address)
    address_entry.pack()

    aadharuid_lable = Label(register_screen, text="12-Digits Aadhar UID")
    aadharuid_lable.pack()
    aadharuid_entry = Entry(register_screen, textvariable=aadharuid, show='*')
    aadharuid_entry.pack()

    Label(register_screen, text="").pack()
    Button(register_screen, text="Register", width=10, height=1, bg="green",
command = register_user).pack()
# (Design) window for login
def login():
    global login_screen
```

```
    login_screen = Toplevel(main_screen)
    login_screen.title("Login")
    login_screen.geometry("300x250")
    Label(login_screen, text="Please enter details below to login").pack()
    Label(login_screen, text="").pack()
    global username_verify
    global password_verify
    username_verify = StringVar()
    password_verify = StringVar()
    global username_login_entry
    global password_login_entry
    Label(login_screen, text="Username").pack()
    username_login_entry = Entry(login_screen, textvariable=username_verify)
    username_login_entry.pack()
    Label(login_screen, text="").pack()
    Label(login_screen, text="Password").pack()
    password_login_entry = Entry(login_screen, textvariable=password_verify,
show= '*')
    password_login_entry.pack()
    Label(login_screen, text="").pack()
    Button(login_screen, text="Login", width=10, height=1, command =
login_verify).pack()

# Implementing event on register screen "register" button
# storing data for new user registration
def register_user():
    username_info = username.get()
    password_info = password.get()
    phone_info = phone.get()
    address_info = address.get()
    aadharuid_info = aadharuid.get()
    file = open(username_info, "w")
    file.write(username_info + "\n")
    file.write(password_info + "\n")
    file.write(phone_info + "\n")
    file.write(address_info + "\n")
    file.write(aadharuid_info)

    file.close()
    username_entry.delete(0, END)
    password_entry.delete(0, END)
    phone_entry.delete(0, END)
    address_entry.delete(0, END)
    aadharuid_entry.delete(0, END)
```

```
    Label(register_screen, text="Registration Success", fg="green",
font=("calibri", 11)).pack()

# event on login button
def login_verify():
    username1 = username_verify.get()
    password1 = password_verify.get()
    username_login_entry.delete(0, END)
    password_login_entry.delete(0, END)

    list_of_files = os.listdir()
    if username1 in list_of_files:
        file1 = open(username1, "r")
        verify = file1.read().splitlines()
        if password1 in verify:
            login_sucess()
            slot()
        else:
            password_not_recognised()
    else:
        user_not_found()
# "login success" pop up design
def login_sucess():
    global login_success_screen
    login_success_screen = Toplevel(login_screen)
    login_success_screen.title("Success")
    login_success_screen.geometry("150x100")
    Label(login_success_screen, text="Login Success").pack()
    Button(login_success_screen, text="OK",
command=delete_login_success).pack()

# "invalid login password" pop up design
def password_not_recognised():
    global password_not_recog_screen
    password_not_recog_screen = Toplevel(login_screen)
    password_not_recog_screen.title("Success")
    password_not_recog_screen.geometry("150x100")
    Label(password_not_recog_screen, text="Invalid Password ").pack()
    Button(password_not_recog_screen, text="OK",
command=delete_password_not_recognised).pack()
# "user not found" pop up design
def user_not_found():
    global user_not_found_screen
    user_not_found_screen = Toplevel(login_screen)
```

```
    user_not_found_screen.title("Success")
    user_not_found_screen.geometry("150x100")
    Label(user_not_found_screen, text="User Not Found").pack()
    Button(user_not_found_screen, text="OK",
command=delete_user_not_found_screen).pack()
# Deleting popups
def delete_login_success():
    login_success_screen.destroy()
def delete_password_not_recognised():
    password_not_recog_screen.destroy()
def delete_user_not_found_screen():
    user_not_found_screen.destroy()
# Main window design
def main_account_screen():
    global main_screen
    main_screen = Tk()
    main_screen.geometry("400x400")
    main_screen.title("Covid Vaccination Management System")

    Label(text="Select Your Choice", bg="green", width="250", height="2",
font=("Calibri", 14)).pack()

    Label(text="").pack()
    Button(text="Login", height="3", width="30", command = login).pack()

