SRM
INSTITUTE OF SCIENCE AND TECHNOLOGY
(Deemed to be University *u/s 3 of UGC Act, 1956)*
DELHI-NCR CAMPUS, GHAZIABAD (U.P)

SOFTWARE ENGINEERING AND PROJECT MANAGEMENT (18CSC206J)

(COVID-19 Vaccination Monitoring System)

Submitted by: **FIV07**

ISHAN THAKUR (RA1911003030068)

AKHIL ABHILASH (RA1911003030086)

MANAS SHARMA (RA1911003030090)

HARSHIT TYAGI (RA1911003030093)

SEM IV, Year- ll (2020-2021)

Under the supervision of

MR. SUNIL KUMAR

(Department of Computer Science and Engineering)

EXPERIMENT 1

Aim:

Identify the software Project, create business case, Arrive at a Problem Statement.

Requirements:

- Development Platforms
- Server
- Support
- Data (AUID, Location, Personal Info., Health History)

Software Overview:

This 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.

Business case:

Project Name:	CVMS (COVID-19 Vaccination Monitoring System)
Project 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.
Benefits:	• This project aims for a Vaccination monitoring system. • This will help in automating the process of categorizing the citizens. • Real time vaccination status. • Vaccination appointment tracking function for end users.
Timescale:	Initial analysis shows that the system will take approximately 10 - 12 weeks to implement.

Problem Statement:

Right now, the project looks pretty straightforward but there are still some unknowns surrounding implementation. There is also the risk that the project doesn’t meet the customer needs.

Result:

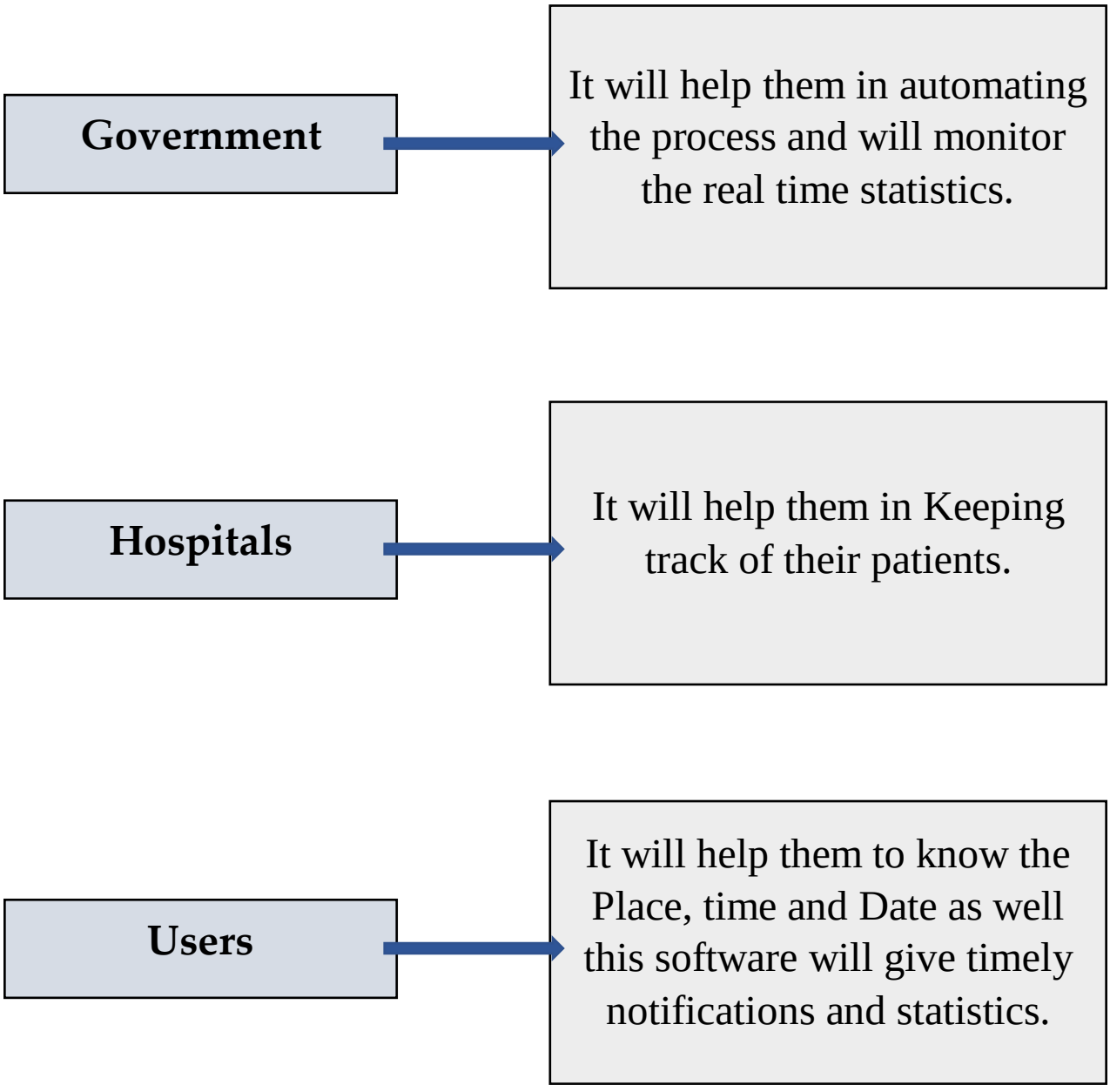
Experiment successfully executed. CVMS Project Introduction Completed.

EXPERIMENT 2

Aim:

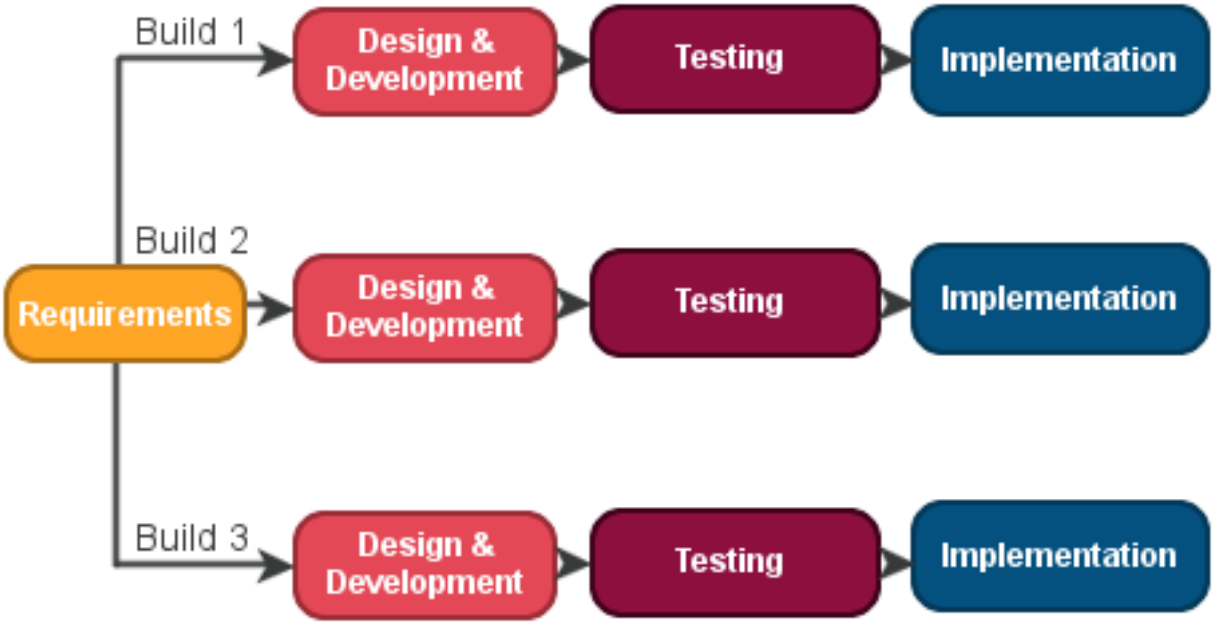
Stakeholder and User Description, Identify the appropriate Process Model, Comparative study with Agile Model.

Stakeholder:



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).



Comparative study with Agile Model:

Iterative Model	Agile Model
Tasks not divided in time boxes.	Tasks are divided in time boxes.
Software builds by taking a small set of requirements and enhancing it further until the entire product is ready to be deployed to the end-user or the customer.	Software builds and grows from the start of the project until delivering all at once near the end.
User gets a chance to experiment with a partially developed system.	Transfer of technology to new team members may be quite challenging due to lack of documentation.

Result:

This program has been successfully executed.

EXPERIMENT 3

Aim:

Identify the Requirements, System Requirements, Functional Requirements, Non-Functional Requirements.

Requirement:

- Window XP, Internet, MS Office Google etc.

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.

Software Requirement:

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

Functional Requirement:

The functional requirements the stakeholder requires:

- **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.

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.

Result:

The Program has been successfully executed.

EXPERIMENT 4

Aim:

Prepare project plan-based scope, find the job roles and responsibilities, and calculate project effort based on resources.

Project Plan:

- **Introduction:** Objective of this project is to simplify the COVID - 19 Vaccination process that's running throughout the world, and help people to keep track of their Vaccination Date, Time & Slot, Place and will also remind them to complete the process in time. It will also keep count on the total number and type of vaccines distributed.

  Constraints that can affect this project can be:

  a) Lack of internet.

  b) Lack of communication device.

  c) Lack of information.

- **Project organisation:** This project is managed by a private organization i.e., the developers of this project.
  Moreover, the feasibility of this project is efficient only when there is a clear understanding of the requirements.

- **Risk analysis:** Risks during the software development of this project can be:

  a) Risk of stealing the design

  b) Risk of system crash (rare case) We can access this factor of risk by protecting the data of the software in an encrypted account.

  Moreover, by using appropriate software process model we can decrease the rate of risk involved in the project.

- **Resource Management:**
  - **Hardware requirements:**

  a) Computer / Smart Phone.
  b) Multiprocessing system required.

  - **Software requirements:**

  a) Software process model.
  b) An IDE for implementing code.
  c) Blender or other software

- **Cost factor:** The total cost factor that can hamper requirements is around 5000 rupees (excluding the price of cloud services) or may be free if all the software and hardware on which we are working are freely accessible.

- **Work breakdown:** We can break our project into small sub parts or modules that are:

  a) Software layout

  b) Software design

c) Software coding

d) Software testing

e) Software deployment

f) Software maintenance

- **Project schedule:** Here, all the modules are interrelated with each other.
  Moreover, the design is important as it will decide the further testing of the code behind the layout and as if the testing is 100% success then we can deploy the project without any hesitation.

Job roles and Responsibilities:

Job Roles	Responsibilities
Developer	• Develop the software. • Writing and implementing efficient code • Deploying software tools, processes, and metrics • Working closely with other developers, UX designers, business and systems analysts

Designer	• Design the layout and interface of the software. • Design, develop and execute unit test plans, test designs, test cases and test strategies. • Design, develop and execute subsystem test plans, procedures and processes. • Document all test plans, test cases and strategies procedures and issues. • Design and implement test scripts on test tools and scripting languages. • Design, develop and implement program and process improvements. • Design and develop coding, code reviews, unit testing and release management. • Develop design specifications in accordance with business requirements and issues. • Recommend strategic improvements to optimize performances. • Perform analyses and interpretations of strategies and software

Programmer	• Creating software programs, integrating systems and software, training end-users, analysing algorithms, modifying source-code, writing system instructions, debugging, and maintaining operating systems.
Maintenance	• Maintain server and software after deployment.
Customer support	• Assisting the user regarding any queries. And also, to take feedbacks related to the software interface and working.
Accountant	• Manage the resources and for audit purposes.

Result:

The Program has been successfully executed.

EXPERIMENT 5

Aim:

Prepare work breakdown structure based on timelines, risk identification and plan.

Requirements:

- Proper knowledge of project
- Structure for work breakdown
- Knowledge about making diagrams

Procedure:

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.

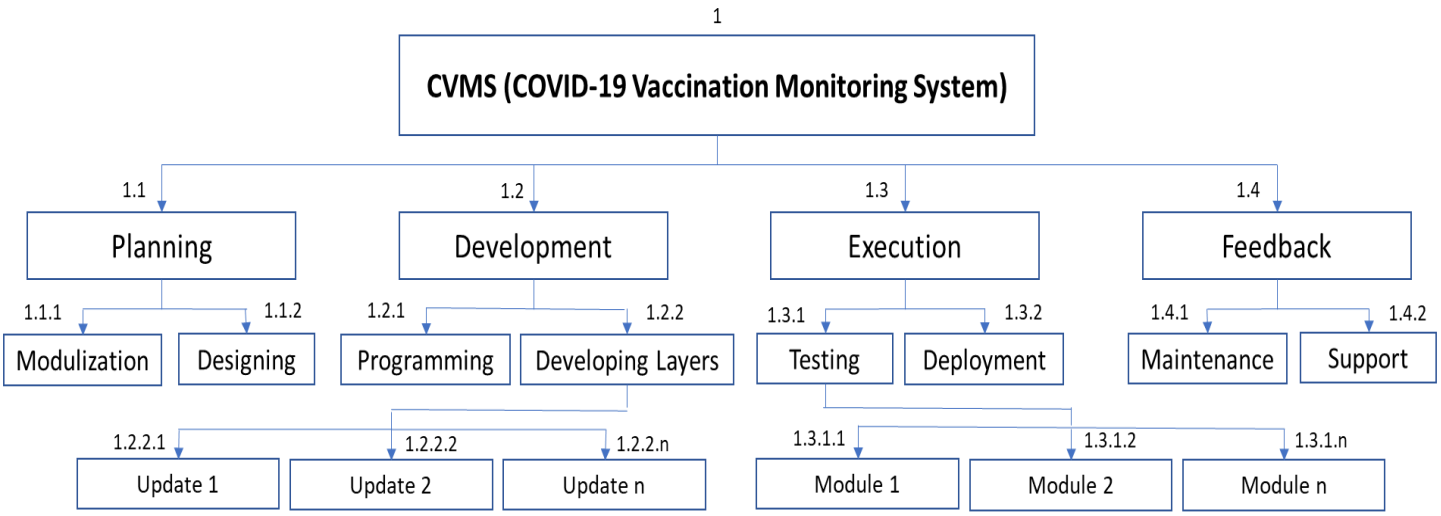
For this we need to follow the following three steps:

Step 1: Identify the major functions of the software.

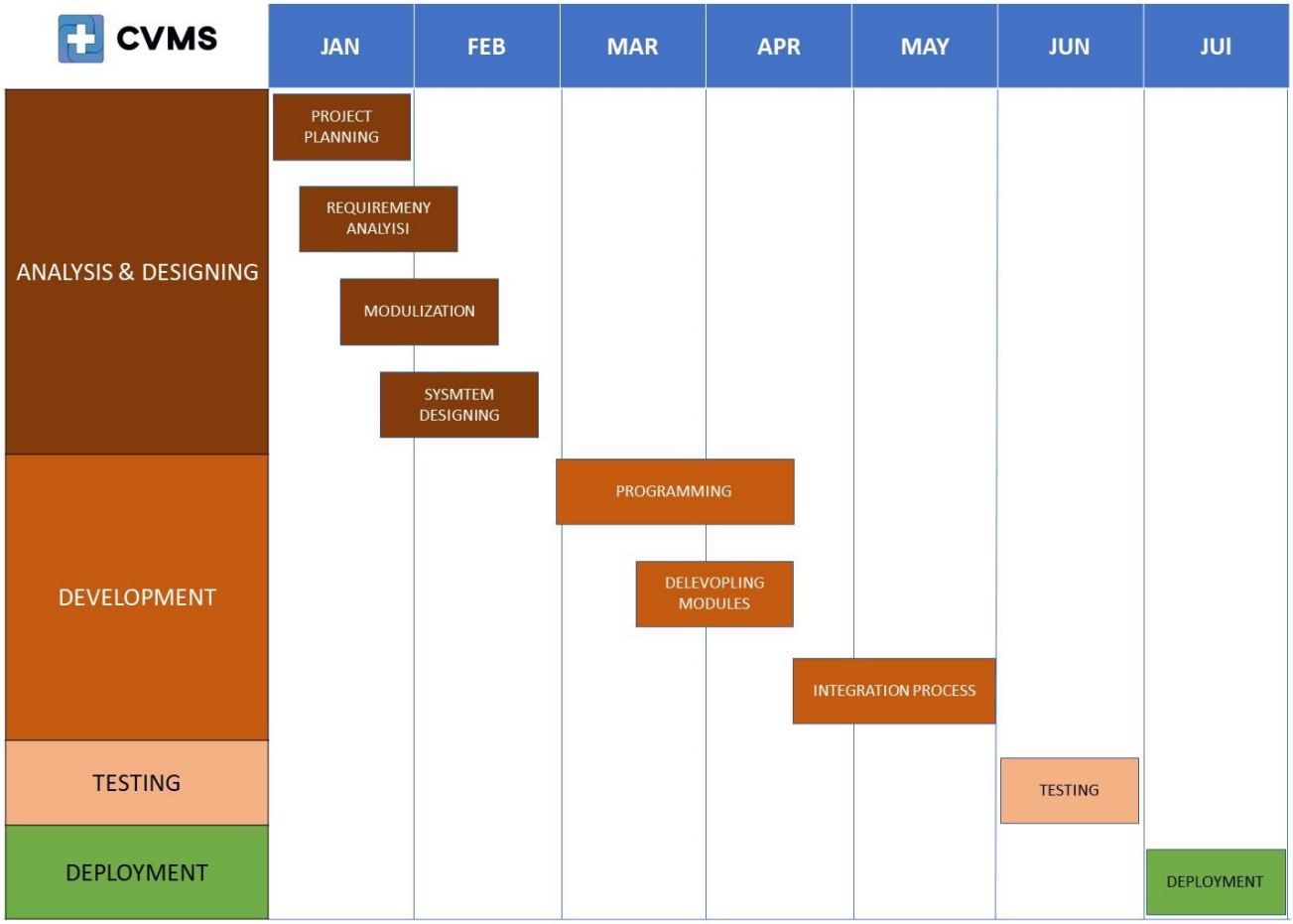
Step 2: Identify the sub-functions of the major functions.

