# CHAPTER 1

# INTRODUCTION

## 1.1 Introduction to the title of project

Covid and Vaccination System is capable of managing each and every data regarding Covid and vaccination. This system helps the Government and other concerned authorities in making effective policies because of proper flow of information from ground reality to higher levels.

Patient records and vaccination slot maintenance modules are also included in this system which would keep track of the people using the index files and also a detailed description about the personal information. With this computerised system there will be no loss of positive patients records or vaccination records which generally happens when a Non computerised system is used.

Initially the authentic databases/ledgers were present which was mostly managed by humans manually which were more prone to mistakes and had many disadvantages like: Vulnerability to human errors, Chances of missing the entry important information in the record, Search for records takes longer, cost to maintain records, Less efficient. But by using this system these barriers can be overcome. And our project aims to create easy to use, error free and efficient Covid and vaccination system.

## 1.2 Architecture

The File System Architecture Specifies that how the Files will be stored into the Computer system means how the Files will be Stored into the System. Means how the data of the user will be Stored into the Files and how we will Access the data from the File. There are many types of Storage or Space allocation techniques which specify the Criteria by using; the files will store the data into them.

Allocation has three types i.e. Continuous space allocation, Linked allocation and Indexed allocation.

## 1.3 Organisation of the report

The next chapter 2 contains the Problem statement of the project. Chapter 3 deals with the database design model. Chapter 4 defines the database implementation. Chapter 5 explains the software requirement analysis. Chapter 6 describes the User interface and functionality design. Chapter 7 interprets with Implementation. Chapter 8 illustrates testing. Chapter 9 contains Results and Snapshots. References are added at the end.

# CHAPTER 2

# PROBLEM STATEMENT

## 2.1 Business domain

A correlation study to assess the knowledge and self-expressed stigma regarding COVID-19 Outbreak and its vaccination among adults at selected society of Bangalore city.

The project can be easily implemented under various situations. We can add patient records and vaccination slots when required. Reusability is possible as and when we require in this application. There is flexibility in modules like searching the records of the patients and count of the people vaccinated and so on.

## 2.2 Business domain diagram and analysis

The below flowchart explains how the Covid and vaccination entry process in the real world. After entering into the home page there are two choices, **Covid menu, and Vaccination menu.** Admin can choose either of them where they can perform C.R.U.D operations on the file.

Thousands of patients and hundreds of appointments to book slots for vaccination are difficult to maintain and they are exposed to error since human intervention is involved.

The purpose of the Covid and Vaccination system is to allow for storing details of a large number of patients and vaccination appointments which allow for add, delete records and other facilities separately to administrators.

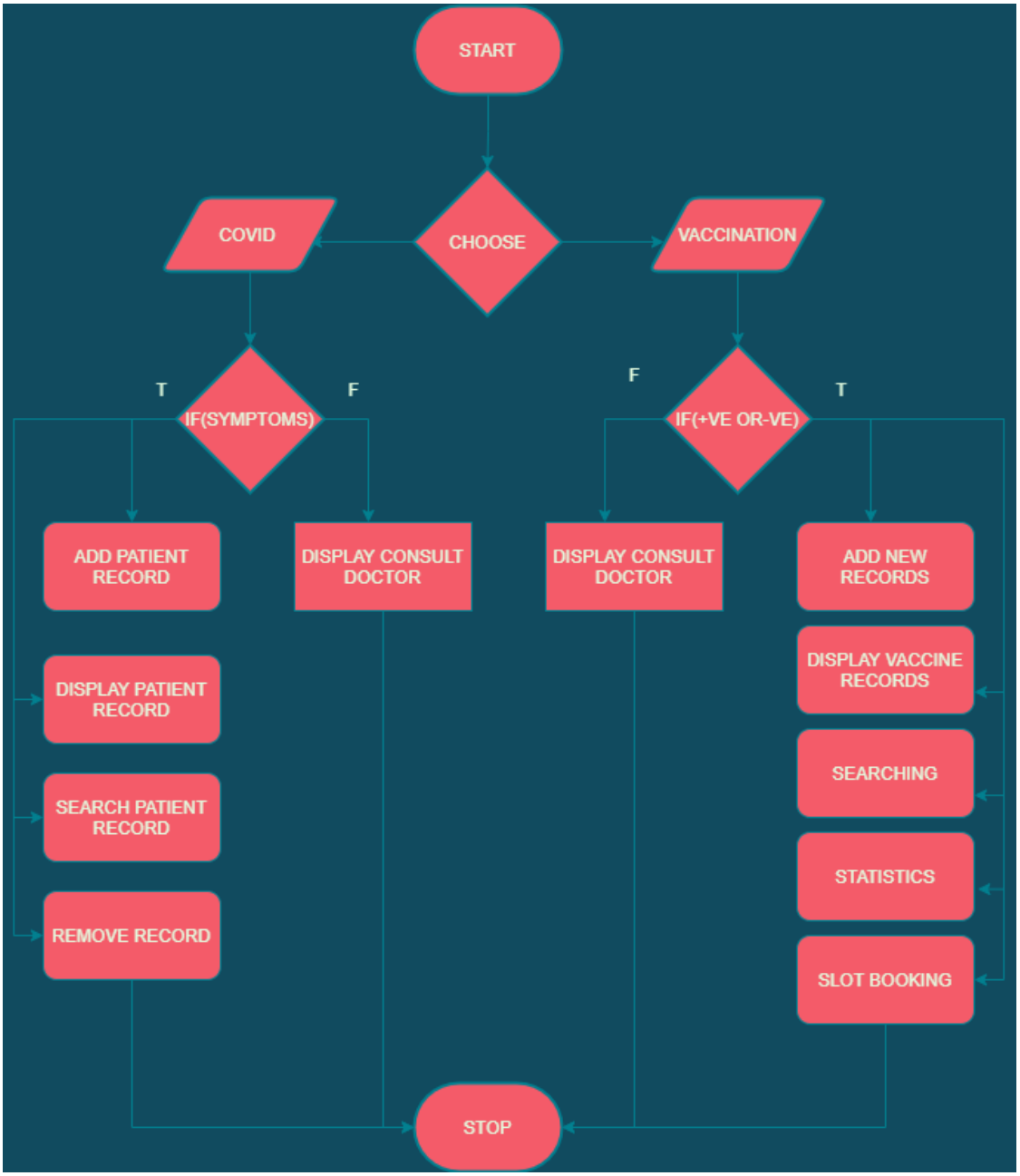


Fig 2.1: Flowchart of covid and vaccination system

# CHAPTER 3

# LITERATURE SURVEY

## 3.1 HISTORY OF FILE STRUCTURES

Early work with files presumed that files were on tape, since most files were. Access was sequential and the cost of access grew in direct proportion to the size of the file. The indexes made it possible to keep a list of keys and pointers in a smaller file that could be searched more quickly. With the key and pointer, the user had direct access to the large, primary file.