    Label(text="").pack()
    Button(text="Register", height="3", width="30", command=register).pack()

    Label(text="").pack()
    Button(text="Active Covid Cases", height="3", width="30",
command=cases).pack()

    Label(text="").pack()
    Button(text="Support", height="3", width="30", command=support).pack()

    main_screen.mainloop()
main_account_screen()
```

## 7.2 M2: Real Time Active Covid Cases

```
#Driver Code
#importing modules for ACTIVE CASES IN VARIOUS COUNTRIES
from PyQt5.QtWidgets import *
from PyQt5 import QtCore, QtGui
from PyQt5.QtGui import *
from PyQt5.QtCore import *
from bs4 import BeautifulSoup as BS
import requests
import sys

casesData()

"""
code for ACTIVE CASES IN VARIOUS COUNTRIES
"""

class Window(QMainWindow):

    def __init__(self):
        super().__init__()

        # setting title
        self.setWindowTitle("Covid Vaccination Management System")
        # setting geometry
        self.setGeometry(100, 100, 400, 500)
        # calling method
        self.UiComponents()
        # showing all the widgets
        self.show()

    # method for widgets
    def UiComponents(self):

        # countries list // user can add other countries as well
        self.country = ["us", "india", "brazil", "france", "russia", "uk", "italy", "spain",
"mexico", "iran", "canada", "belgium", "pakistan", "bangladesh", "japan", "nepal",
"denmark"]
        # creating a combo box widget
        self.combo_box = QComboBox(self)
        # setting geometry to combo box
        self.combo_box.setGeometry(100, 50, 200, 40)
```

```
    # setting font
    self.combo_box.setFont(QFont('Times', 15))
    # adding items to combo box
    for i in self.country:
        i = i.upper()
        self.combo_box.addItem(i)
    # adding action to the combo box
    self.combo_box.activated.connect(self.get_cases)
    # creating label to show the total cases
    self.label_total = QLabel("Total Cases ", self)
    # setting geometry
    self.label_total.setGeometry(100, 300, 200, 40)
    # setting alignment to the text
    self.label_total.setAlignment(Qt.AlignCenter)
    # adding border to the label
    self.label_total.setStyleSheet("border : 2px solid black;")
    # creating label to show the recovered cases
    self.label_reco = QLabel("Recovered Cases ", self)
    # setting geometry
    self.label_reco.setGeometry(100, 350, 200, 40)
    # setting alignment to the text
    self.label_reco.setAlignment(Qt.AlignCenter)
    # adding border
    self.label_reco.setStyleSheet("border : 2px solid black;")
    # creating label to show death cases
    self.label_death = QLabel("Total Deaths ", self)
    # setting geometry
    self.label_death.setGeometry(100, 400, 200, 40)
    # setting alignment to the text
    self.label_death.setAlignment(Qt.AlignCenter)
    # adding border to the label
    self.label_death.setStyleSheet("border : 2px solid black;")


# method called by push
def get_cases(self):

    # getting country name
    index = self.combo_box.currentIndex()
    country_name = self.country[index]


    # creating url using country name
```

```
        url = "https://www.worldometers.info/coronavirus/country/" + country_name +
"/"
        # getting the request from url
        data = requests.get(url)
        # converting the text
        soup = BS(data.text, 'html.parser')
        # finding meta info for cases
        cases = soup.find_all("div", class_="maincounter-number")
        # getting total cases number
        total = cases[0].text
        # filtering it
        total = total[1: len(total) - 2]
        # getting recovered cases number
        recovered = cases[2].text
        # filtering it
        recovered = recovered[1: len(recovered) - 1]
        # getting death cases number
        deaths = cases[1].text
        # filtering it
        deaths = deaths[1: len(deaths) - 1]
        # show data through labels
        self.label_total.setText("Total Cases : " + total)
        self.label_reco.setText("Recovered Cases : " + recovered)
        self.label_death.setText("Total Deaths : " + deaths)


# create pyqt5 app
App = QApplication(sys.argv)

# create the instance of our Window
window = Window()

window.show()