Step 3: Repeat till undividable, simple and independent activities are created.

Work Breakdown Structure:



Timeline:



Output:

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.

A project timeline is a visual list of tasks or activities placed in chronological order, which lets project managers view the entirety of the project in one place.

A risk is an event that may or may not happen, but could have a significant effect on the outcome of a project, if it were to occur.

Result: The Program has been successfully executed.

EXPERIMENT 6

Aim:

Design a System Architecture, Use Case Diagram, ER Diagram, ER Diagram (Database), DFD Diagram (process) (Up to Level 1), Class Diagram (Applied for OOPs based project), Collaboration Diagram (Applied for OOPS based Project) (Software – Rational rose)

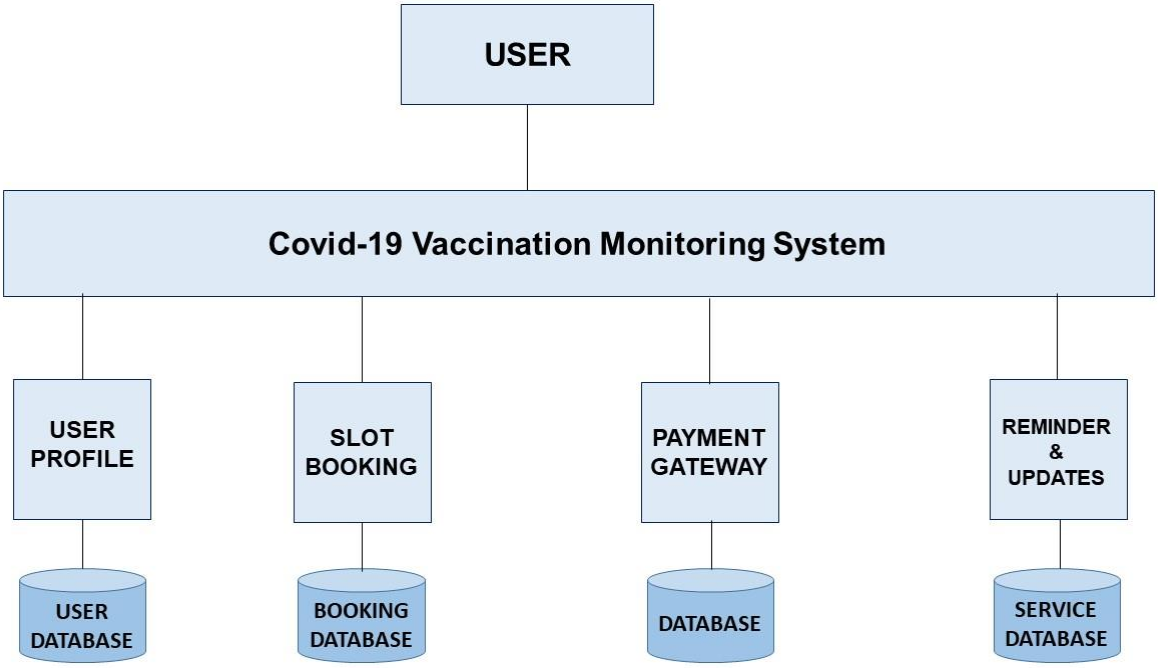
Requirements:

Windows, Internet, MSO, Google Search Engine and Lucidchart.

Problem Description:

System Architecture Diagram:

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.



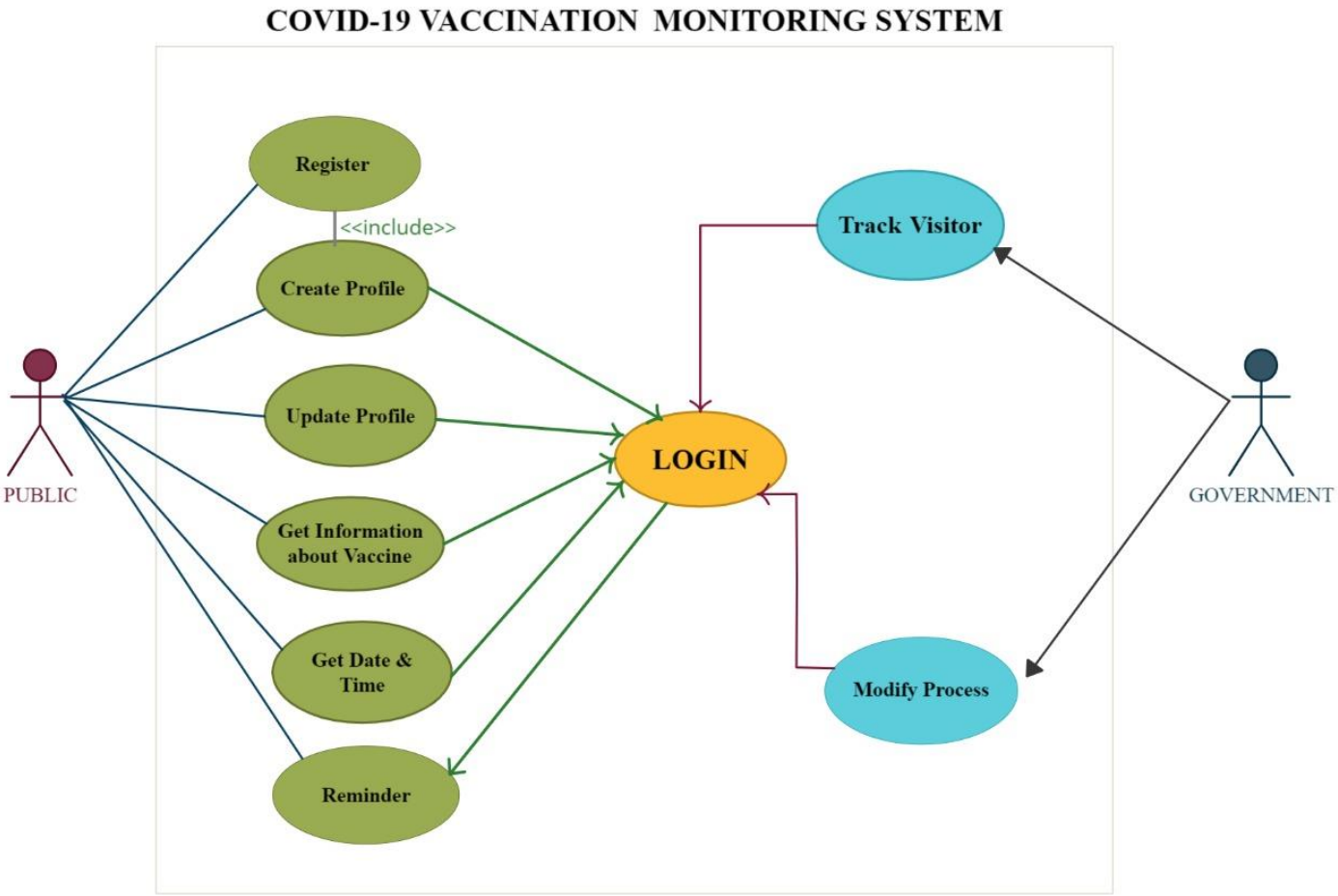
Use Case Diagram:

In the Unified Modeling Language (UML), a use case diagram can summarize the details of your system's users (also known as actors) and their interactions with the system.

To build one, you'll use a set of specialized symbols and connectors.

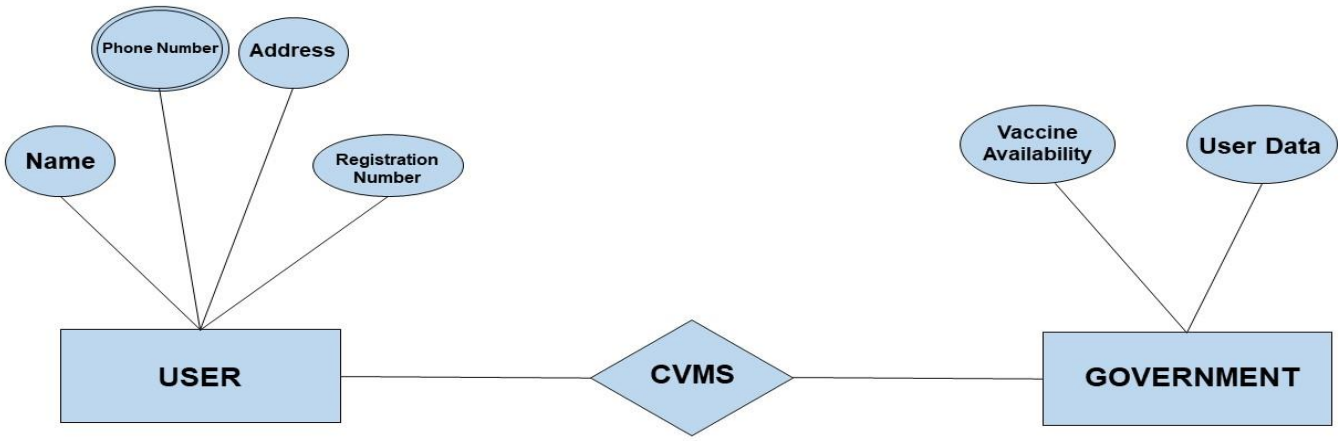
An effective use case diagram can help your team discuss and represent:

- Scenarios in which your system or application interacts with people, organizations, or external systems
- Goals that your system or application helps those entities (known as actors) achieve
- The scope of your system



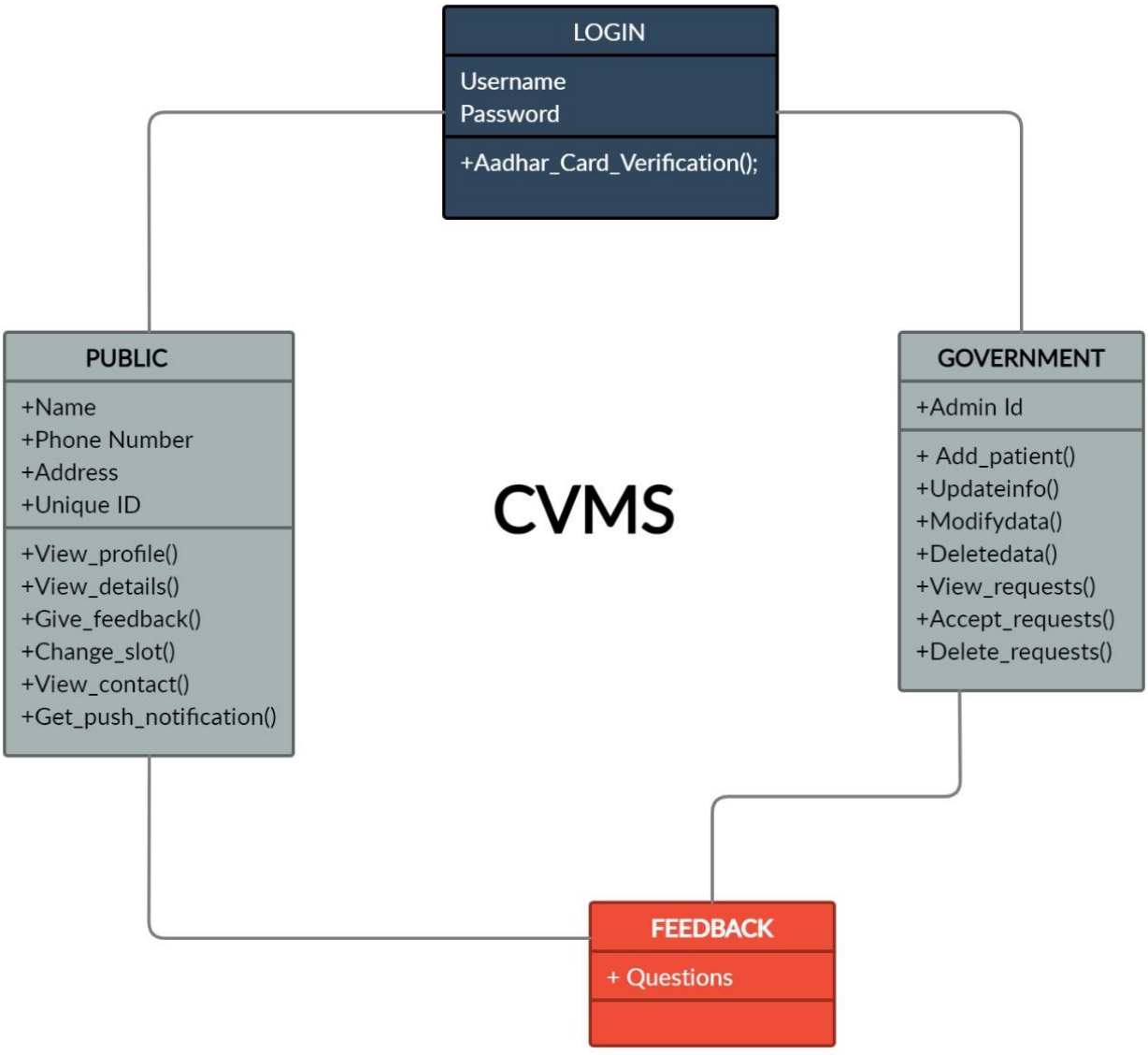
ER Diagram:

An Entity Relationship (ER) Diagram is a type of flowchart that illustrates how “entities” such as people, objects or concepts relate to each other within a system. ER Diagrams are most often used to design or debug relational databases in the fields of software engineering, business information systems, education and research. Also known as ERDs or ER Models, they use a defined set of symbols such as rectangles, diamonds, ovals and connecting lines to depict the interconnectedness of entities, relationships and their attributes. They mirror grammatical structure, with entities as nouns and relationships as verbs.



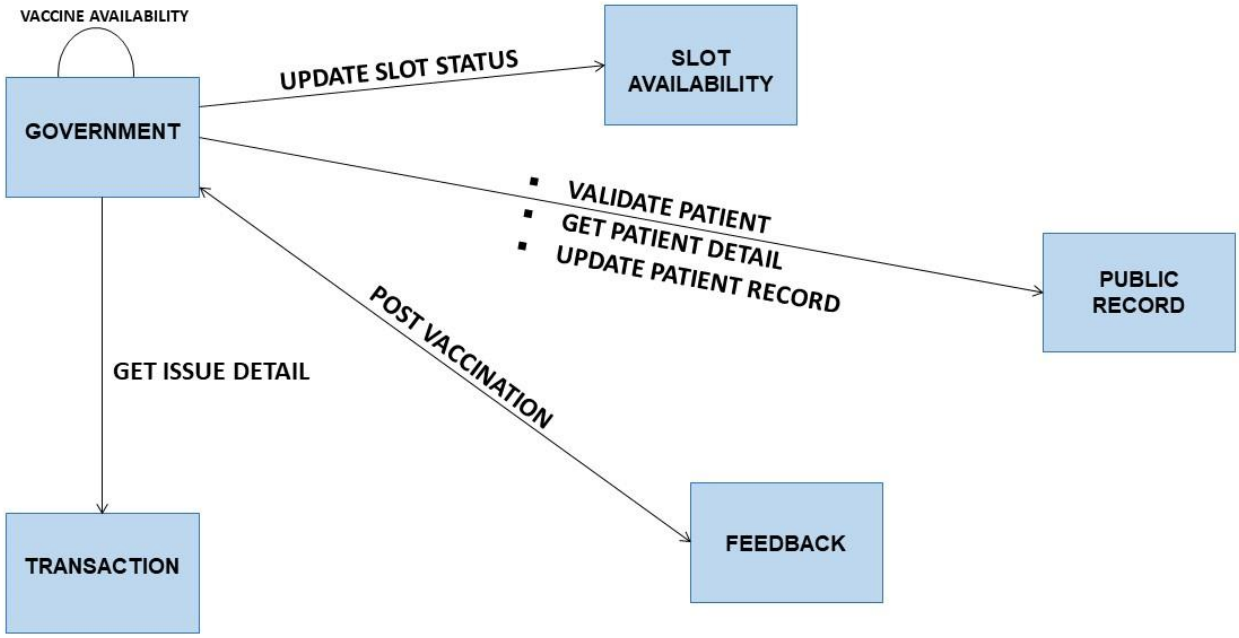
Class Diagram:

Class diagrams 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.