Unfortunately, simple indexes had some of the same sequential flavor as the data files, and as the indexes grew, they too became difficult to manage, especially for dynamic files in which the set of keys changes. Then in the early 1960's, the idea of applying tree structures emerged. Unfortunately, trees can grow very unevenly as records are added and deleted.

In 1963 researchers developed the tree, an elegant, self-adjusting binary tree structure, called AVL tree, for data in memory. The problem was that even with a balanced binary tree, dozens of accesses were required to find a record in even moderate sized files.

It took nearly ten more years of design work before a solution emerged in form of the B-tree. AVL trees grows from top down as records are added, B-trees grow from the bottom up.

B-trees provided excellent access performance, but there was a cost: no longer could a file be accessed sequentially with efficiency. Fortunately, this problem was solved almost immediately by adding a linked list structure at the bottom level of the B-tree. The combination of a B-tree and a sequential linked list is called a B+ tree.

Being able to retrieve information with just three or four accesses is pretty good. But the goal of being able to get what we want with a single request was satisfied by hashing. From early on, hashed indexes were used to provide fast access to file. After the developed of B-trees, researchers turned to work on systems for

extendible, dynamic hashing that could retrieve information with one or, at most, two disk accesses no matter how big the file became.

## 3.2 TYPES OF FILES

As we know that computers are used for storing the information for a permanent time or the files are used for storing the Data of the users for a long time period. And the files can contain any type of information which means that they can store text, any images or pictures or data in any format. So that there must be some mechanism which are used for storing the information, Accessing the information and also performing some operations on the files.

When we store a file in the system, then we have to specify the name and the type of file. The name of file will be any valid name and type means the application with which the file is linked.

When we say that every file has some type, it means that every file belongs to a special type of application software. When we provide a name to a file, we also specify the extension of the file because the system will retrieve the contents of that file into application software. For example, if there is a file which contains some paintings then this will open in software.

**1. Ordinary file or Simple file:** Ordinary file may belong to any type of application. For example, notepad, paint, C program, music[mpeg] etc. All the files that are created by a user are ordinary files. Ordinary files are used for storing information about the user programs. With the help of ordinary files, we can store the information which contain text, database, images or any other type of information.

**2. Directory File:** The files those are stored into a particular directory or folder. These are known as Directory files because they belong to a particular directory.

**3. Special Files:** The special files are those which are not created by the user or the files which are necessary to run a system. These are the files that are created by the system. All the files of an operating system are referred to as special files. There are many types of special files and they are, system files or windows files, input/output files. System files are stored using .sys extension.

**4. FIFO Files:** The first in first Out Files are used by the system for executing the processes in a particular order. Which means that the files which comes first, will be executed first and the system maintains a order known as Sequence order. When a user requests for a service from the system, then the requests of the users are arranged into some files and all the requests will be performed by the system, using some sequential order known as FIFO order

## 3.3 TYPES OF OPERATION

Files are not created only for reading, we can also perform some operations on files, they are

1. Read Operation: This is meant to read the content which is stored in the files.

2. Write Operation: This is meant to read the content which is stored in the files.

3. Rename or Change the name of file.

4. Copy the file from one location to another.

5. Sorting or Arranging the contents of file.

6. Move or copy the file from one place to another.

7. Delete a file.

Execute the file (this displays the output of the file)

## 3.4 INDEXING

An index is a table containing a list of keys and corresponding reference fields. The reference field (address) points to the record where the information referenced by the key is found. (Or) An index is a tool for finding records in a file it consists of Key field on which the index is searched and Reference field that tells particular address of the key.

- An index lets us to maintain the order of a file without rearranging the file;
- The records in the file are not sorted, but the index file is sorted.
- Indexing gives us keyed access to variable length record files.

# CHAPTER 4

# SOFTWARE REQUIREMENT AND ANALYSIS

## 4.1 Software Requirements

• Operating system: Windows 7/8/8.1/10 or Linux or macOS

• IDE: DevC++

• Compiler: GCC on Linux, Clang on macOS and GCC

• G++ compiler on windows 4.2.

## 4.2 Hardware Requirements

• Processor: Any intel or AMD processor with at least 2 cores

• RAM: minimum 2GB RAM

• Storage: approximately 500MB storage

# CHAPTER 5

# DESIGN

The covid and vaccination has 2 main menus each with different monitoring:

1. Covid menu 2. Vaccination menu

## 5.1 Covid menu:

Admin has some power. They are:

- **Add Patient Records:** Here admins can add patient record like BUNo, Name, Phone Number, Age and hospital if the person is Positive and has some symptoms like Fever,Cold,Caugh etc.
- **Displaying records**: The admins can view the records or details entered with their primary key reference.
- **Searching Patient records**: Admins can easily search the patient records using their name or buno for reference.
- **Deleting Patient records**: Records can be deleted if there is any wrong entry in the details or when they test negative.

## 5.2 Vaccination Menu:

User can book their vaccination slots using these

- **Entering Personal details:** Here users can add their personal information for the booking of the vaccination if and only if they are tested negative and don’t have any covid symptoms.
- **View Vaccine Data:** Users can view the people who have been vaccinated till the moment.
- **Searching:** Users can easily search the patient records using their name or aadaar for reference.

- **Statistics:** This displays the number of people vaccinated and total vaccinations left/available for others.
- **Slot Booking:** This books the slots for the user when they enter the time and date of booking.

# CHAPTER 6

# SYSTEM IMPLEMENTATION

Implementation is the stage where the theoretical design is turned into a working system. The most crucial stage in achieving a new successful system and in giving confidence on the new system for the users that it will work efficiently and effectively.