# start the app
sys.exit(App.exec())
```

## 7.3 M3: Slot Booking

```
def register_slot():
    hospitalname_info = hospitalname.get()
    date_info = date.get()
    timeslot_info = timeslot.get()
    uid_info = uid.get()

    file = open(hospitalname_info, "w")
    file.write(hospitalname_info + "\n")
    file.write(date_info + "\n")
    file.write(timeslot_info + "\n")
    file.write(uid_info)



    file.close()
    hospitalname_entry.delete(0, END)
    date_entry.delete(0, END)
    timeslot_entry.delete(0, END)
    uid_entry.delete(0, END)

    Label(slot_screen, text="Registration Success", fg="green", font=("calibri",
11)).pack()

# popup for slot booking

def slot():
    global slot_screen
    slot_screen = Toplevel(login_screen)
    slot_screen.title("Slot Register")
    slot_screen.geometry("300x400")
    global hospitalname
    global date
    global timeslot
    global uid
    global hospitalname_entry
    global date_entry
    global timeslot_entry
    global uid_entry
    hospitalname = StringVar()
    date = StringVar()
    timeslot = StringVar()
    uid = StringVar()
```

```
  Label(slot_screen, text="Please enter details below").pack()
  Label(slot_screen, text="").pack()

  hospitalname_lable = Label(slot_screen, text="Hospital Name")
  hospitalname_lable.pack()
  hospitalname_entry = Entry(slot_screen, textvariable=hospitalname)
  hospitalname_entry.pack()

  date_lable = Label(slot_screen, text="Date")
  date_lable.pack()
  date_entry = Entry(slot_screen, textvariable=date)
  date_entry.pack()

  timeslot_lable = Label(slot_screen, text="Time Slot")
  timeslot_lable.pack()
  timeslot_entry = Entry(slot_screen, textvariable=timeslot)
  timeslot_entry.pack()

  uid_lable = Label(slot_screen, text="Enter Aadhar UID Code")
  uid_lable.pack()
  uid_entry = Entry(slot_screen, textvariable=uid)
  uid_entry.pack()

  Label(slot_screen, text="").pack()
  Button(slot_screen, text="Register", width=10, height=1, bg="green",
command = register_slot).pack()
```

## 7.4 M4: User Support

```
from tkinter import *
import webbrowser

def callback(url):
   webbrowser.open_new(url)

root = Tk()
link1 = Label(root, text="Covid Help Portal- Ministry of Health, GoI", fg="green",
cursor="hand2")
link1.pack()
link1.bind("<Button-1>", lambda e: callback("https://www.mohfw.gov.in"))
link2 = Label(root, text="Install Aarogya Setu App (Android)", fg="green",
cursor="hand2")
link2.pack()
link2.bind("<Button-1>", lambda e:
callback("https://play.google.com/store/apps/details?id=nic.goi.aarogyasetu&hl=en_
IN&gl=US"))
link3 = Label(root, text="Install Aarogya Setu App (iOS)", fg="green",
cursor="hand2")
link3.pack()
link3.bind("<Button-1>", lambda e:
callback("https://apps.apple.com/in/app/aarogyasetu/id1505825357"))
link4 = Label(root, text="Email Ministry of Health, GoI", fg="green",
cursor="hand2")
link4.pack()
link4.bind("<Button-1>", lambda e: callback("mailto:ncov2019@gov.in"))
link5 = Label(root, text="Helpline Numbers of States & Union Territories (UTs)",
fg="green", cursor="hand2")
link5.pack()
link5.bind("<Button-1>", lambda e:
callback("https://www.mohfw.gov.in/pdf/coronvavirushelplinenumber.pdf"))
link6 = Label(root, text="Consult a doctor", fg="green", cursor="hand2")
link6.pack()
link6.bind("<Button-1>", lambda e: callback("https://www.practo.com"))