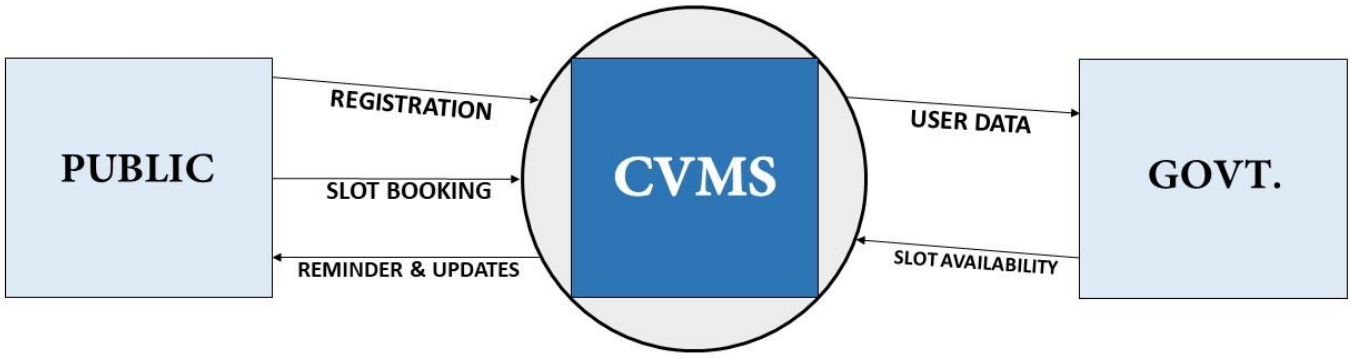
Collaboration Diagram:

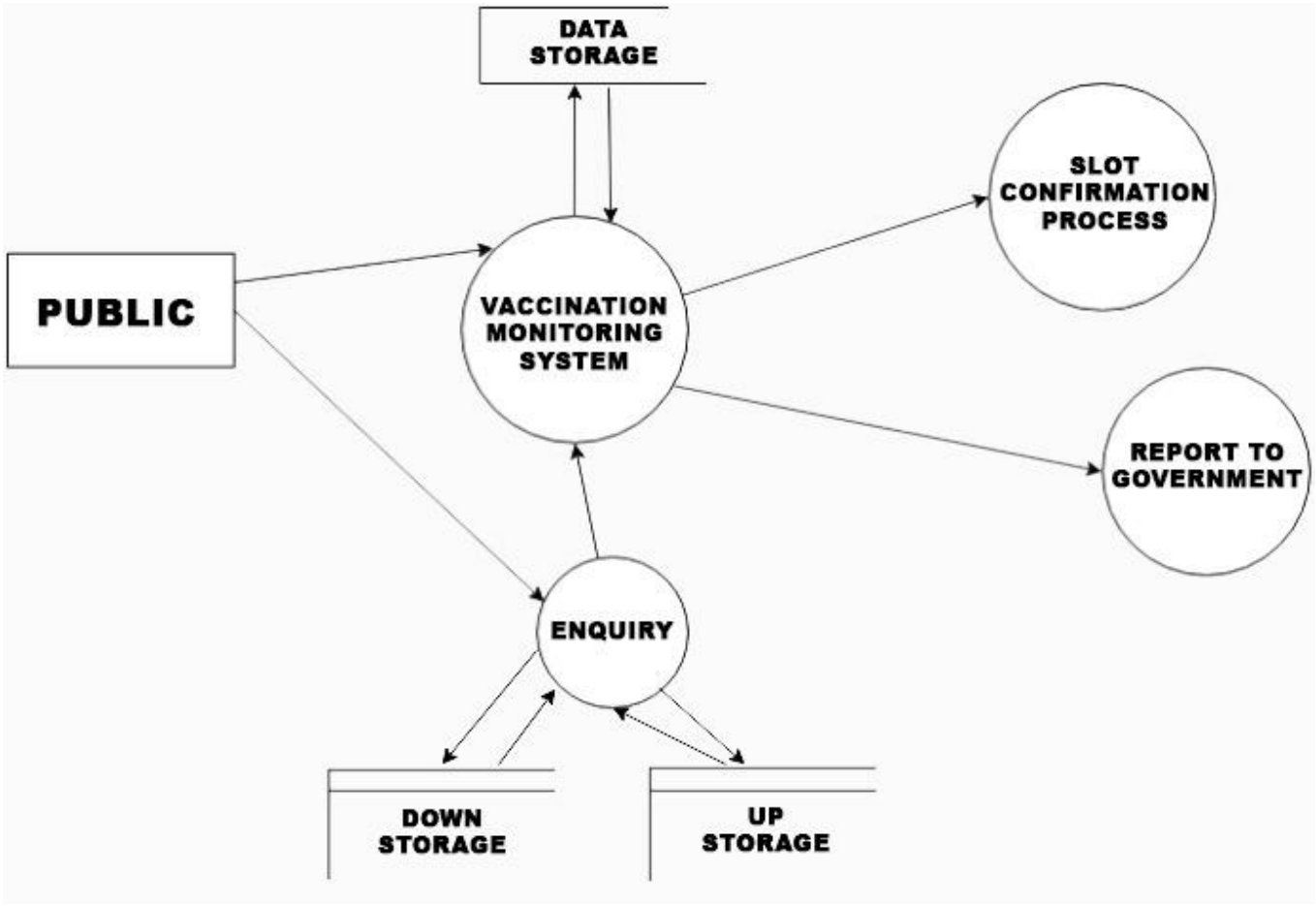
The collaboration diagram is used to show the relationship between the objects in a system. Both the sequence and the collaboration diagrams represent the same information but differently. Instead of showing the flow of messages, it depicts the architecture of the object residing in the system as it is based on object-oriented programming. An object consists of several features. Multiple objects present in the system are connected to each other. The collaboration diagram, which is also known as a communication diagram, is used to portray the object's architecture in the system.



DFD Diagram:

DFD is the abbreviation for Data Flow Diagram. The flow of data of a system or a process is represented by DFD. It also gives insight into the inputs and outputs of each entity and the process itself. DFD does not have control flow and no loops or decision rules are present. Specific operations depending on the type of data can be explained by a flowchart. Data Flow Diagram can be represented in several ways. The DFD belongs to structured-analysis modeling tools. Data Flow diagrams are very popular because they help us to visualize the major steps and data involved in software-system processes.





Result: The Program has been successfully executed.

EXPERIMENT 7

Aim:

Design State and Sequence Diagram, Deployment Diagram and Front-End Diagram.

Requirements:

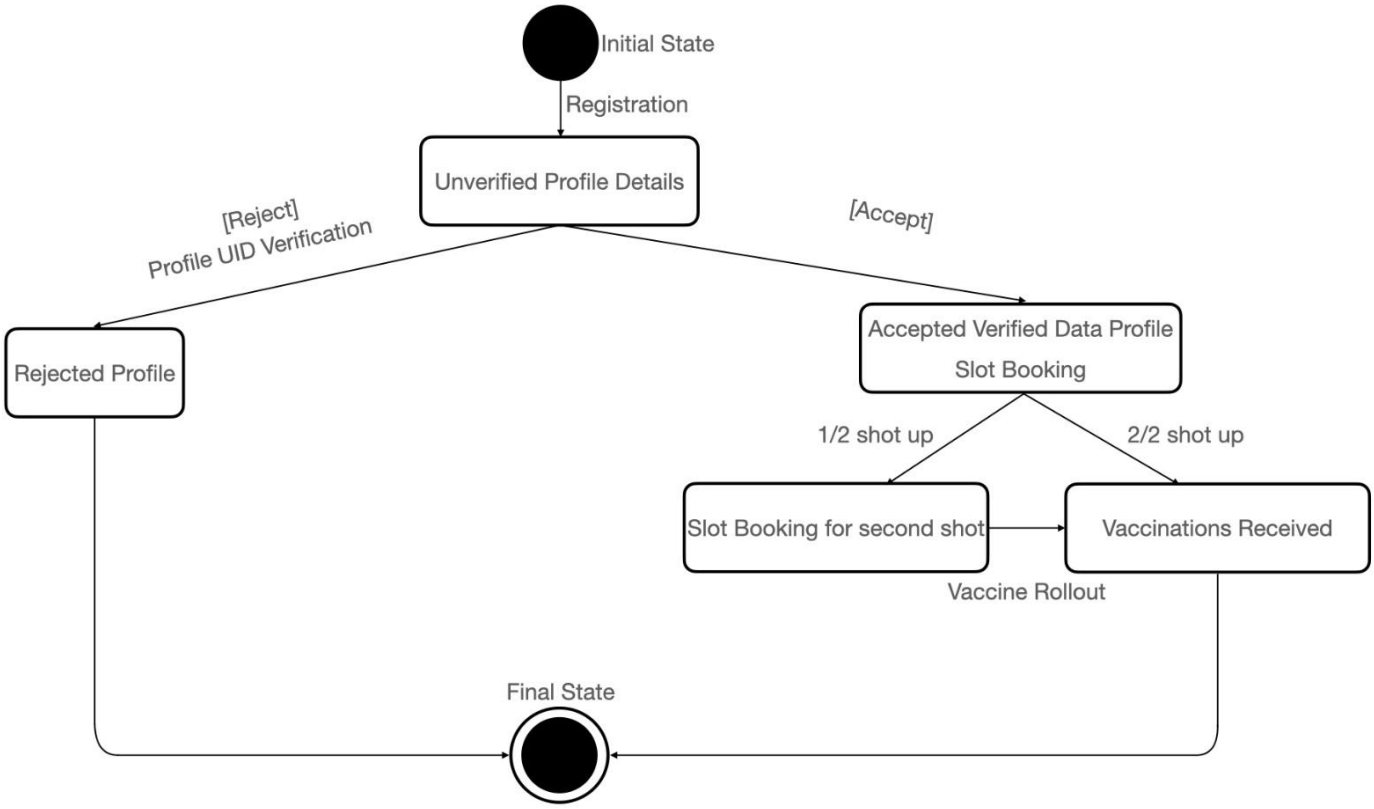
Windows, Internet, MSO, Google Search Engine and Lucidchart.

Problem Description:

- **State Diagram:**
  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.

  So simply, a state diagram is used to model the dynamic behaviour of a class in response to time and changing external stimuli.

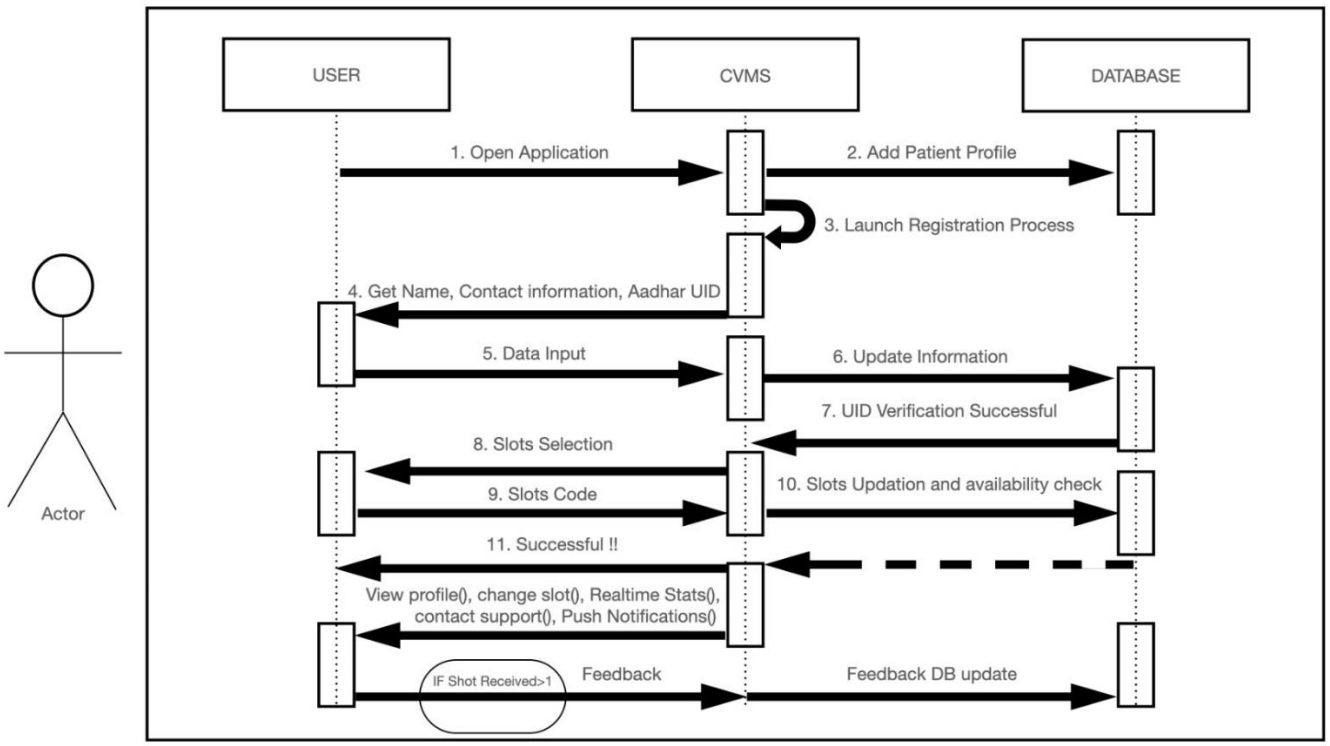


- **Sequence Diagram:**

  A sequence diagram simply depicts interaction between objects in a sequential order i.e., the order in which these interactions take place.

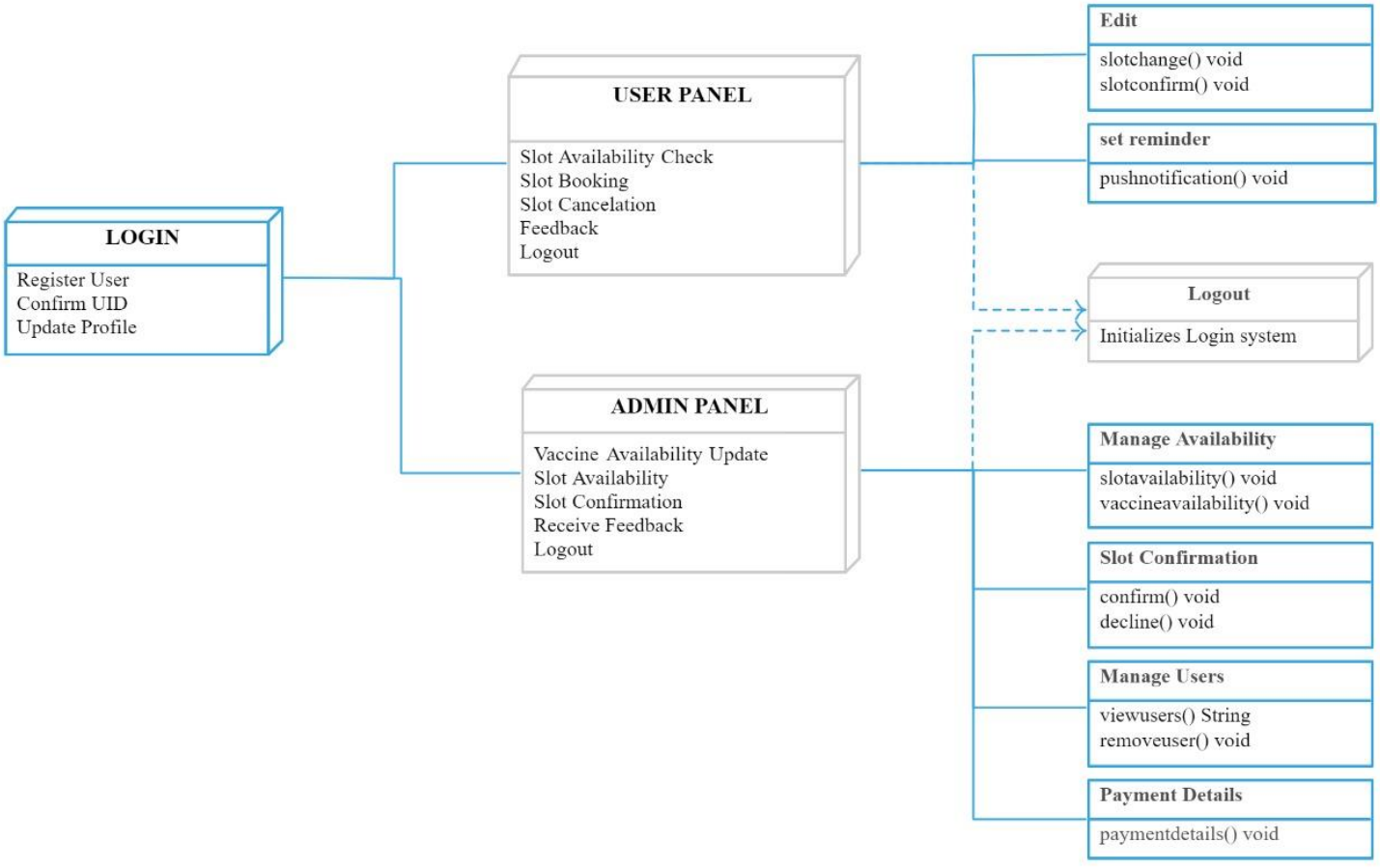
  We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function.

  These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.