The system can be implemented only after thorough testing is done and if it is found to work according to the specification. It involves careful planning, investigation of existing systems and its constraints on implementations, design of methods to achieve the change. Two major tasks of preparing the implementation are education and training of the users and testing of the system.

The implementation phase consists of several activities. The required hardware and software acquisition are carried out. The system may require the need to develop a software. For this purpose, programs are written and tested.

## 6.1 Source code:

The source code of the Covid and vaccination system is given below briefly.

### 6.1.1 Menu Page:

The option page displays the users/admins to select the options based on their requirements.

1. COVID MENU
2. VACCINATION MENU
3. EXIT

## 6.1.2 Covid Menu:

In Covid menu there are different options which are described briefly below

- **Adding Patients**

```
void covid::patient_details()
{
        int k;

    cout<<"\n*************************************************
*************************************************************
********\n";
        cout<<"\n\t\t\t\t ENTERING PATIENT DETAILS \t\t\t"<<endl;

    cout<<"\n*************************************************
*************************************************************
********\n";
        cout<<" \n\t\t\t Enter the buno number=";
        cin>>dbuno;
        if(search(dbuno)>=0)
        {
          cout<<"\n\t\t\t BU Number is already present we can't add to index file\n";
          return;
        }
        for(i=indsize;i>0;i--)
        {
```

```
            if(strcmp(dbuno,id[i-1].ibuno)<0)
                id[i]=id[i-1];
            else
                break;
        }
        fopen(dfile,datafile,ios::app);
        cout<<"\n\t\t\t Enter the name=";      cin>>name;
        cout<<"\n\t\t\t Enter the age=";       cin>>age;
        cout<<"\n\t\t\t Enter the phone num=";       cin>>phone;
        cout<<"\n\t\t\t Enter the hospital=";        cin>>hospital;
        pack();
        dfile.seekg(0,ios::end);
        k=dfile.tellg();
        dfile<<buffer<<"\n";
        strcpy(id[i].ibuno,dbuno);
        sprintf(id[i].addr, "%d",k);
        indsize++;
        for(i=sindsize;i>0;i--)
        {
            if(strcmp(name,sid[i-1].sname)<0)
                sid[i]=sid[i-1];
            else        if((strcmp(name,sid[i-1].sname)==0)        &&
(strcmp(dbuno,sid[i-1].sbuno)<0))
                sid[i]=sid[i-1];
```

```
                else
                        break;
            }
            strcpy(sid[i].sname,name);
            strcpy(sid[i].sbuno,dbuno);
            sindsize++;
        system("cls");
    }
```

- **Searching patients:**

```
int covid::search(char * fbuno)
{
    int low=0,high=indsize-1,mid;
    while(low <=high)
    {
        mid = (low+high)/2;
        if(strcmp(fbuno,id[mid].ibuno)==0)
                return mid;
        if(strcmp(fbuno,id[mid].ibuno)>0)
                low=mid+1;
        else
                high=mid-1;
    }
    return -1;
}
```

```
//function to search based on secondary key
int sec_search()
{
    int pos,j,flag=-1;
    cout<<"\n**********************************************************
**************************************************************************
*************\n";
    cout<<"\n\t\t\t\t SEARCHING PATIENT DETAILS \t\t\t"<<endl;
    cout<<"\n**********************************************************
**************************************************************************
**************\n";
    cout<<"\n\t\t\t Enter the name to search(sec key):";
    cin>>skey;
    cout<<"\n\t\t\t The searched record details are :"<<endl;
    cout<<setiosflags(ios::left);
    cout<<"\t\t\t BU NO:"<<"\t\t\t Name:"<<endl;
    fopen(dfile,datafile,ios::in|ios::out);
    for(j=0;j<sindsize;j++)
                if(strcmp(skey,sid[j].sname)==0)
        {
                cout<<"\t\t\t"<<sid[j].sbuno<<"\t\t\t"<<sid[j].sname<<endl;
                flag=j;
        }
    return flag;
```

```
    system("cls");
}
```

- **Delete Patients:**

```
void covid::remove_patients()
{
    char rbuno[10];
    int pos,spos;
    cout<<"\n**********************************************************
**********************************************************************
**********\n";
    cout<<"\n\t\t\t\t REMOVING PATIENT DETAILS \t\t\t"<<endl;
    cout<<"\n**********************************************************
**********************************************************************
**********\n";
    cout<<"\n\t\t\t Enter the BU number above listed to delete:";
    cin>>rbuno;
    for(i=0;i<sindsize;i++)
    {
        if(strcmp(sid[i].sbuno,rbuno)==0)
        {
            spos=i;
            break;
        }
    }
```

```
if(strcmp(sid[spos].sname,skey)==0)
{
    pos=search(rbuno);
    dfile.seekp(atoi(id[pos].addr),ios::beg);
    dfile.put('$');
    for(i=pos;i<indsize;i++)
        id[i]=id[i+1];
    indsize--;
    for(i=spos;i<sindsize;i++)
        sid[i]=sid[i+1];
    sindsize--;
    cout<<"\n\t\t\t The patient record with BU number "<<rbuno<<"is deleted"<<endl;
}
else
    cout<<"\n\t\t\t BU number and name doesnot match";
}
```

## 6.1.3 Vaccination Menu:

- **Displaying Vaccination Details**

```
void vaccine::datadisp()
{
  cout<<"\n**************************************************
**************************************************************
*****************\n";
```

```
    cout<<"\n\t\t\t\t VACCINATION DRIVE DETAILS \t\t\t"<<endl;
    cout<<"\n**********************************************************
**************************************************************
*****************\n";
    cout<<setiosflags(ios::left);
    cout<<setw(16)<<"Aadaar"<<setw(16)<<"Name"<<setw(16)<<"Age"<<se
tw(16)<<"Gender"<<setw(16)<<"Phone"<<setw(2)<<"Address\n";
    cout<<"\n";
    while(1)
    {
        unpack();
        if(dfile1.eof())
        break;
    }
    cout<<endl<<"\n\t\t\t The index file details are "<<endl;
    cout<<setw(10)<<"Aadaar"<<setw(10)<<"address";
    for(j=0;j<indsize1;j++)
         cout<<endl<<setw(10)<<idv[j].iaadaar<<setw(10)<<idv[j].addr;
    cout<<endl<<"\n The secondary file details are " <<endl;
    cout<<setw(20)<<"Name"<<setw(15)<<"primary reference";
    for(j=0;j<sindsize1;j++)
          cout<<endl<<setw(20)<<sidv[j].sname<<setw(15)<<sidv[j].saadaar;
}
```