root.mainloop()
```

# CHAPTER 8

# TEST CASE DESIGN

## 8.1 Objective:

To test the interface, front end and integration of all functions to find the smallest bug to rectify it to make the software error free to provide Cent efficient software for Covid Vaccine Monitoring to the Government and Public.

## 8.2 Testing Strategy:

Testing will be done by dividing software into small modules. And each module will be executed with all possible test cases.

## 8.3 Testing Criteria:

| **Module 1:** Main Screen, New user registration and login window popups |
| --- |
| **Test Case 1:**<br>Covid Vaccination Monitoring System main screen layout and interface.<br><br>**Test Case 2:**<br>User registration window + Detail fill up + Registration function check<br><br>**Test Case 3:**<br>User Login window<br>• ***Case 3.1:*** Successful login<br>• ***Case 3.2:*** Invalid Password<br>• ***Case 3.3:*** User not found |

| **Module 2:** Real time active Covid cases |
| --- |
| **Test case 1:**<br>Real time active cases window + Check accuracy by taking a data |

| **Module 3:** Slot Registration |
| --- |
| **Test case 1:**<br>Slot registration window + Data fill up + confirm slot registration |

| **Module 4:** Support |
| --- |
| **Test case 1:**<br>Check the hyperlinks and confirm the working conditions for the same. |

## 8.4 Dependencies:

**Module 1:** Depends on the user provided data.

**Module 2:** Depends on the statistics provided by various institutions.

**Module 3:** Depends on the slot availability and hospital limits.

**Module 4:** Depends on the working conditions of external sources.

## 8.5 Risks or Assumptions:

Slot unavailability can affect the working of module 3. Slow internet connections can Delay the opening of support websites which will affect module 4.

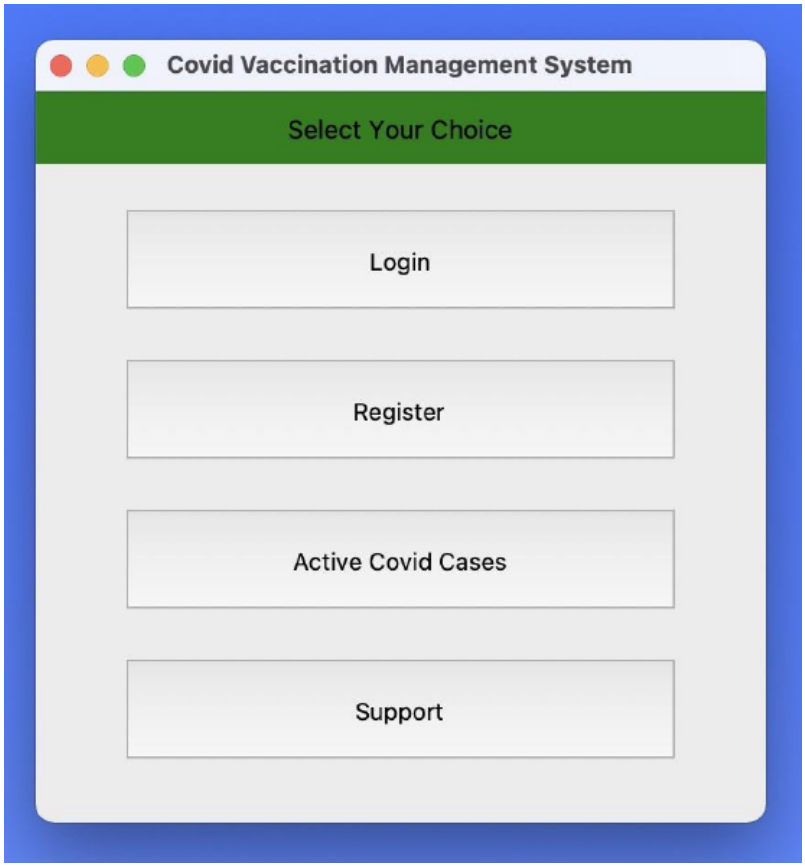