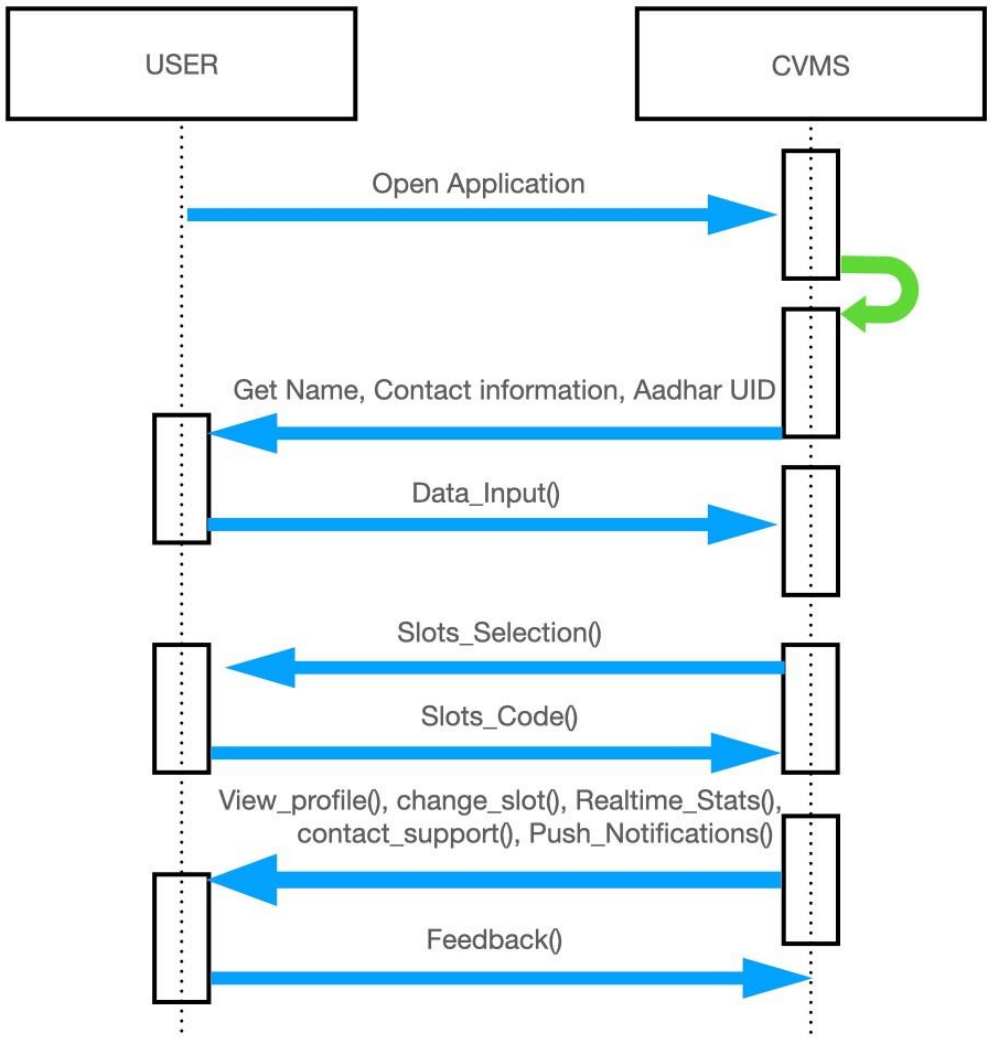


- **Deployment Diagram:**
  In the context of the Unified Modelling Language (UML), a deployment diagram falls under the structural diagramming family because it describes an aspect of the system itself. In this case, the deployment diagram describes the physical deployment of information generated by the software program on hardware components.
  The information that the software generates is called an artifact. This shouldn't be confused with the use of the term in other modelling approaches like BPMN.

Deployment diagrams are made up of several UML shapes. The three-dimensional boxes, known as nodes, represent the basic software or hardware elements, or nodes, in the system. Lines from node to node indicate relationships, and the smaller shapes contained within the boxes represent the software artifacts that are deployed.



- **Front End Diagram:**
  A Front-End diagram is a type of interaction diagram because it describes how a group of objects works together. These diagrams are used by software developers and business professionals to understand requirements for a new system or to document an existing process. Sequence diagrams are sometimes known as event diagrams or event scenarios.



Result: The Program has been successfully executed.

EXPERIMENT 8

Aim:

Module Description, Module Implementation (phase Using Agile)

Requirements:

Window 10, Internet, MS Office, vsCODE, Pycharm, git and python modules.

Problem Description:

Module Description and Implementation:

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.

MODULE 1: Main screen and new user registration and login window popups.

Modules required and Installation:

Tkinter: Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit. Creating a GUI application using Tkinter is an easy task.

Implementation steps:

1. Import tkinter.
2. So first of all you have to design the main screen. This screen have two buttons Login and Register.
3. Design a new screen for registration. That means if a user press register button on main screen then a new window will appear where users have to enter username, password, contact number, aadhar uid code, address. And this way they can register themselves.
4. And now we have to add two things inside the main_account_screen() method. So, as user clicks register button on main window(first) then a new screen will be appear where user have to enter their entry.
5. Now we have to implement event on register button. It means, after filling the entries, as soon as the register button is pressed, entries are saved in a file. So let's see how to do it. And now we have to declare username, password, phone, aadharuid, address username_entry, password_entry, phone_entry, aadharuid_entry, address_entry as global so

add following codes inside register() function. And add one more thing inside register() function. And now we will finally test our registration process. So fill the username and password field in register form and press register button. You will find, a Registration Success message on screen. And after registration a text file has been created which contain user's informations such as username and password.

6. We have seen register process, now we have to implement login process. So for this, first of all we have to design a login window. It is same as register window but having little changes. And now add a line inside main_account_screen().
7. Now we will define a function that will show a popup for successful login. If user has entered the valid entries then this popup will appear. If user enter wrong password then a popup for invalid password will appear. If user enter wrong username then a popup for User Not Found will appear. And now define a function for deleting this popup.

Implementation:

```
#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()
```

MODULE 2: Real Time Active Covid Cases

Modules required and Installation:

PyQt5: PyQt is a Python binding of the cross-platform GUI toolkit Qt, implemented as a Python plug-in. PyQt is free software developed by the British firm Riverbank Computing.

Requests: Requests allows you to send HTTP/1.1 requests extremely easily. There's no need to manually add query strings to your URLs.

Beautiful Soup: Beautiful Soup is a library that makes it easy to scrape information from web pages. It sits atop an HTML or XML parser, providing Pythonic idioms for iterating, searching, and modifying the parse tree.

Implementation steps:

1. Create a non-editable combo box, set its geometry
2. Create country list which contain countries name and add it to the combo box
3. Create three labels to show information about total cases, recovered cases and death cases
4. Set geometry, alignment and border to each label
5. Add action to the combo box
6. Inside the action get the country name and scrap the data from the website with the help of requests BeautifulSoup
7. Convert the raw data into html code and then filter it to get the required output
8. Show the output the screen with the help of labels

Implementation:

```
#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())
```

MODULE 3: Slot Booking

Modules required and Installation:

Tkinter: Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit. Creating a GUI application using Tkinter is an easy task.

Implementations steps:

1. Import tkinter.
2. Design a new slot registration. And this way they can register themselves for vaccination.
3. And now we have to add two things inside the login screen() method. So, as user login successfully then a new screen will be appear where user have to enter their entry.
4. Now we have to declare hospitalname, date, timeslot, uid, hospitalname_entry, date_entry, timeslot_entry, uid_entry as global so add following codes inside register_slot() function. After registration a text file has been created which contain user's informations.

Implementation:

```
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()
```

MODULE 4: User Support

Modules required and Installation:

Tkinter: Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit. Creating a GUI application using Tkinter is an easy task.

Implementations steps:

1. Import tkinter.
2. Design a support window containing all the hyperlinks. And this way users can get support for themselves.

Implementation:

```
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()
```

Result: The Program has been successfully executed.

EXPERIMENT 9

Aim:

Module Implementation, Scrum Master to Induce New Requirements in Agile Development

Requirements:

Window 10, Internet, MS Office, vsCODE, Pycharm, git and python modules.

Problem Description:

Scrum Master:

Scrum is a subset of Agile. It is a lightweight process framework for agile development, and the most widely-used one. The scrum master helps to facilitate scrum to the larger team by ensuring the scrum framework is followed. A Scrum Master is a facilitator for an Agile development team. They are responsible for managing the exchange of information between team members. Scrum is a project management framework that enables a team to communicate and self-organize to make changes quickly, in accordance with Agile principles. A Scrum Master is a facilitator for an Agile development team. They are responsible for managing the exchange of information between team members. Scrum is a project management framework that enables a team to communicate and self-organize to make changes quickly, in accordance with Agile principles.

MODULE 1 Implementation:

```
#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()
```

MODULE 2 Implementation:

```
#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())
```

MODULE 3 Implementation:

```
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()
```

MODULE 4 Implementation:

```
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()
```

Result: The Program has been successfully executed.

EXPERIMENT 10

Aim:

Master Test Plan, Test Case Design

Requirements:

Window 10, Internet, MS Office

Problem Description:

Master Test Plan:

- Master Test Plan contains all the test scenarios and risks prone areas of the application.
- It captures each and every test to be run during the overall development of application.
- It includes test scenarios to be executed in all the phases of testing that run during the complete lifecycle of the application development.
- Only for big projects, we need a Master Test Plan which requires execution in all phases of testing.

Test Plan:

- Test Plan document contains test cases corresponding to test scenarios.
- It describes the scope, approach, resources and schedule of performing the test.

- A separate Test Plan exists for each phase of testing like Unit, Functional, and System which contains the test cases related to that type only.
- Preparing a basic Test Plan is enough for small projects.

Sample Test Plan:

Name of the software:

Covid Vaccination Monitoring System (CVMS).

Prepared by:

Ishan Thakur (RA1911003030068)

Akhil Abhilash (RA1911003030086)

Manas Sharma (RA1911003030090)

Harshit Tyagi (RA1911003030093)

Introduction:

Objective of this project is to simplify the COVID - 19 Vaccination process that's running throughout the world, and help people to keep track of their Vaccination Date, Time & Slot, Place and will also remind them to complete the process in time. It will also keep count on the total number and type of vaccines distributed.

Scope:

We will 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 100% efficient software for Covid Vaccine Monitoring to the Government and Public.

Testing strategy:

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

Features not to be tested:

Nil, before releasing the software each and every module and even design and interface will be tested.

Features to be tested:

Testing criteria:

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

Module 2: Real time active Covid cases
Test case 1: Real time active cases window + Check accuracy by taking a data

Module 3: Slot Registration
Test case 1: Slot registration window + Data fill up + confirm slot registration

Module 4: Support
Test case 1: Check the hyperlinks and confirm the working conditions for the same.

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.

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.

Tools:

PyCharm, Visual studio and git

Approvals:

Checked and approved individually by each and every member of the team.

Result: The Program has been successfully executed.

EXPERIMENT 11

Aim:

Manual Testing

Requirements:

Window 10, Internet, MS Office, vsCODE, PyCharm, git and python modules.

What is a Test Case?

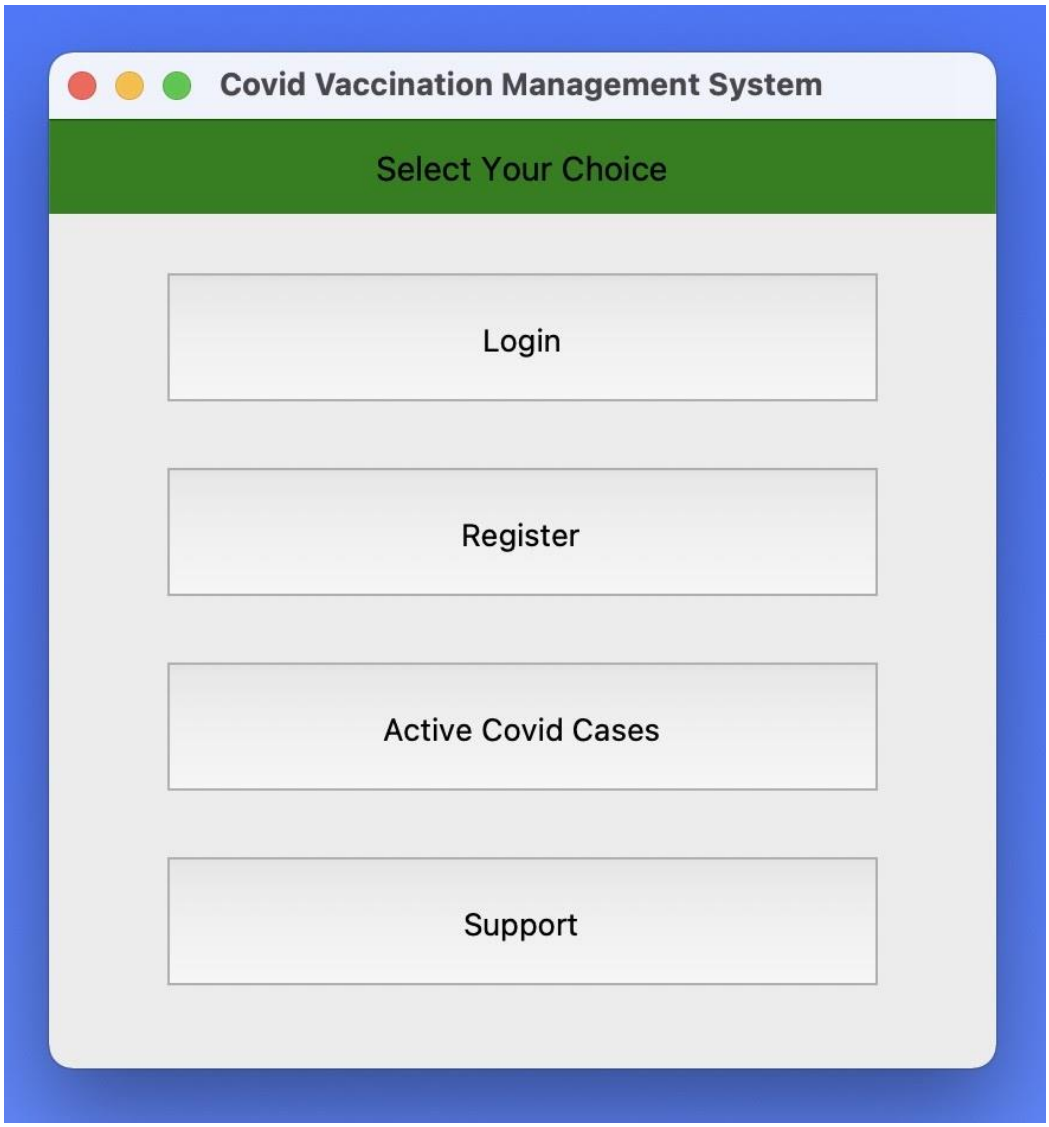
A TEST CASE is a set of actions executed to verify a particular feature or functionality of your software application. A Test Case contains test steps, test data, precondition, postcondition developed for specific test scenarios to verify any requirement. The test case includes specific variables or conditions, using which a testing engineer can compare expected and actual results to determine whether a software product is functioning as per the requirements of the customer.

What is a Test Scenario?

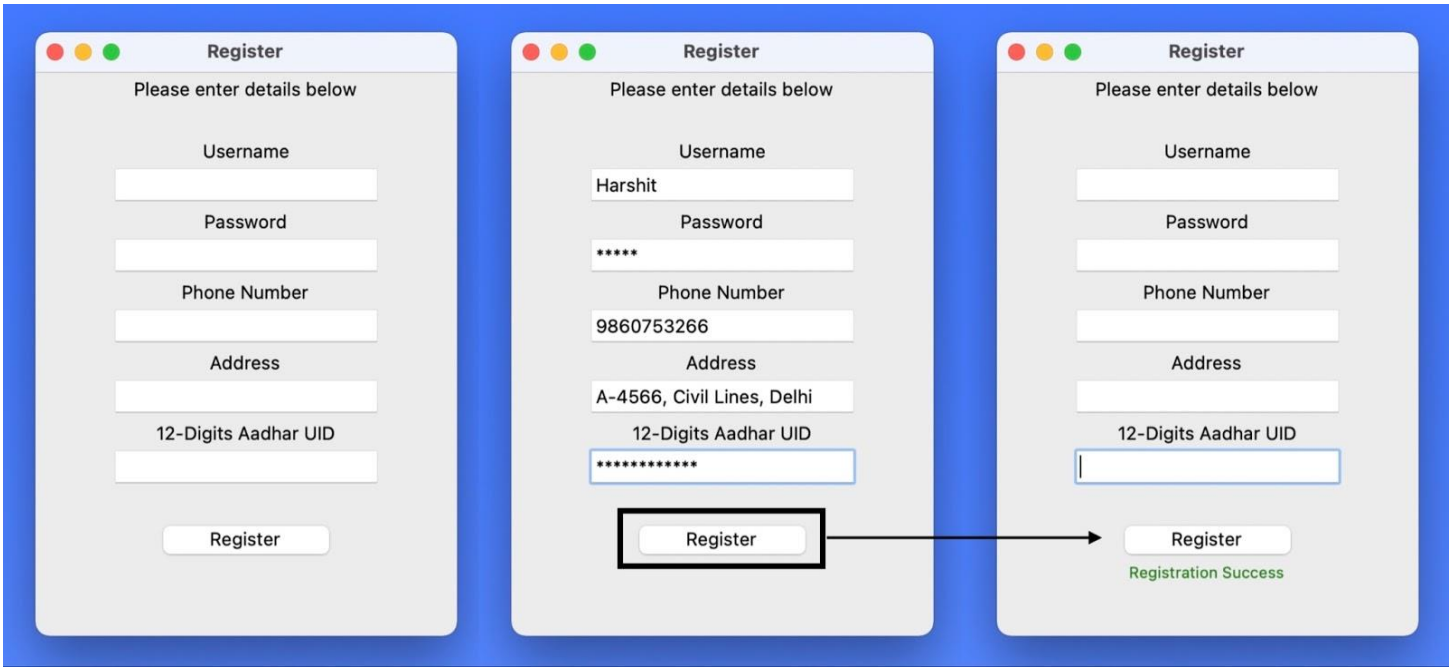
A TEST SCENARIO is defined as any functionality that can be tested. It is also called Test Condition or Test Possibility.