- **Slot Booking**

```
void vaccine ::view_vaccine()
```

```
{
   slots();
}
int slots()
{
   cout<<"\n*************************************************************
*************************************************************
****************\n";
   cout<<"\n\t\t\t\t VACCINATION DRIVE COUNT \t\t\t"<<endl;
   cout<<"\n*************************************************************
*************************************************************
****************\n";
   string line;
   ifstream file("vaccineindex.txt");
   if(file.is_open())
  {
      while(!file.eof())
   {
         getline(file,line);
         //cout<< line << endl;
         count++;
      }
      file.close();
   }
   number();
```

```
}
void number()
{
   count--;
   cout<<"\n\t\t\t Number of people vaccinated : " << count << endl;
   cout<<"\n\t\t\t Vaccinatinons left : "<<num_of_vaccine - count<<endl;
}
```

## 6.1.4 Class Inheritance:

```
class book_appointments :public vaccine
{
   struct appoint
   {
         int status;
         char start[10];
         char end[10];
         char max[10];
         char min[10];
         struct appoint *Next;
   }*head;

void create_app()
{
   int i ;
```

```
    appoint *temp, *p;


    head = NULL;


    cout<<"\n**********************************************************
**********************************************************************
*************\n";
    cout<<"\n\t\t\t\t VACCINATION DRIVE APPOINTMENTS \t\t\t"<<endl;
    cout<<"\n**********************************************************
**********************************************************************
*************\n";
    cout<<"\n\t How many Appointments : ";
    cin>>nodes;


    for(i=0; i<nodes; i++)
    {
          cout<<"\n\t NEW APPOINTMENT FOR VACCINATION ";
          temp = new(struct appoint);  //Step 1: Allocate Memory
         temp->status=0;
         cout<<"\n\t\t\t Enter Start Time: "; //Step 2: Store data and address
         cin>>temp->start;
         cout<<" \n\t\t\tEnter End Time: "; //Step 2: Store data and address
         cin>>temp->end;
         cout<<"\n\t\t\t Enter min Time: "; //Step 2: Store data and address
         cin>>temp->min;
```

```
            cout<<"\n\t\t\t Enter max Time: "; //Step 2: Store data and address
            cin>>temp->max;

        temp->Next = NULL;

        if(head == NULL)   //Step 3: Attach node in linked List
        {
          head = temp;
          p = head;
        }
        else
        {
          p->Next = temp;
          p = p->Next;
        }

      }

}
void display_SLL()
{
  appoint *p;

    cout<<"\n*************************************************
```

```
**********************************************************************
**********\n";
   cout<<"\n\t\t\t\t VACCINATION BOOKINGSLOTS \t\t\t"<<endl;
   cout<<"\n**********************************************************
**********************************************************************
**********\n";
   p = head;
   cout<<"Status\tStart Timet\tEnd Time\tMin Time\tMax Time\n";
   while(p != NULL)
   {
   if(p->status==0)
   {
   cout<<"\n\t\t\t Free";
   }
   else
   {
    cout<<"\n\t\t\t Booked";
 }
      cout<<"\t\t"<<p->start<<"\t\t"<<p->end<<"\t\t"<<p->min<<"\t\t"<<p-
>max<<"\t\t";
      p = p->Next;
   }

}
void book_app()
```

```
{
char time[10];
struct appoint *p;
cout<<"\n\t\t\t Enter The Time Slot to book appointment";
cin>>time;
p=head;
 while(p!=NULL)
 {
  if(strcmp(time,p->start) == 0)
  {
    if(p->status == 0)
    {
     p->status=1;
     cout<<"\n\t\t\t Your Appointment Is Booked\n\n";
    }
    else
    cout<<"\n\t\t\t Appointment slot is Busy\n\n";
    break;
  }
  else
   p=p->Next;
 }

 if(p==NULL)
```

```
 cout<<"\n\t\t\t Appointment slot is Not available\n\n";
 display_SLL();


}


void sort_app()
{
char str[10];
struct appoint *p;
int i;
for(i=0;i<nodes-1;i++)
 {
  p = head;
  while(p->Next!=NULL)
  {
   if(strcmp(p->start,p->Next->start)>0)
   {
    int tmp=p->status;
    p->status=p->Next->status;
    p->Next->status=tmp;


    strcpy(str,p->start);
    strcpy(p->start,p->Next->start);
    strcpy(p->Next->start,str);
```

```
    strcpy(str,p->end);
    strcpy(p->end,p->Next->end);
    strcpy(p->Next->end,str);
    strcpy(str,p->min);
    strcpy(p->min,p->Next->min);
    strcpy(p->Next->min,str);
    strcpy(str,p->max);
    strcpy(p->max,p->Next->max);
    strcpy(p->Next->max,str);
   }
   p=p->Next;
  }
 }
  cout<<"\n\t\t\t SORTED\n";
  display_SLL();
}
public:
int book_menu()
{
    create_app();
display_SLL();
book_app();
sort_app()
return 0; }};
```

# CHAPTER 7

# TESTING

Software testing is defined as an activity to check whether the actual results match the expected results and to ensure that the software system is defect free.

## 7.1 Unit Testing

It focuses on smallest unit of software design. In this we test an individual unit or graph of inter related units. It is often done by programmer by using sample input and observing its corresponding outputs.

Example:

a. In a program we are checking if loop, method or if a function is working correctly.

b. Incorrect or misunderstood arithmetic procedures.

c. Incorrect initialization. In this project we have implemented and tested login for different users like student, librarian as each unit.

## 7.2 Integration Testing

When a software test case covers more than one unit, it is considered an integration test. When developing a software test case, the lines between units can quickly evolve into integration tests. The dependency itself will not need to be tested and the integration to it will be mocked or faked.