# CHAPTER 9

# MANUAL TESTING

## 9.1 Module 1 – Main Screen, New user registration and login window popups

### 9.1.1 Test Case 1:

Covid Vaccination Monitoring System main screen layout and interface.

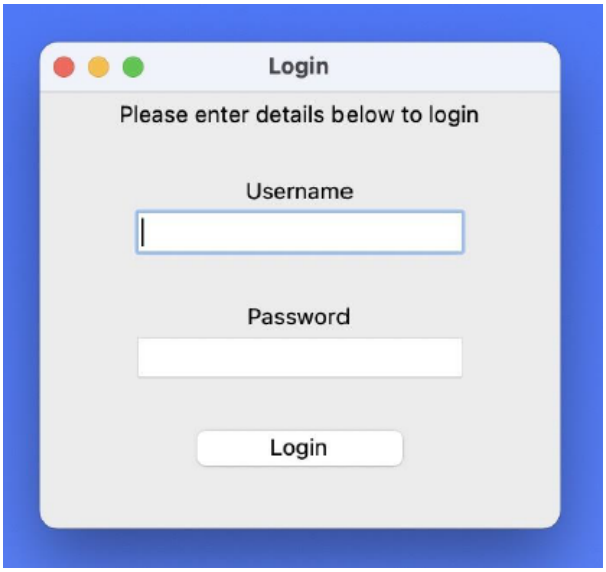


### 9.1.2 Test Case 2:

User registration window + Detail entry + Registration function check

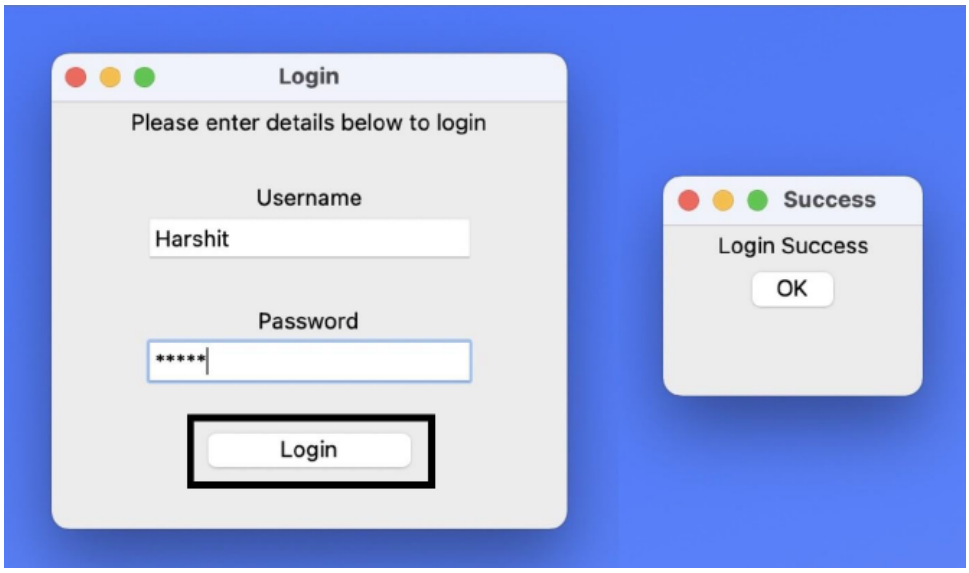

### 9.1.3 Test Case 3:

User Login window



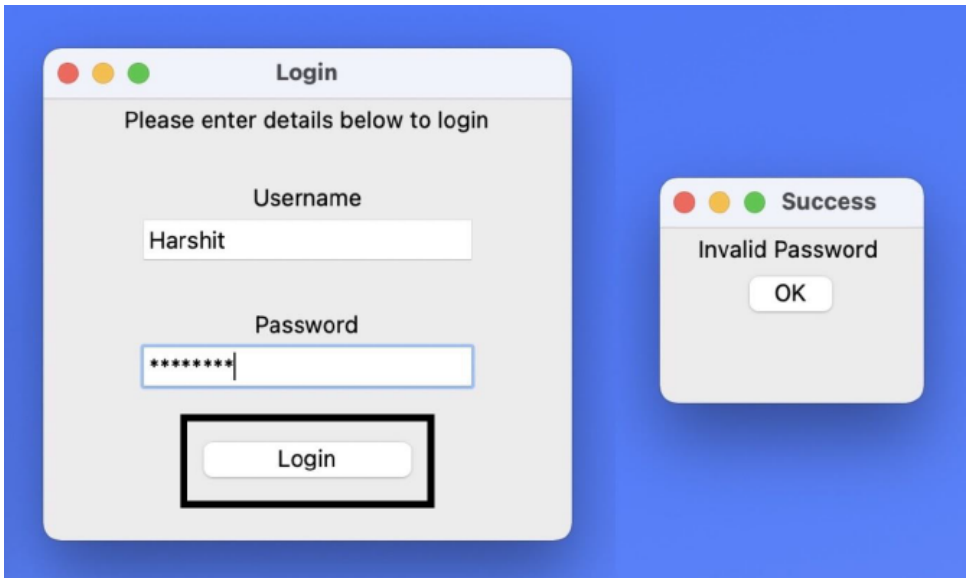
**Test Case 3.1:**

Successful login



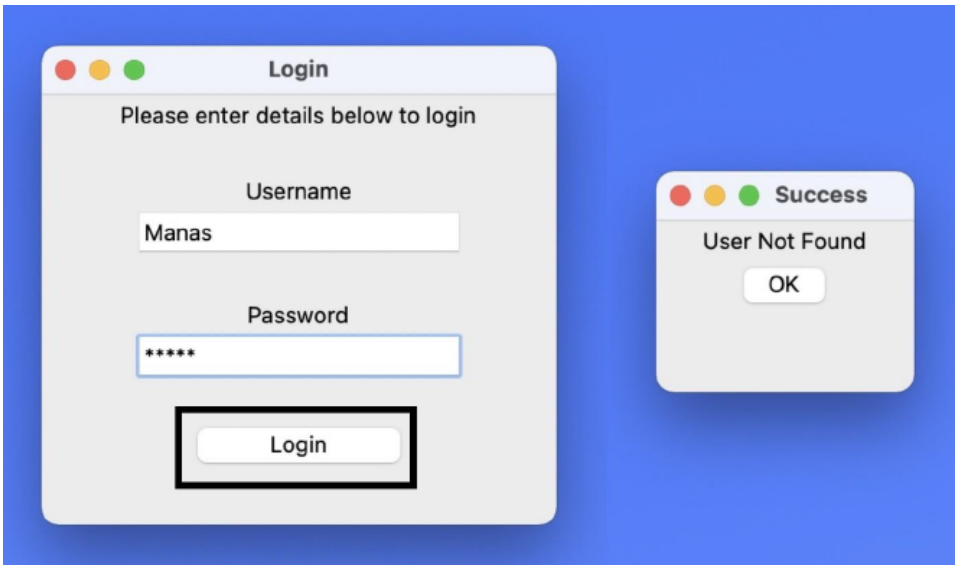

**Test Case 3.2:**

Successful login



**Test Case 3.3:**

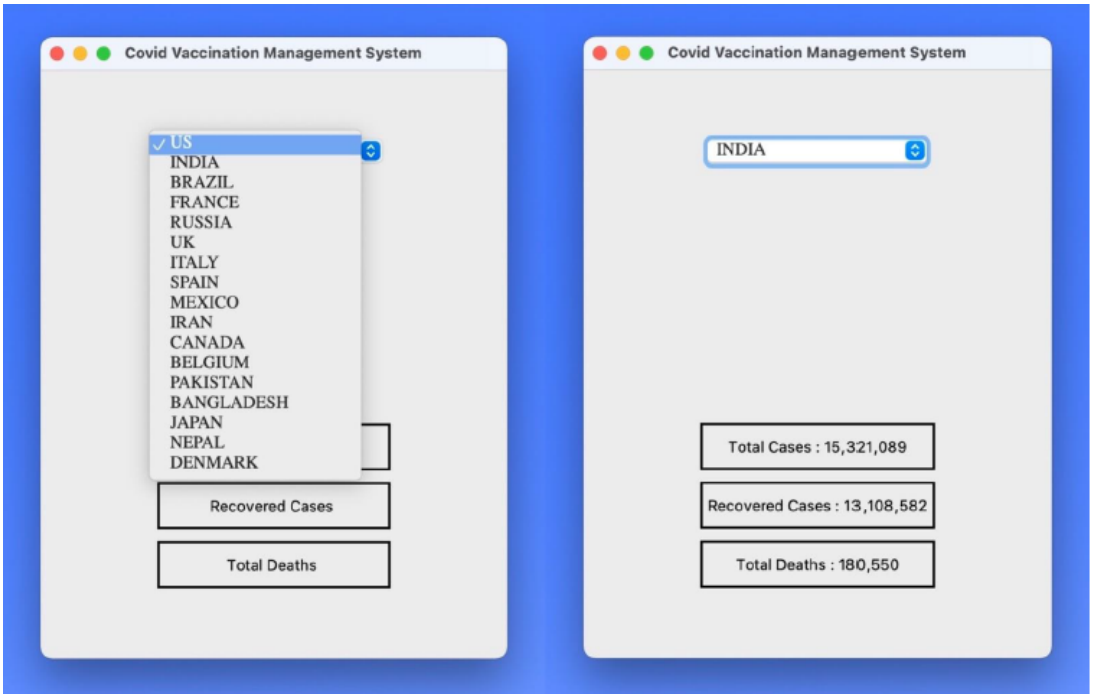
User not found




## 9.2 Module 2 – Real time active Covid cases

### 9.2.1 Test Case 1:

Real time active SARS COVID cases window



## 9.3 Module 3 – Slot Registration

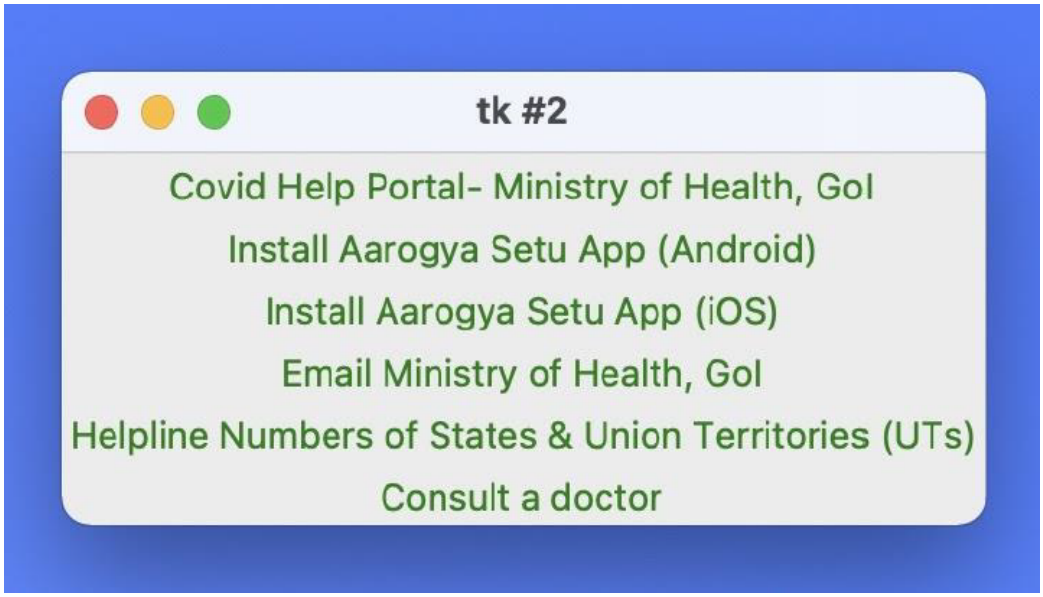
### 9.3.1 Test Case 1:

Slot registration window + Data fill up + confirm slot registration

## 9.4 Module 4 – Support

### 9.4.1 Test Case 1:

Check the hyperlinks and confirm the working conditions for the same.



# APPENDIX A

# VACCINATION MANAGEMENT SYSTEM

## A.1 Covid-19

Everyone, everywhere, should have access to COVID-19 vaccines. WHO is determined to maintain the momentum for increasing access to COVID-19 vaccines and will continue to support countries in accelerating vaccine delivery, to save lives and prevent people from becoming seriously ill. Countries should continue to work towards vaccinating at least 70 percent of their populations, prioritizing the vaccination of 100 percent of health workers and 100 percent of the most vulnerable groups, including people who are over 60 years of age and those who are immunocompromised or have underlying health conditions.

# REFERENCES

1. Clean Code: A Handbook of Agile Software Craftsmanship (Robert C. Martin Series)

2. Design Patterns: Elements of Reusable Object-Oriented Software (Addison-Wesley Professional Computing Series) (Old Edition)

3. https://youtu.be/uJpQlyT$_{C}K4$