Module 1: Main Screen, New user registration and login window popups

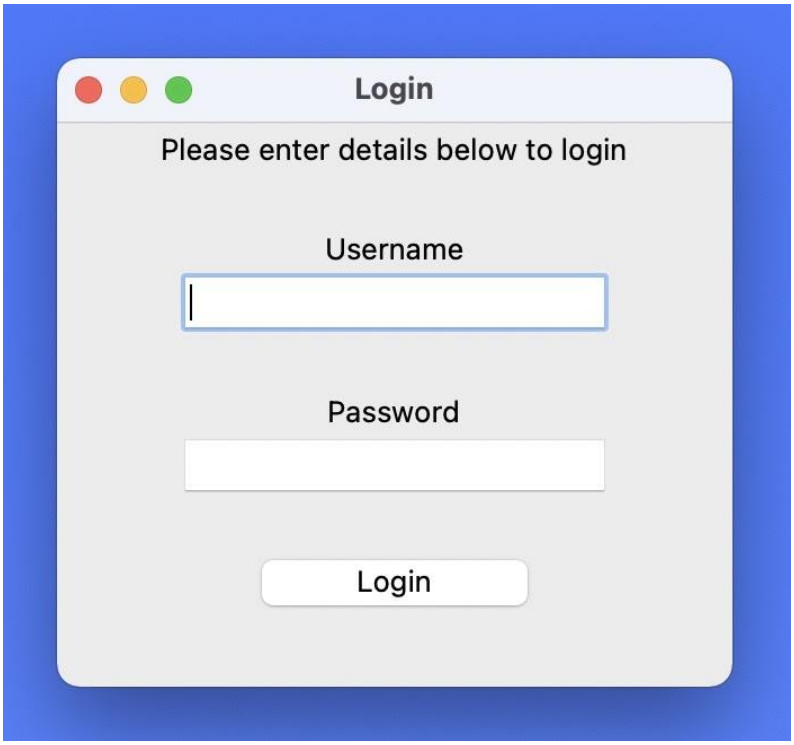
Test Case 1:
Covid Vaccination Monitoring System main screen layout and interface.



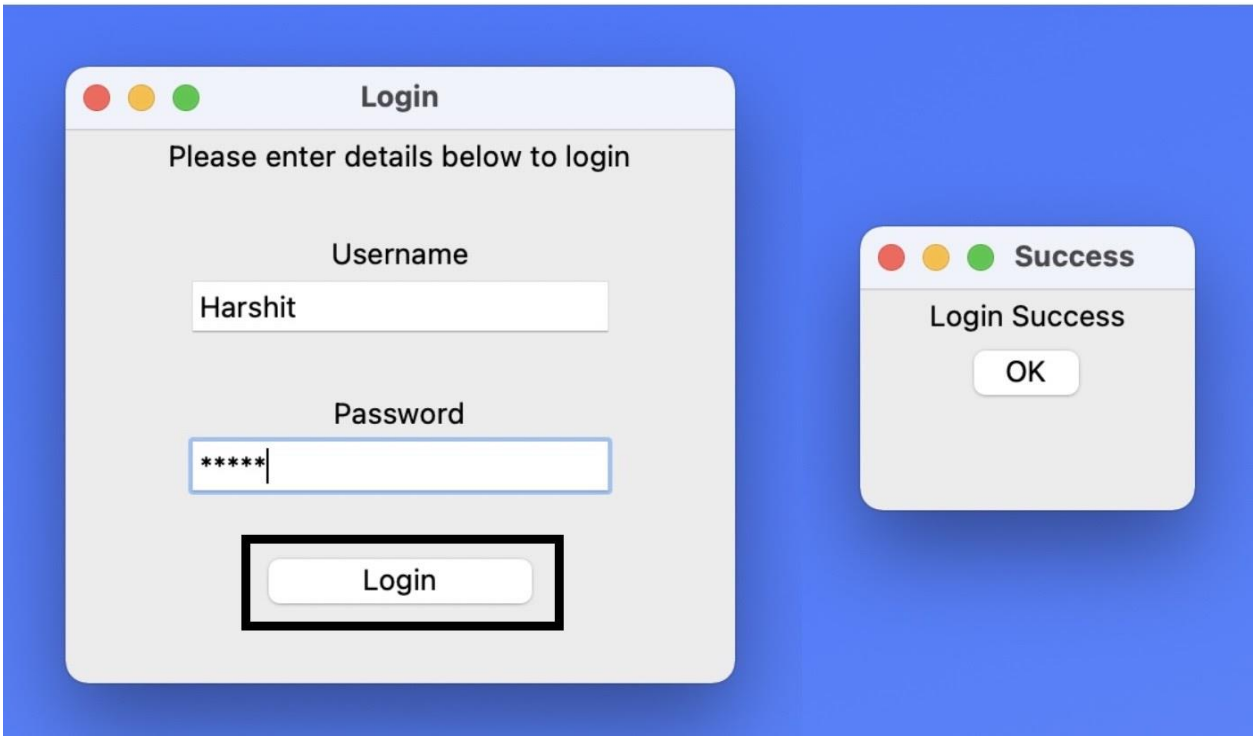
Test Case 2:
User registration window + Detail entry + Registration function check



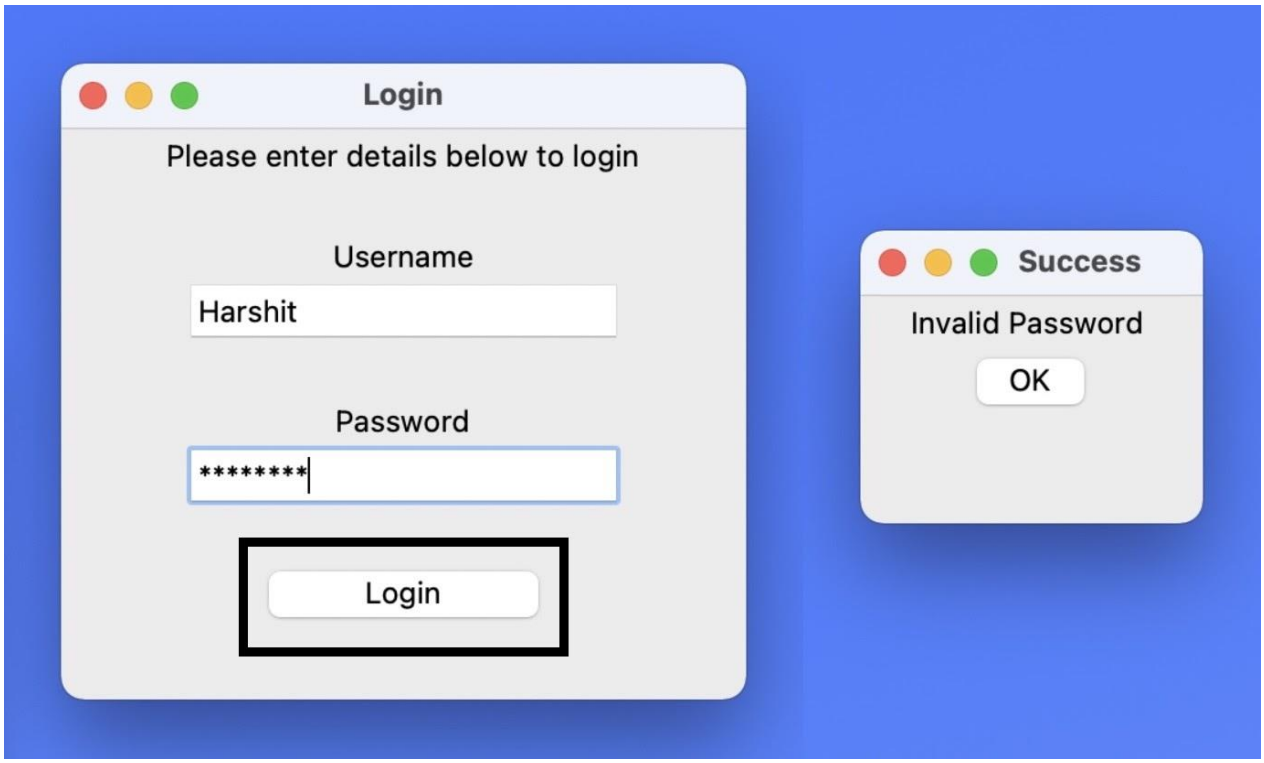
Test Case 3:
User Login window



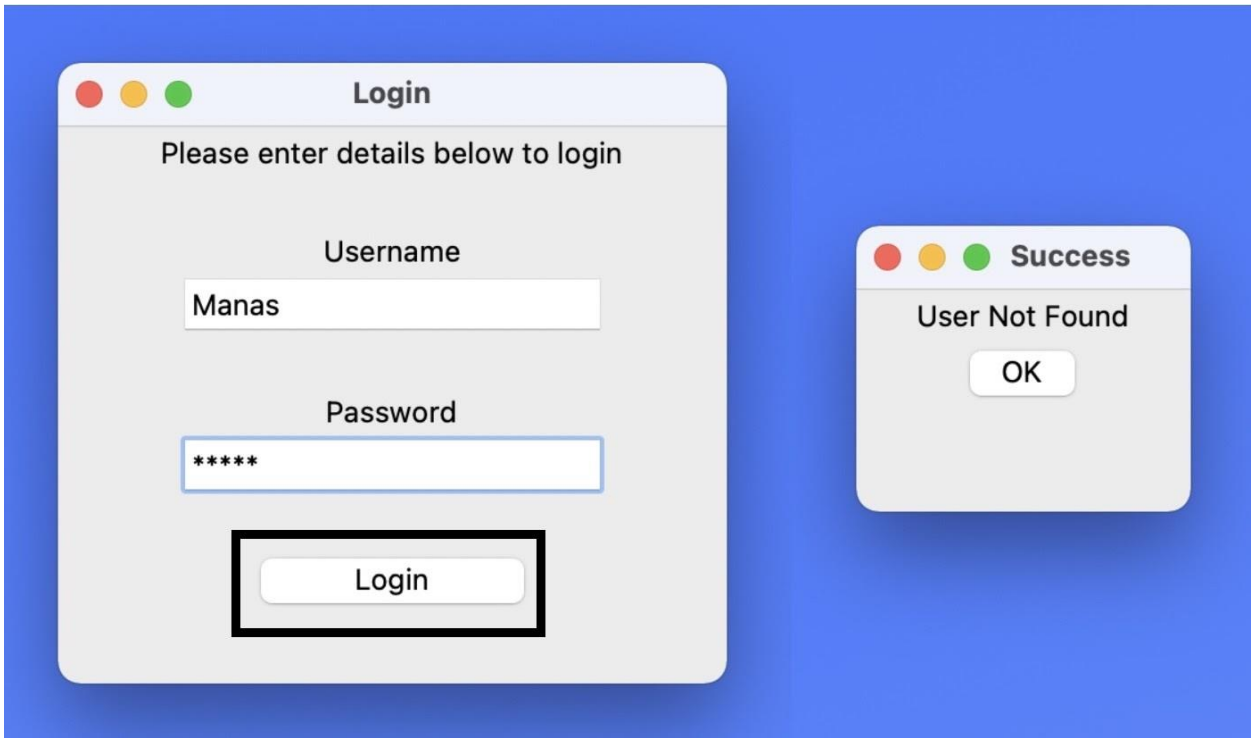
Case 3.1: Successful login



Case 3.2: Invalid Password



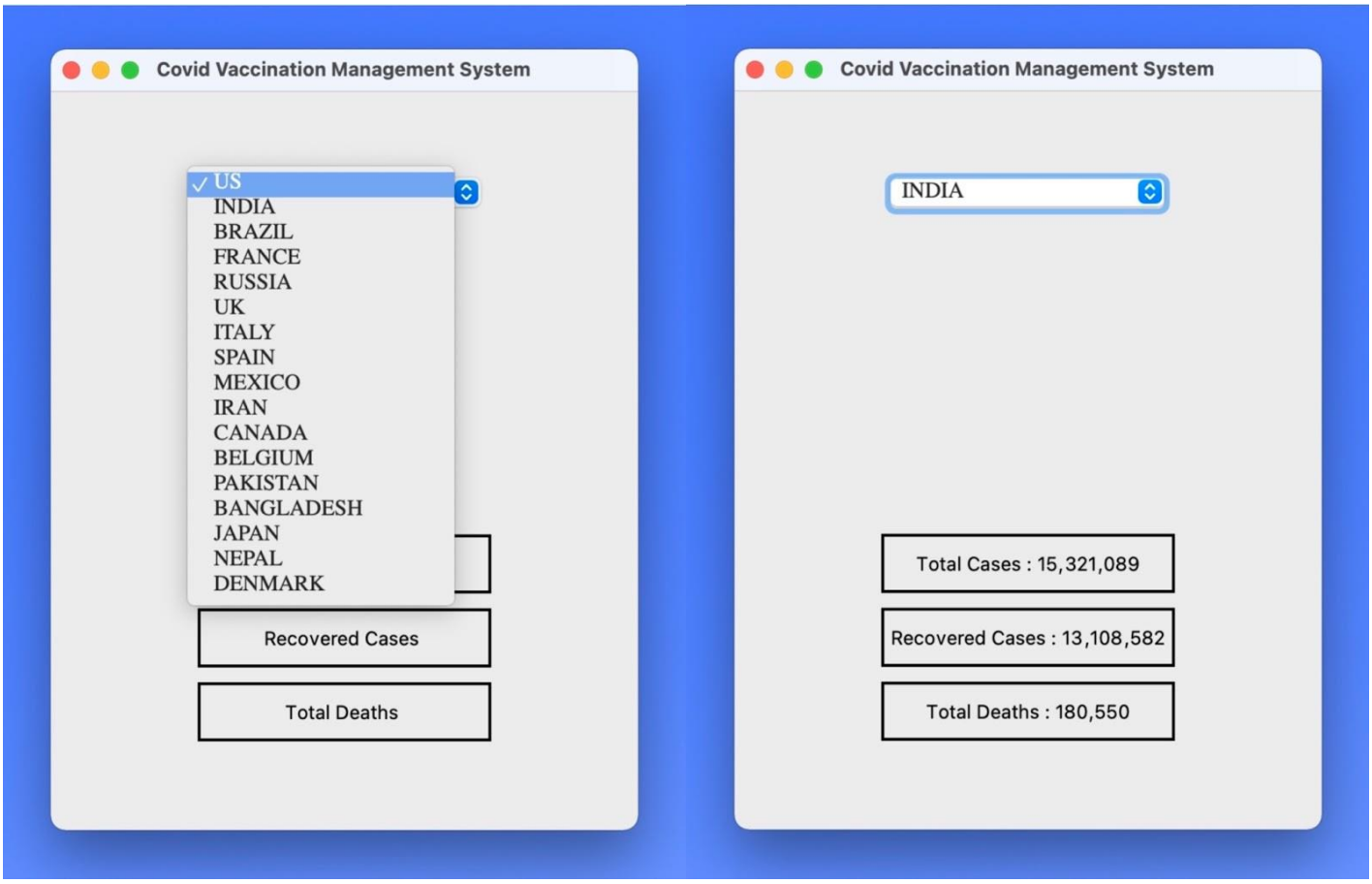
Case 3.3: User not found



Module 2 – Real time active Covid cases

Test case 1:

Real time active SARS COVID cases window



Module 3 – Slot Registration

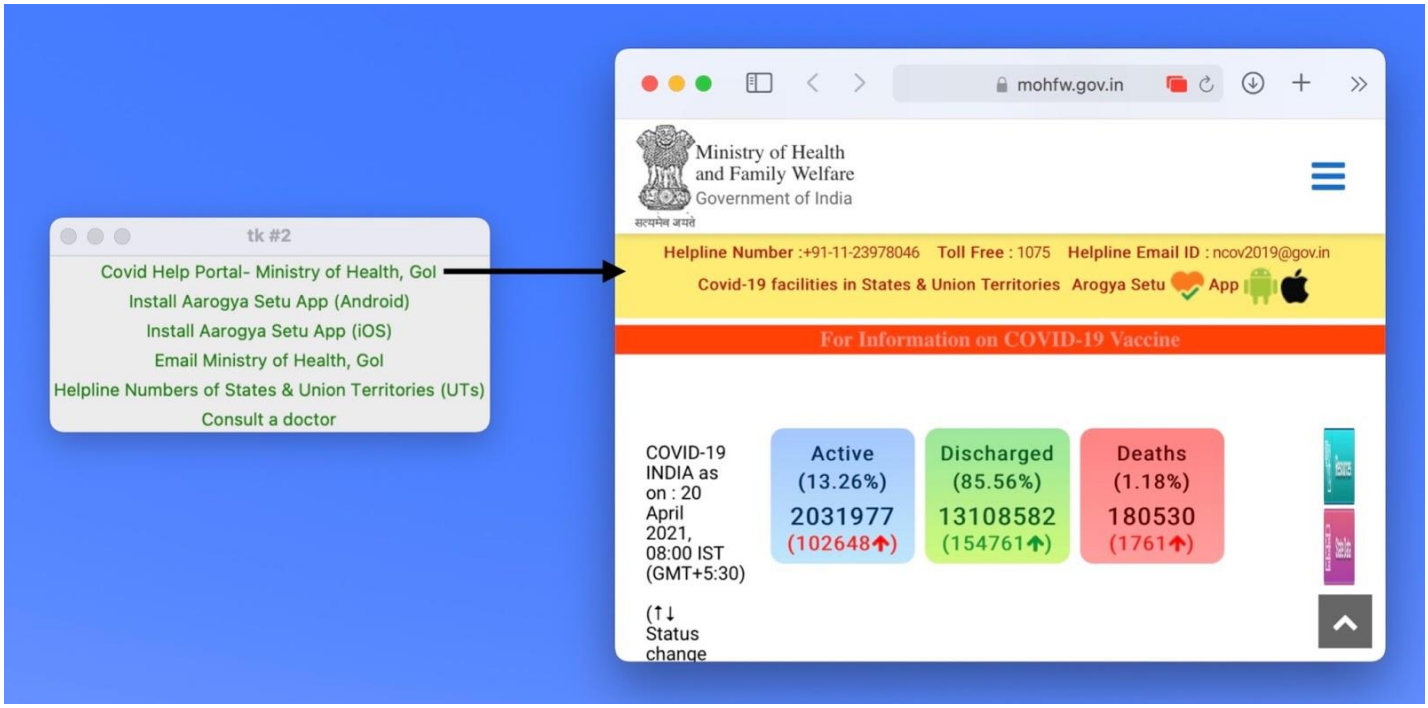
Test case 1:

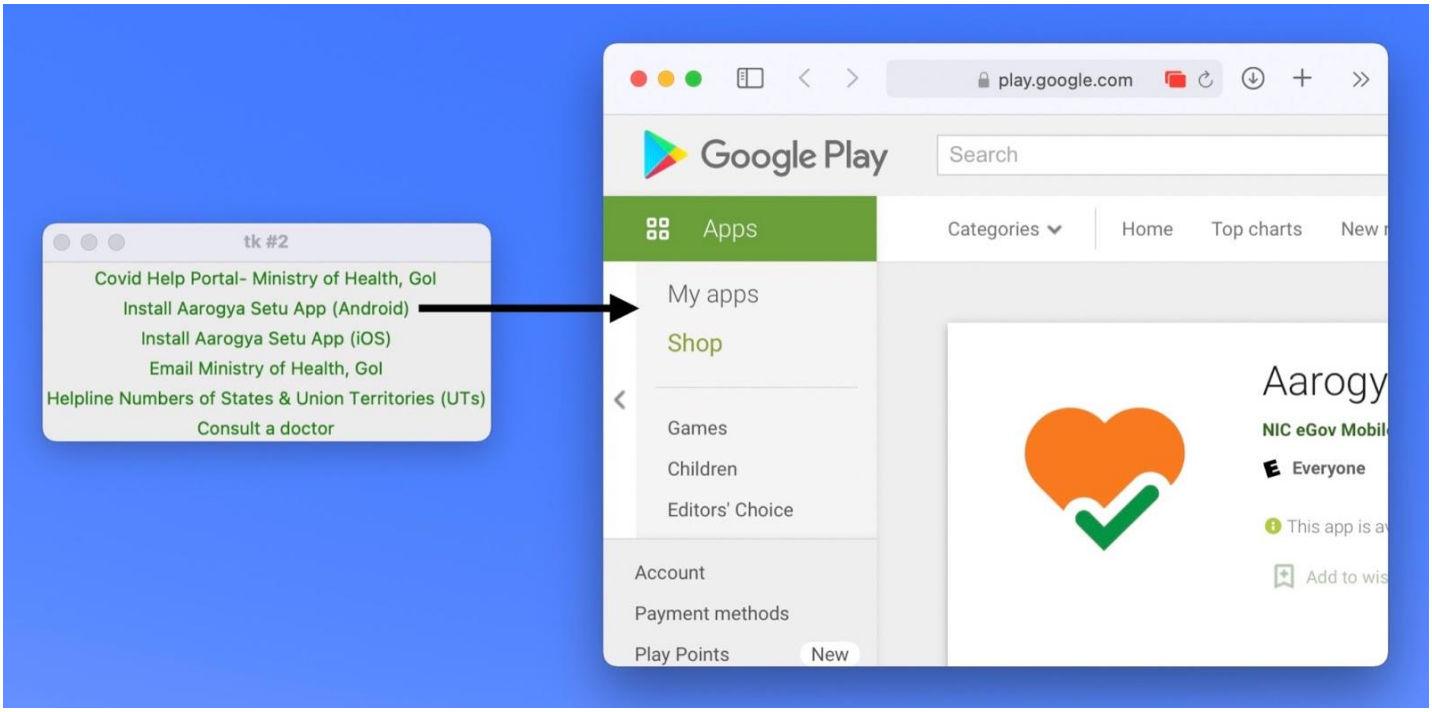
Slot registration window + Data fill up + confirm slot registration

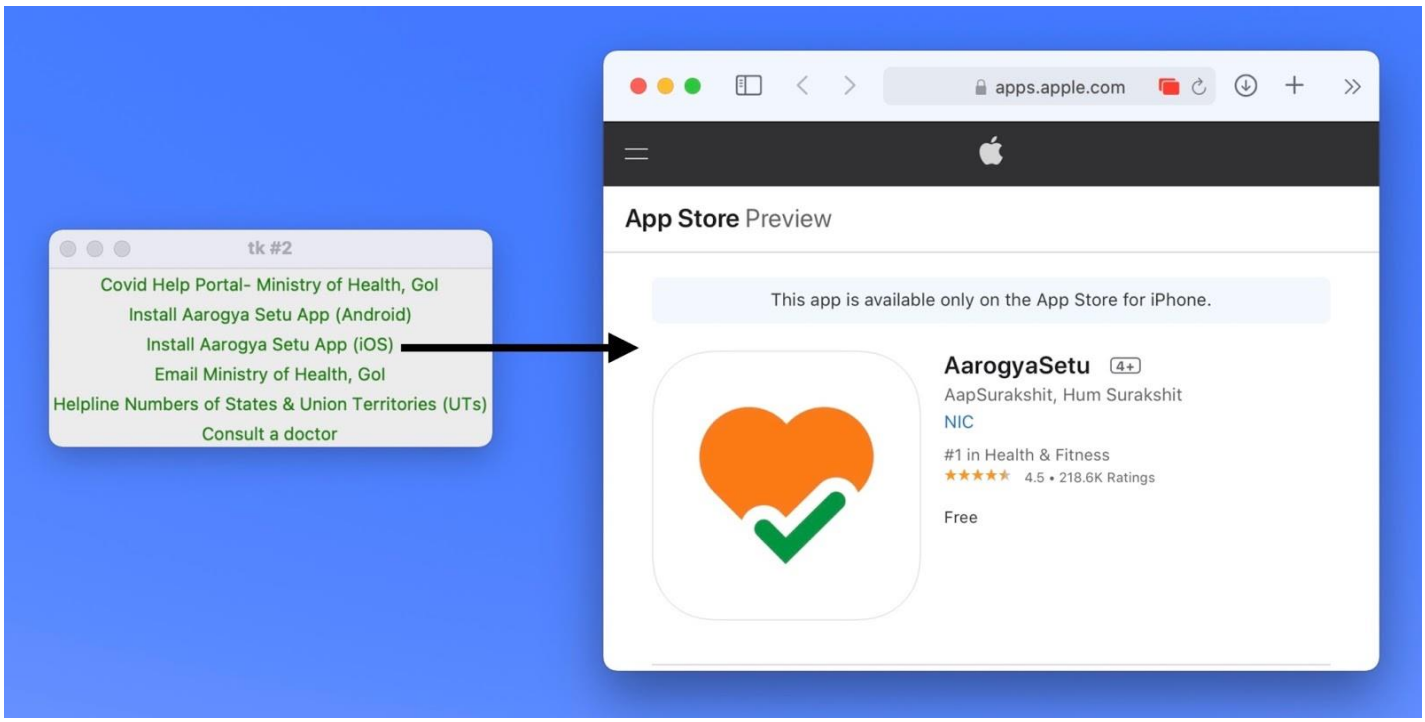
Module 4 – Support

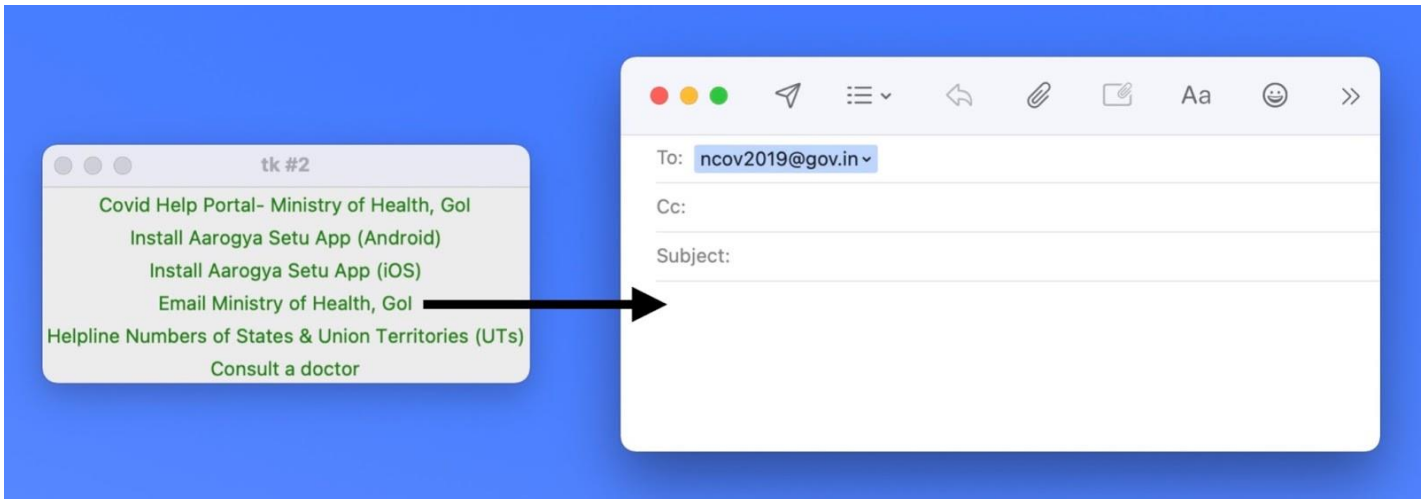
Test case 1:

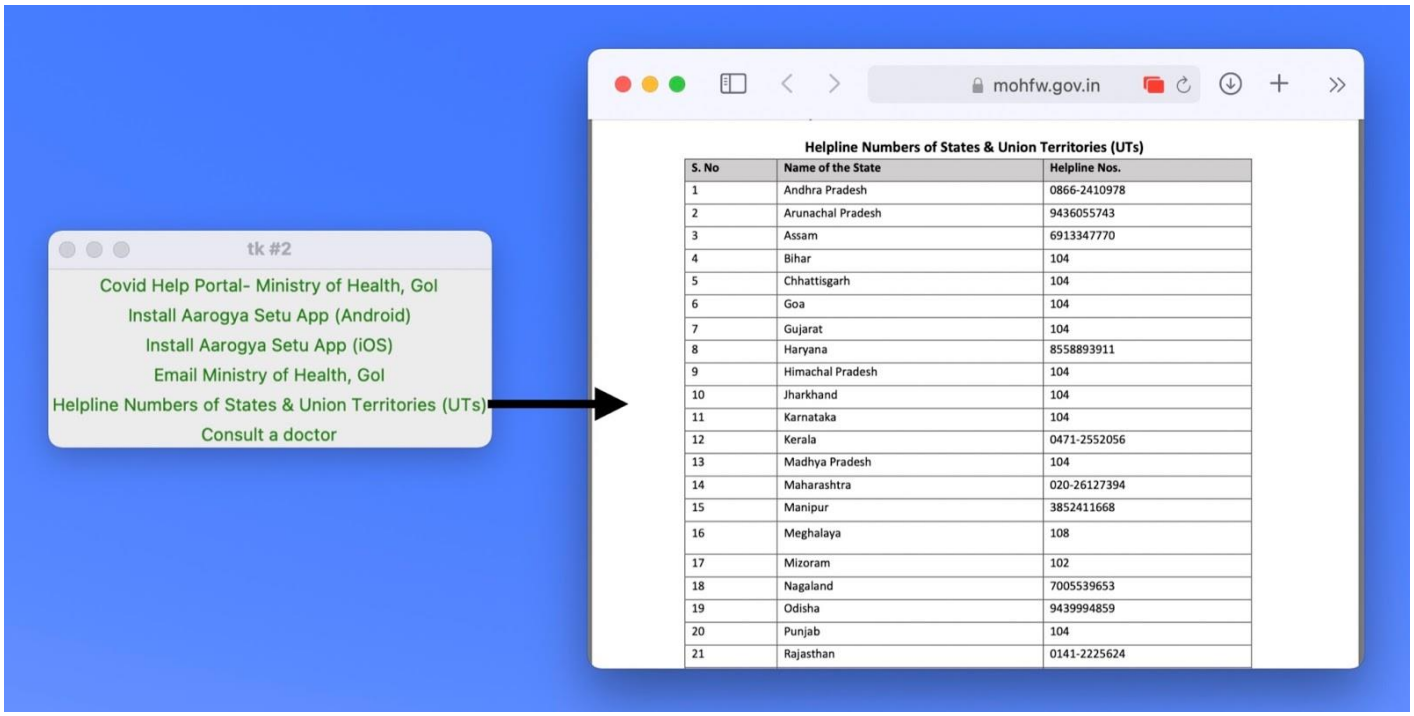
Check the hyperlinks and confirm the working conditions for the same.




tk #2
Covid Help Portal- Ministry of Health, GoI
Install Aarogya Setu App (Android)
Install Aarogya Setu App (iOS)
Email Ministry of Health, GoI
Helpline Numbers of States & Union Territories (UTs)
Consult a doctor
play.google.com
Google Play
Search
Apps
Categories
Home
Top charts
My apps
Shop
Games
Children
Editors' Choice
Account
Payment methods
Play Points
New
Everyone

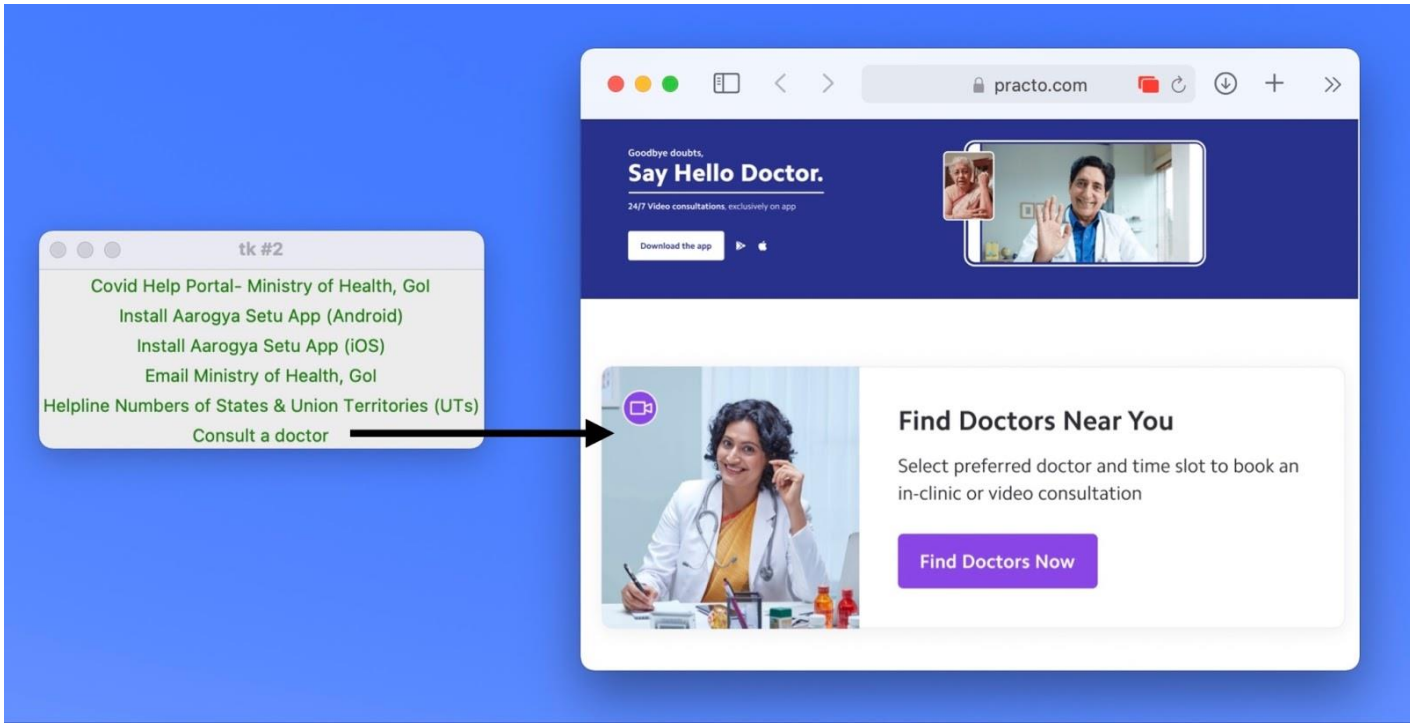

tk #2
Covid Help Portal- Ministry of Health, GoI
Install Aarogya Setu App (Android)
Install Aarogya Setu App (iOS)
Email Ministry of Health, GoI
Helpline Numbers of States & Union Territories (UTs)
Consult a doctor
apps.apple.com
App Store Preview
This app is available only on the App Store for iPhone.
AarogyaSetu
4+
AapSurakshit, Hum Surakshit
NIC
#1 in Health & Fitness
4.5 • 218.6K Ratings
Free





Helpline Numbers of States & Union Territories (UTs)

S. No	Name of the State	Helpline Nos.
1	Andhra Pradesh	0866-2410978
2	Arunachal Pradesh	9436055743
3	Assam	6913347770
4	Bihar	104
5	Chhattisgarh	104
6	Goa	104
7	Gujarat	104
8	Haryana	8558893911
9	Himachal Pradesh	104
10	Jharkhand	104
11	Karnataka	104
12	Kerala	0471-2552056
13	Madhya Pradesh	104
14	Maharashtra	020-26127394
15	Manipur	3852411668
16	Meghalaya	108
17	Mizoram	102
18	Nagaland	7005539653
19	Odisha	9439994859
20	Punjab	104
21	Rajasthan	0141-2225624



Result: The Program has been successfully executed.