It is of two types: (i) Top Down (ii) Bottom up

Example:

a. Black Box testing –It is used for validation. Here we ignore internal working mechanism and focus on what the output would be.

b. White Box testing- It is used for verification. Here we focus on internal mechanism i.e. how the output is obtained.

## 7.2 System Testing

In this the software is tested such that it works fine for different operating systems. It is covered under the black box testing technique. In this we focus on required inputs and outputs without focusing on internal working.

| FUNCTIONALITY | ACTION | EXPECTED OUTPUT | ACTUAL RESULT | TEST RESULT |
|---|---|---|---|---|
| Checking if person has the given symptoms of covid | If YES, then proceed with further operations | Admin can create patient details | Admin gets the access to create patient records | Pass |
| Checking if person has the given symptoms of covid | If NO, then cannot have access for other operations | It must return to the main menu page | It returns to the main menu page | Pass |
| Checking if person is covid positive or negative for vaccination process | If YES, then cannot apply for vaccination drive | It must return to the main menu page | It returns to the main menu page | Pass |
| Checking if person is covid positive or negative for vaccination process | If NO, then proceed with further operations | User can start with the process by applying personal details | User can access all the operations of drive | Pass |

# CHAPTER 8

# RESULT

- **MENU**

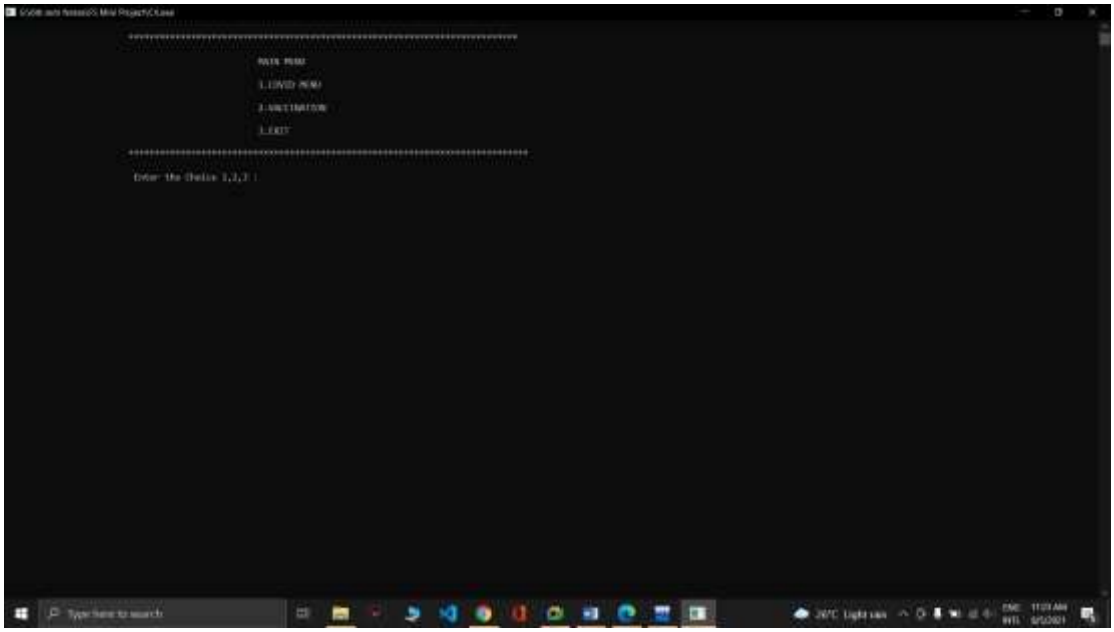

Fig. 8.1 MENU PAGE

- **COVID MENU**

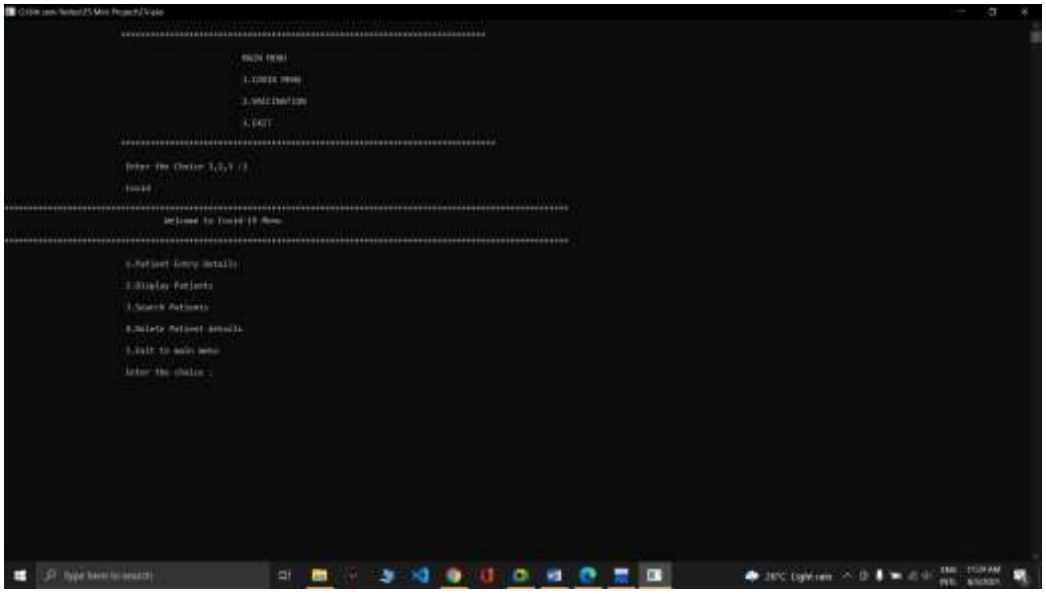

Fig. 8.2 Covid menu

- **CREATE PATIENT DETAILS**

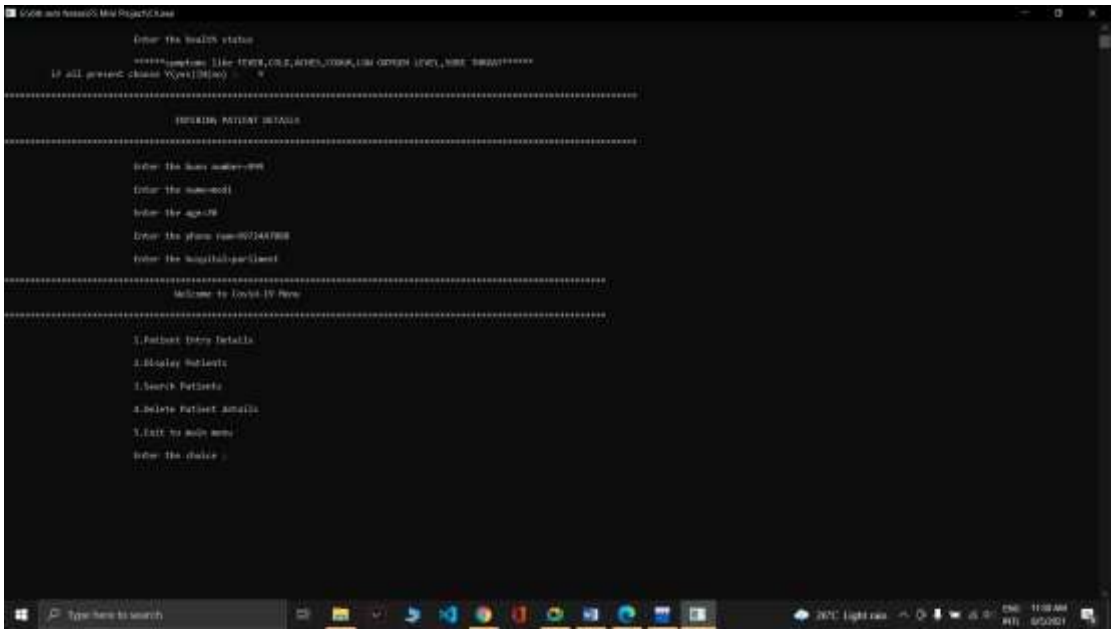

Fig. 8.2 Patient entry details

- **DISPLAY DETAILS**

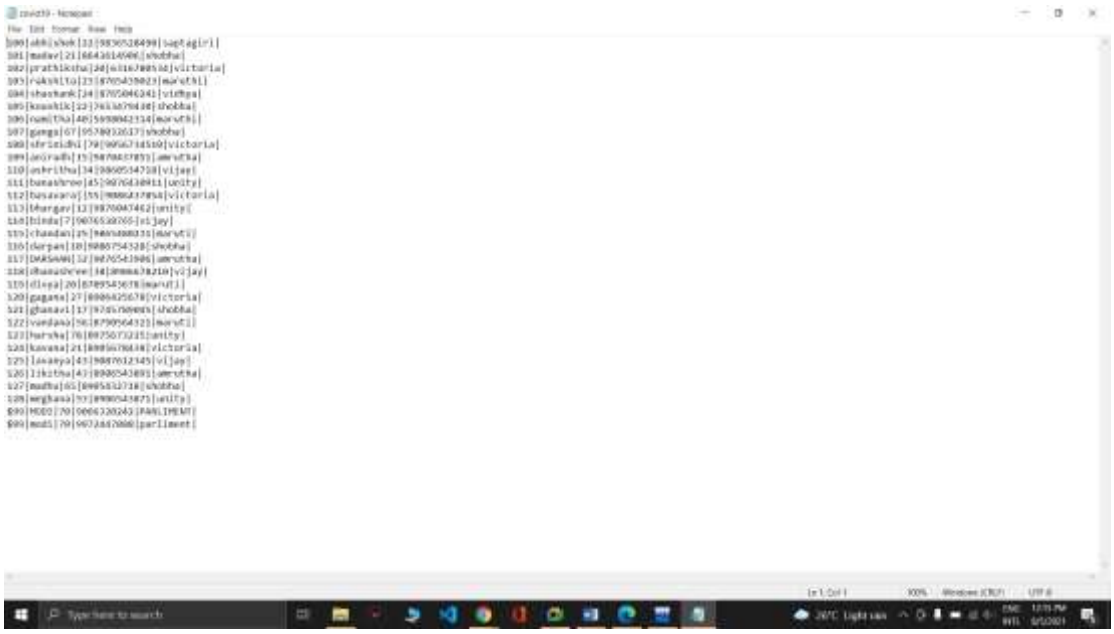

Fig. 8.3.1 Displaying patient details from data file