EXPERIMENT 12

Aim:

User Manual, Analysis of Costing and Resource

Resources:

Each project needs resources to contribute to its implementation and ultimately to its success. The three types of resources that enabled us to complete the CVMS in a timely and budgeted manner.

- Human Resources:
  Great Team work helped us in carrying out the tasks necessary for the smooth running and completion of the project. In addition, technical skills, knowledge and time management helped us a lot.

- Material resources:
  High speed internet and External softwares and applications like PyCharm, Git, Python Modules, Photoshop, MS Office Suite and Creately helped us a lot to complete the project without any much difficulty.

- Time resources:
  A good time managing skills, a good project plan and timely availability of human and material resources helped us to complete our software within the set time period.

Costing:

CVMS development team is committed to privacy and being transparent about government requests for customer data globally. We've created this project by matching all the project plans and covering all the necessary functions focusing on the current situation, and we are always on the edge to bring new updates. The price for the CVMS License starts from 5K INR.

User Manual:

A **user guide**, also commonly called a technical communication document or **manual**, is intended to give assistance to people using a particular system. It is usually written by a technical writer, although user guides are written by programmers, product or project managers, or other technical staff, particularly in smaller companies. Most user guides contain both a written guide and associated images. In the case of computer applications, it is usual to include screenshots of the human-machine interface(s), and hardware manuals often include clear, simplified diagrams. The language used is matched to the intended audience, with jargon kept to a minimum or explained thoroughly.

CVMS

(COVID-19 Vaccination Monitoring System)

User Manual

Table of Contents

1. GETTING STARTED

1.1 How to download:

Click on the link provided below to download the CVMS software on your device OR Scan the QR code.

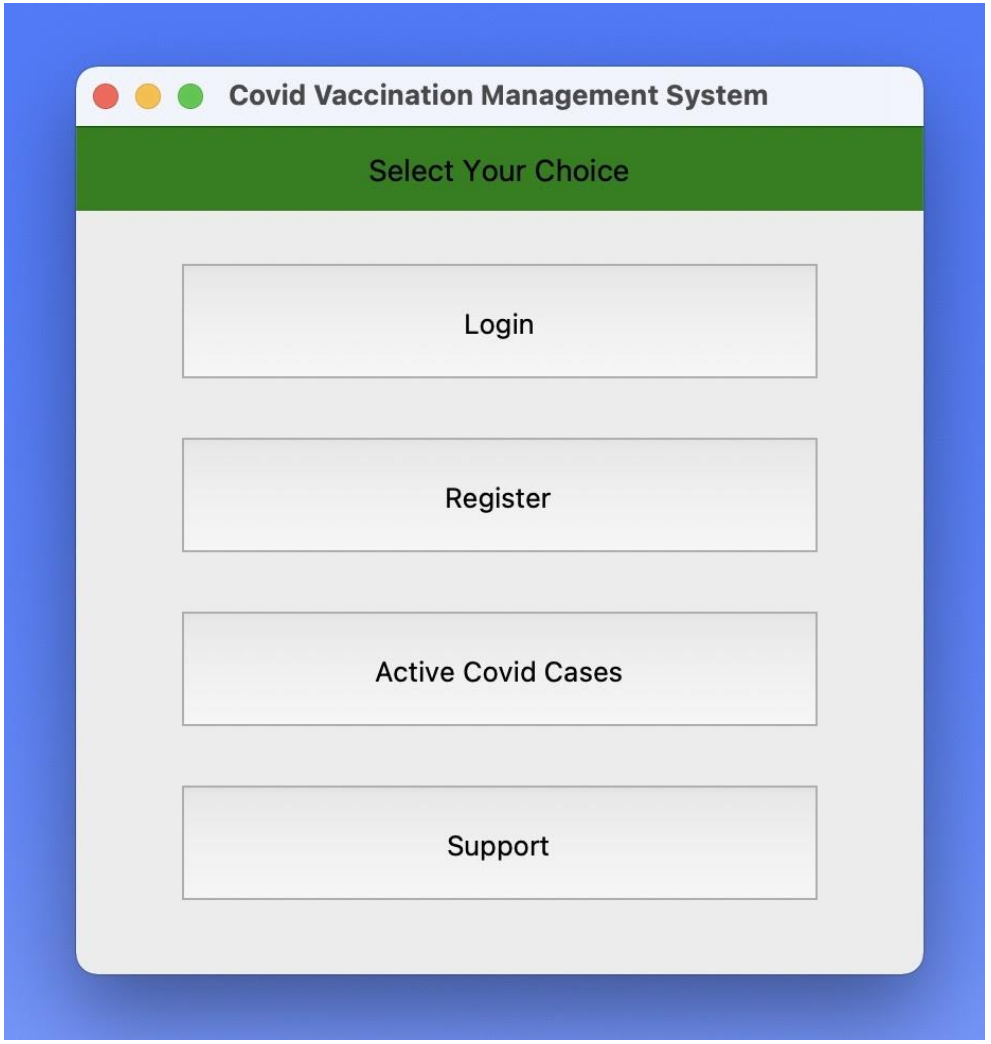
https://github.com/HtGeeky/CVMS-Covid19-Vaccination-Monitoring-System-.git

2. SETTING UP

2.1 How to register

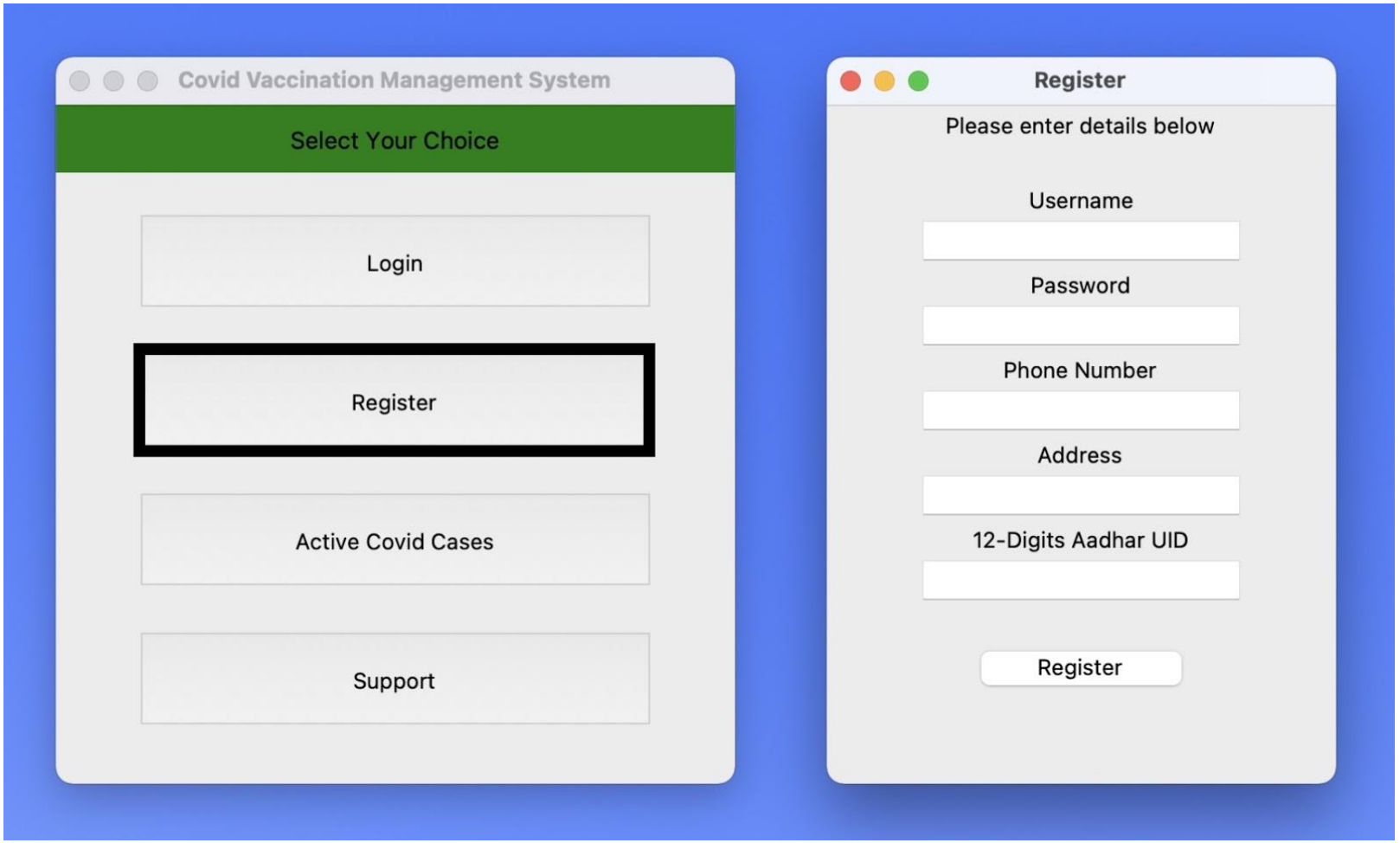
After installation, open the software.

A Dialogue box will appear.

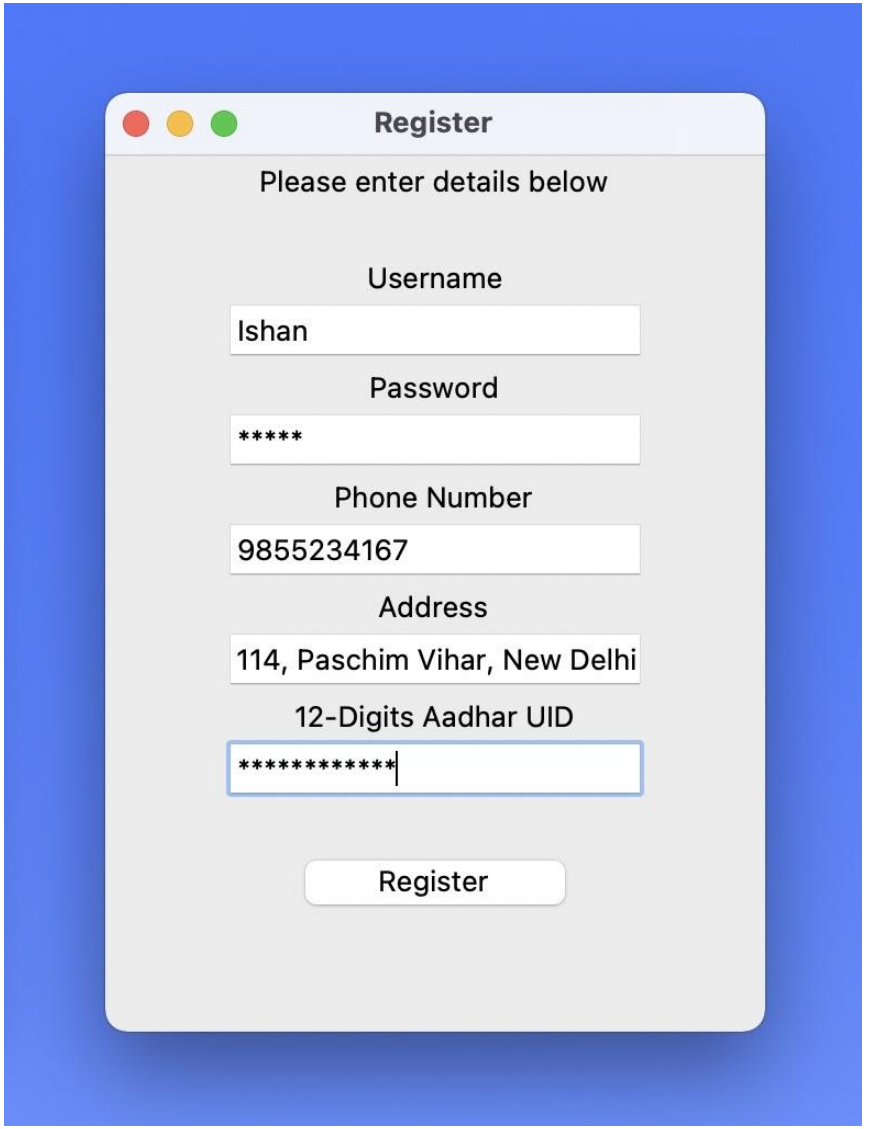


Click on the register button.

A register window will appear.

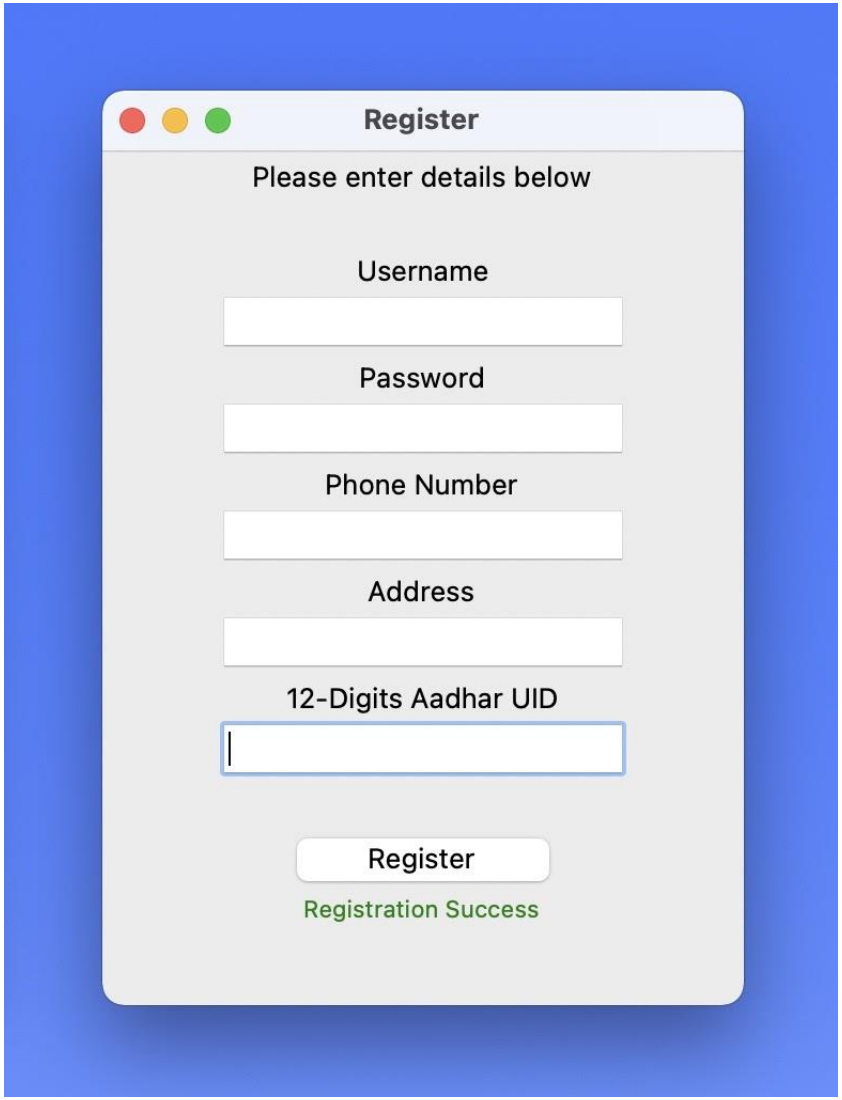


Fill in the details carefully in the respective area.



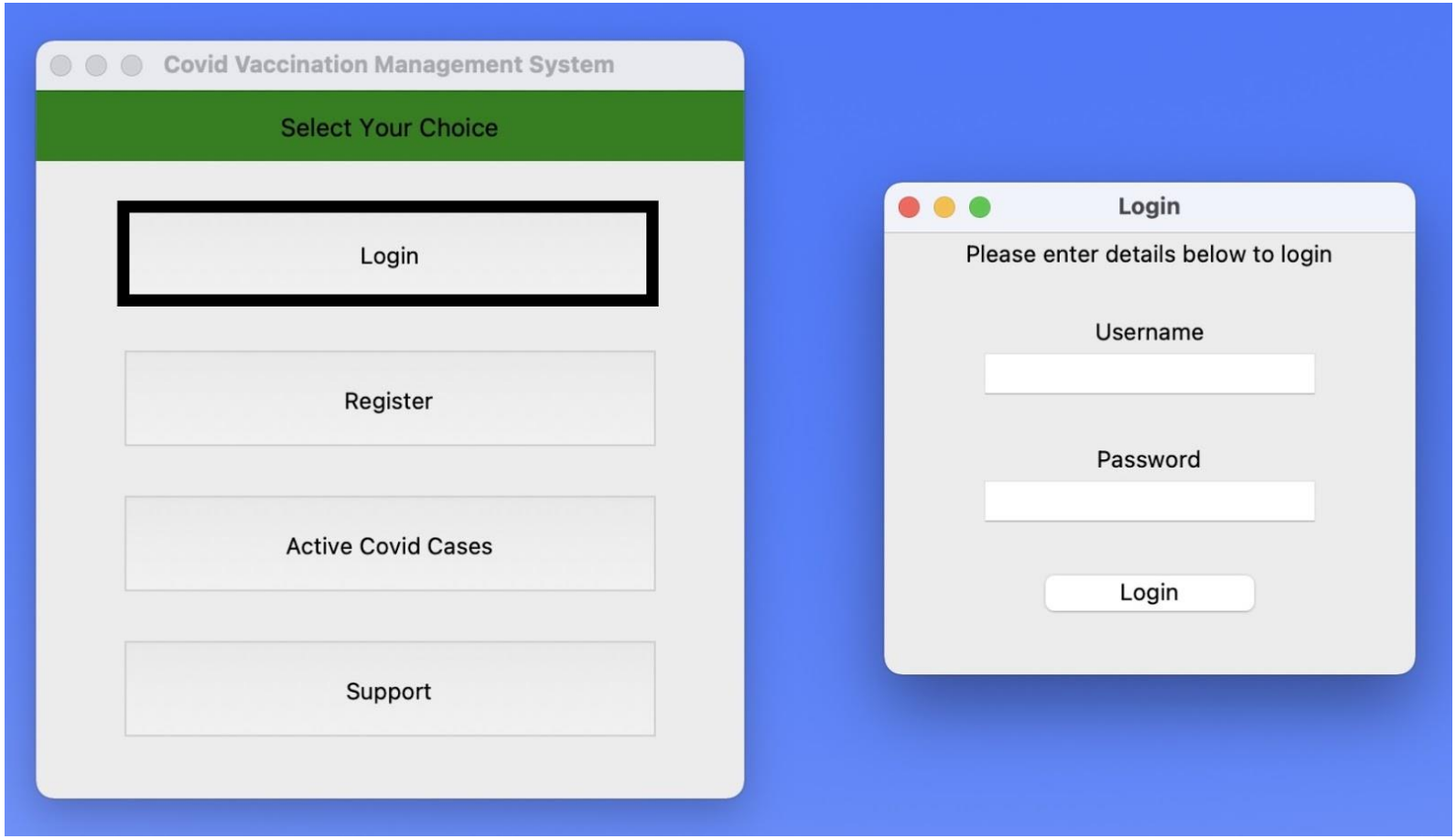
After filling the details, click on the Register button.

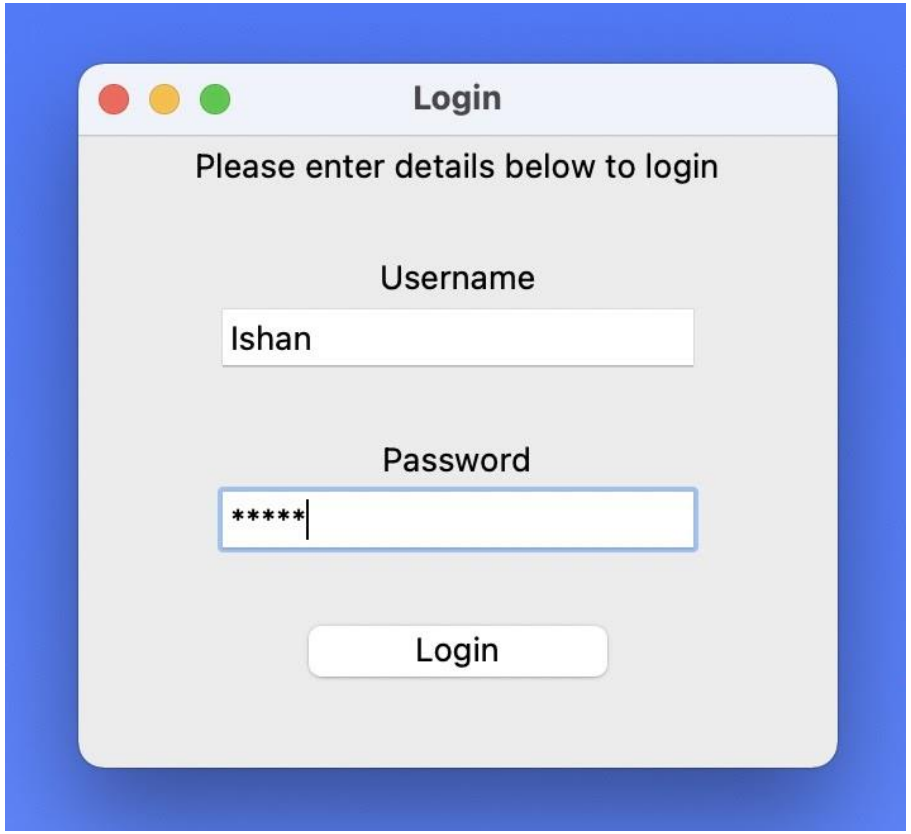
You have been successfully registered.



3. How to Login

In the home screen click on the Login button. A window will pop up.





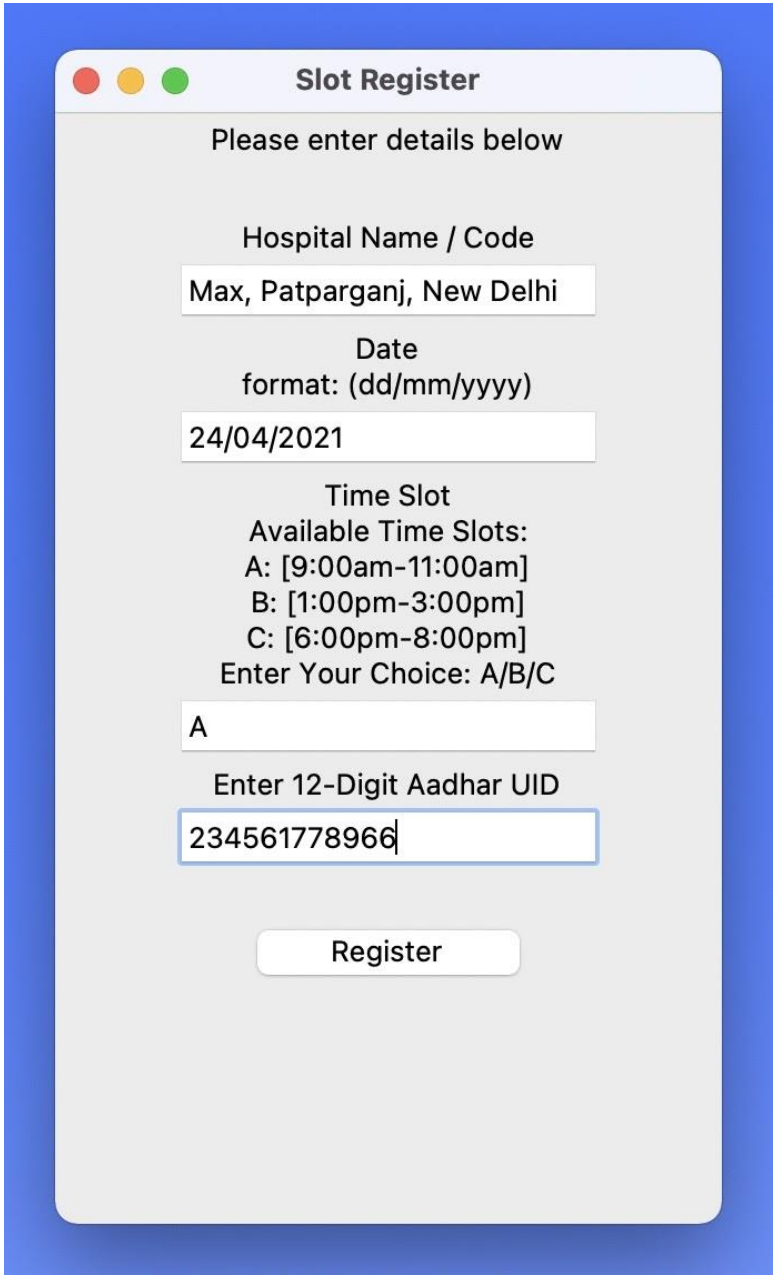
Fill in your correct details and click Login.

4. How to register for vaccination

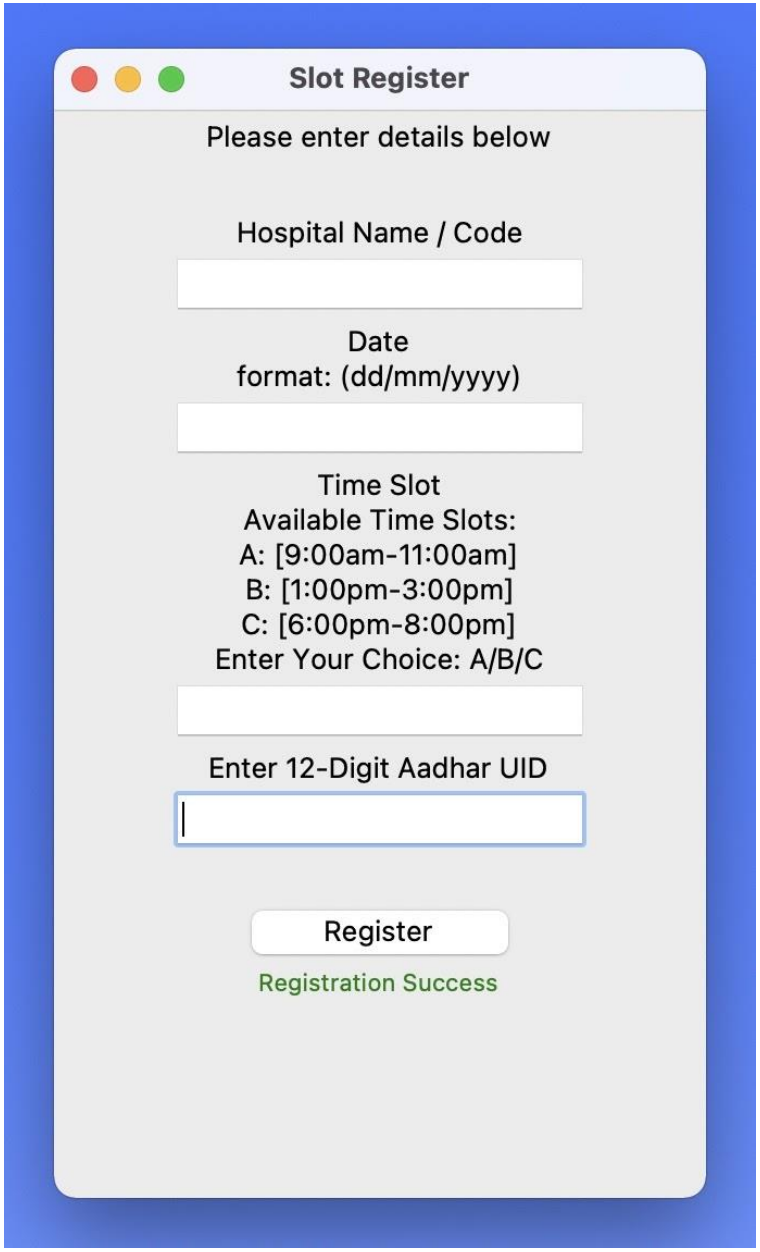
After successful login you will be redirected to the Slot Registration window.

Choose the Hospital name, date, time as per your convenience.

Lastly fill in your 12 Digit Aadhar UID.



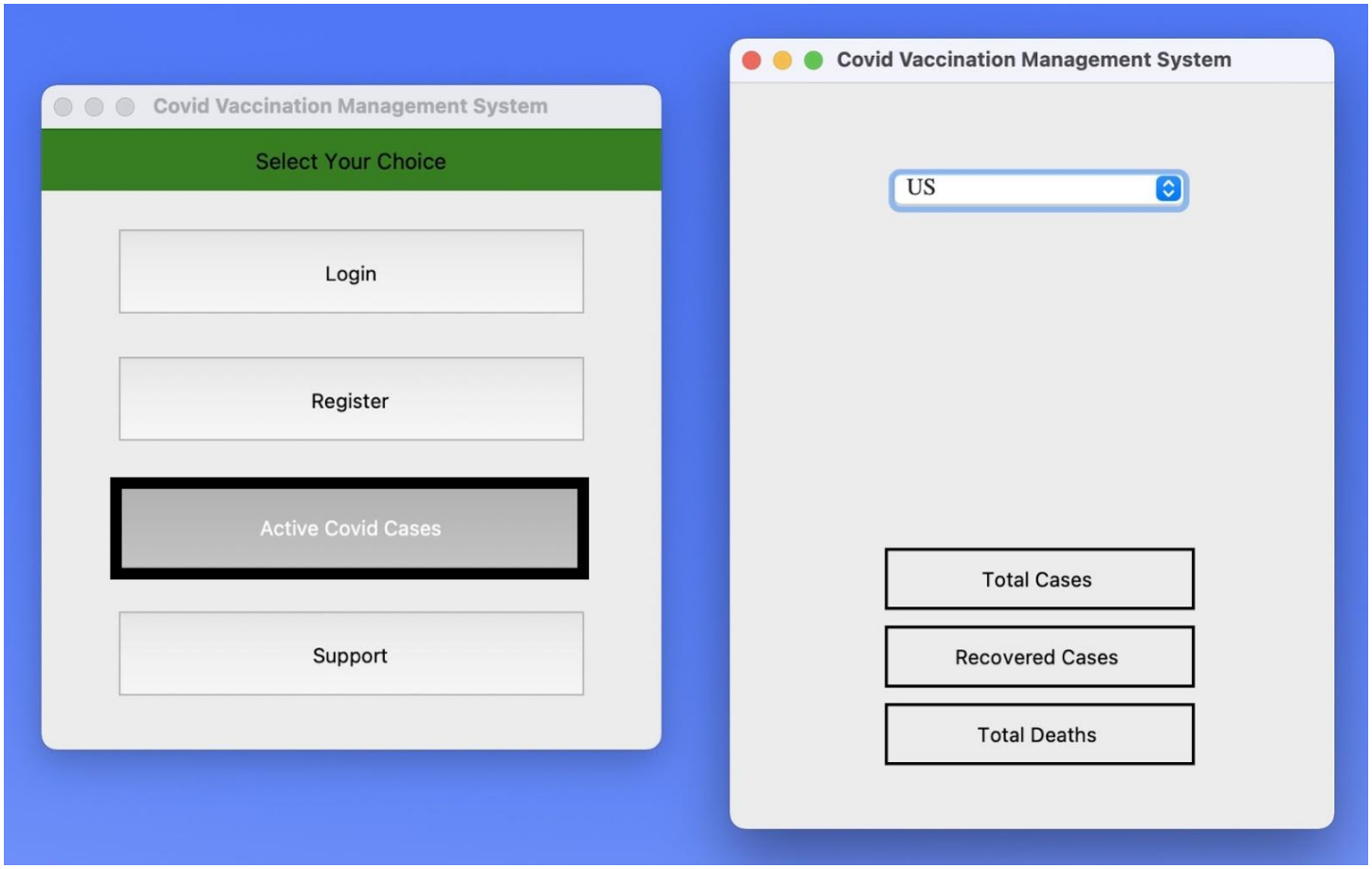
Click on Register to book your appointment.



5. How to check real time COVID cases.

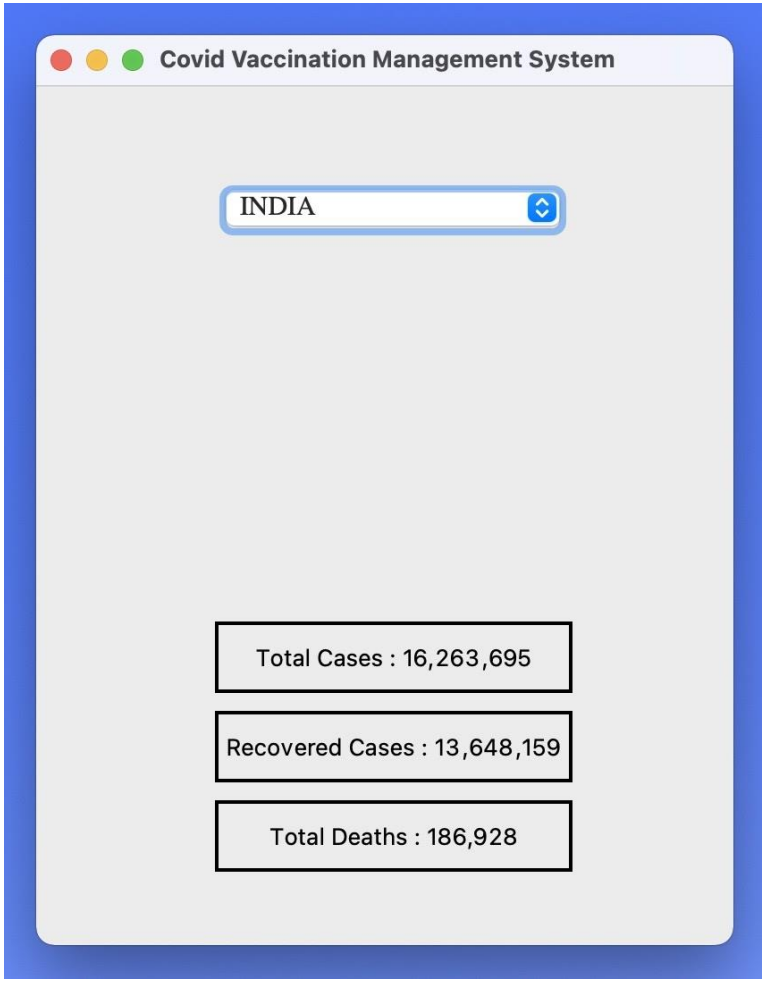
In the home screen click on the Active Covid Cases button.

A window will appear.



Choose the country you want to see the total number of COVID cases from the drop down menu.

The details will be displayed as follows:



6. User Support

In the home screen, click on the Support button.

A new window will appear.

Check the hyperlinks for user support.

7. System Updates

Due to software updates, your experience of the software interface (including but not limited to software features, user interfaces, and interaction experiences) may differ from the interface presented in this manual. The software interface is subject to change.

8. Safety & Privacy

As per the regulations directed by the Indian government, all the personal data provided by the user will be encrypted and won't be shared to any third parties.
Selected details will be shared and used by the respective hospitals/ covid vaccination centers.

9. Contact Us

For troubleshooting section and instructions on how to solve problems or any other queries, email us at the following: getsupport@cvms.com