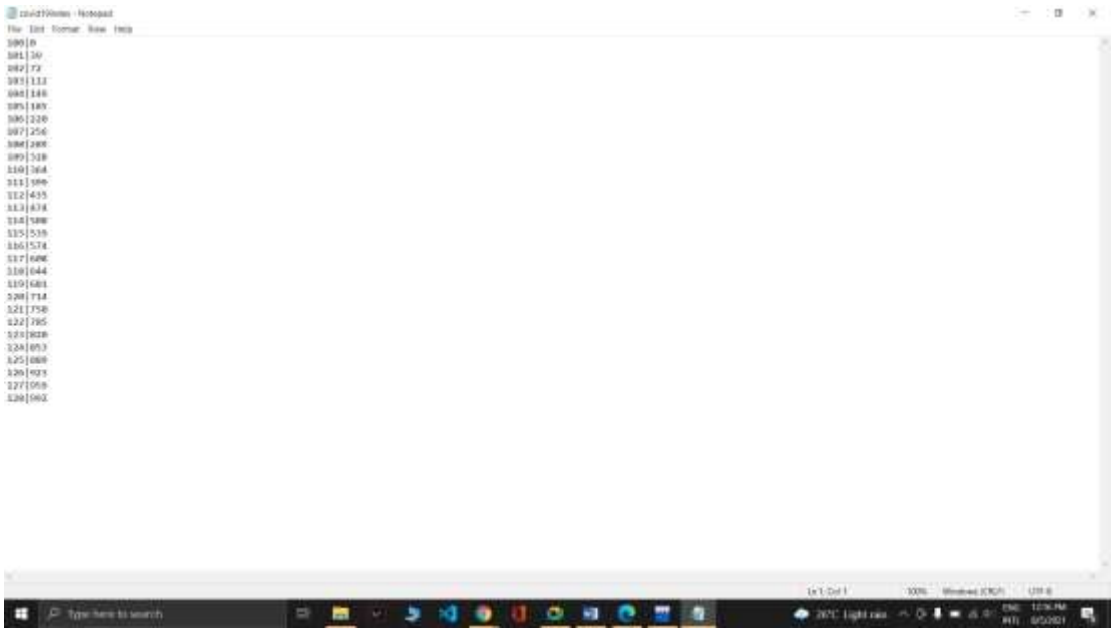

Fig. 8.3.2 Displaying patient details from index file

- **SEARCH DETAILS**

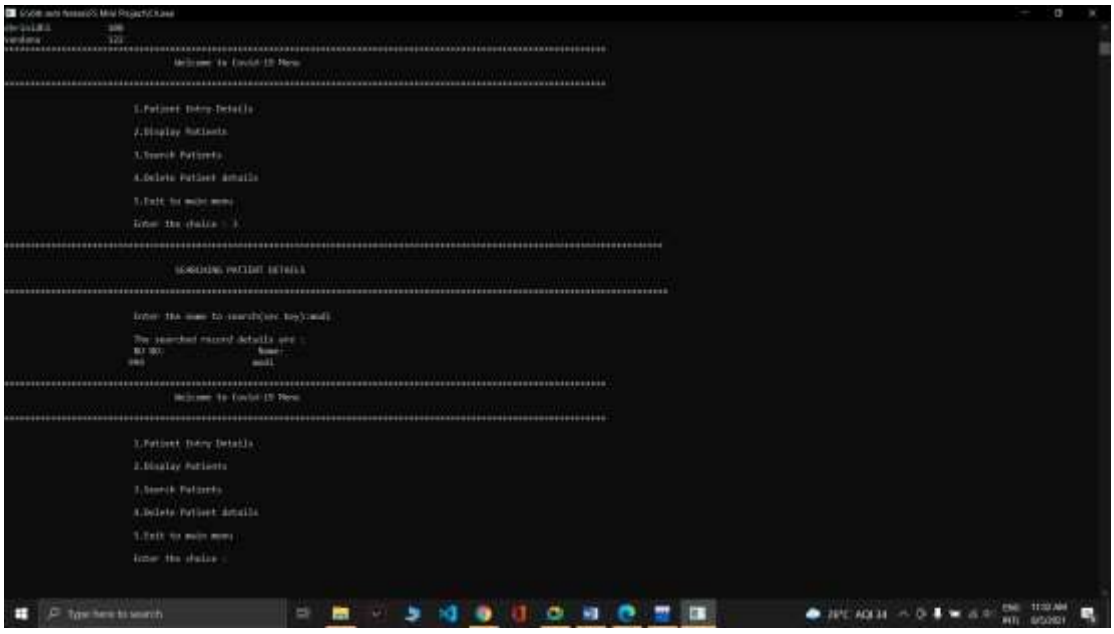

Fig. 8.4 Searching patient details

- **REMOVE DETAILS**

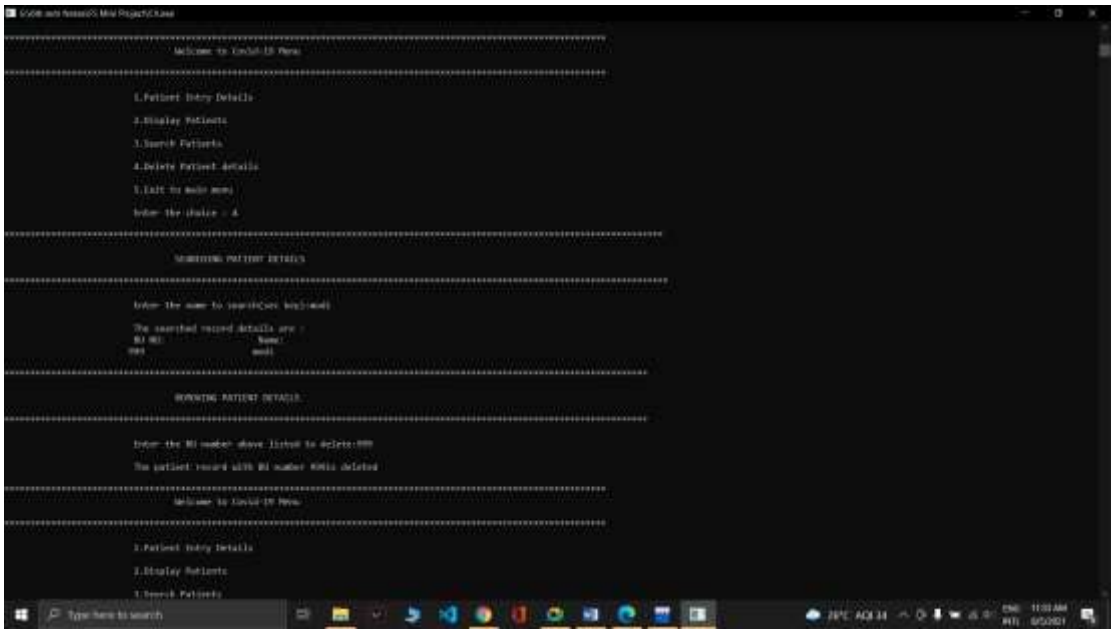

Fig. 8.5 Removing patient details

- **VACCINATION MENU**

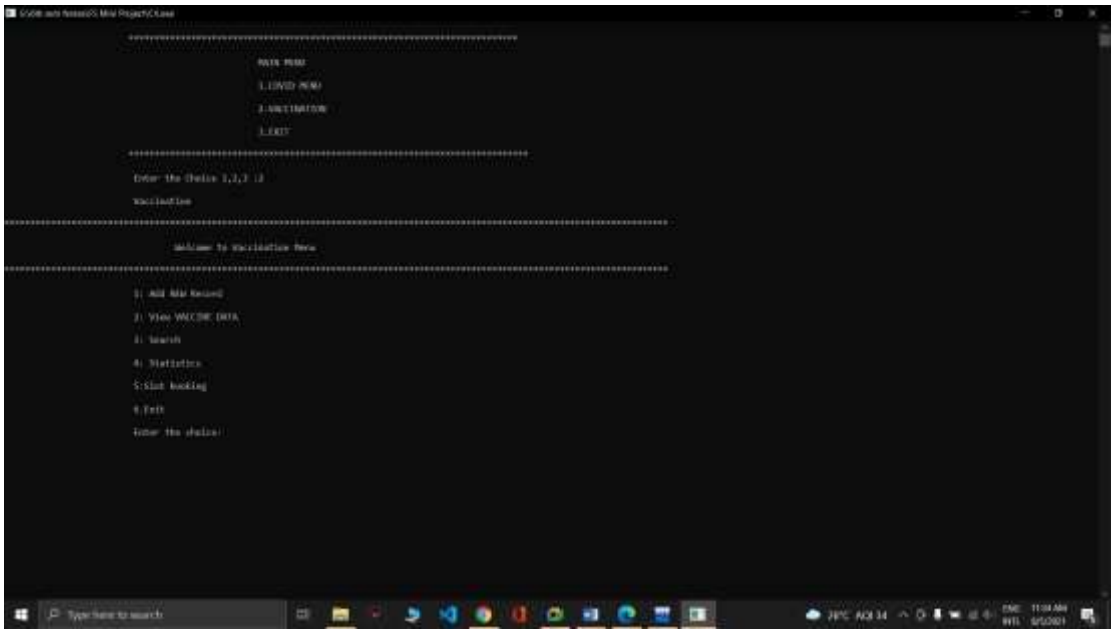

Fig. 8.6 Vaccination Menu

- **USER DETAILS**

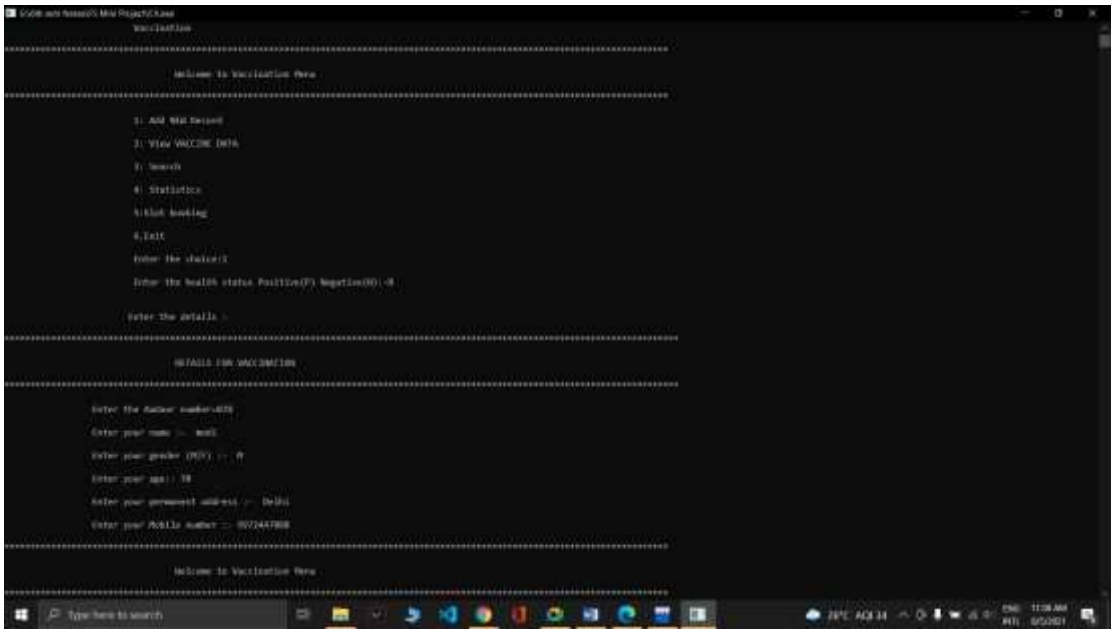

Fig. 8.7 Adding New records

- **DISPLAY VACCINATION DETAILS**

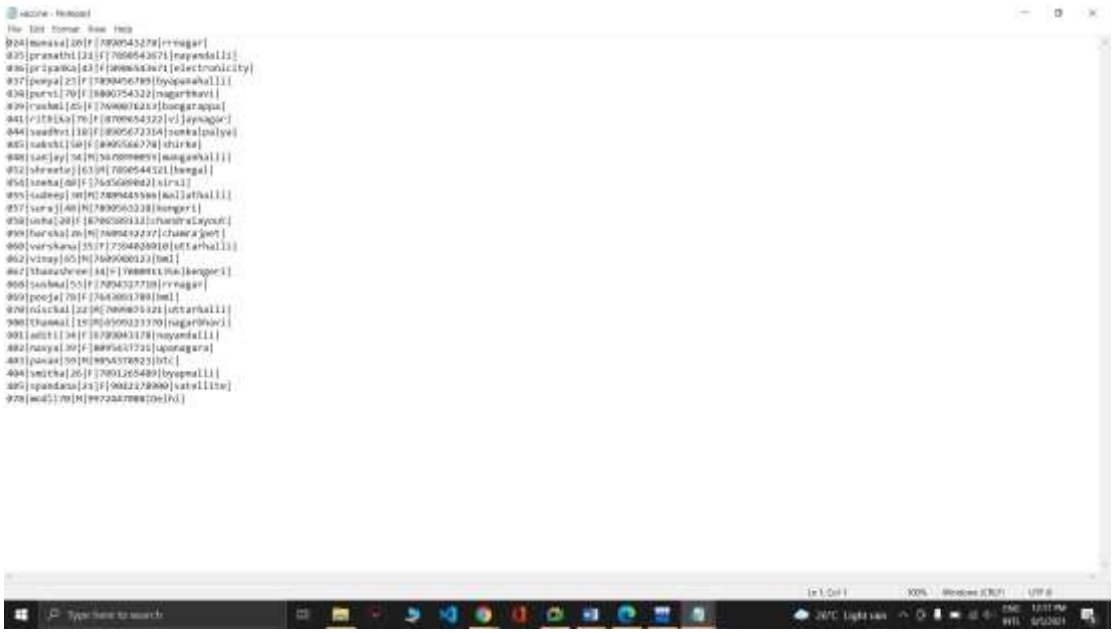

Fig. 8.8.1 Displaying vaccinated people from data file

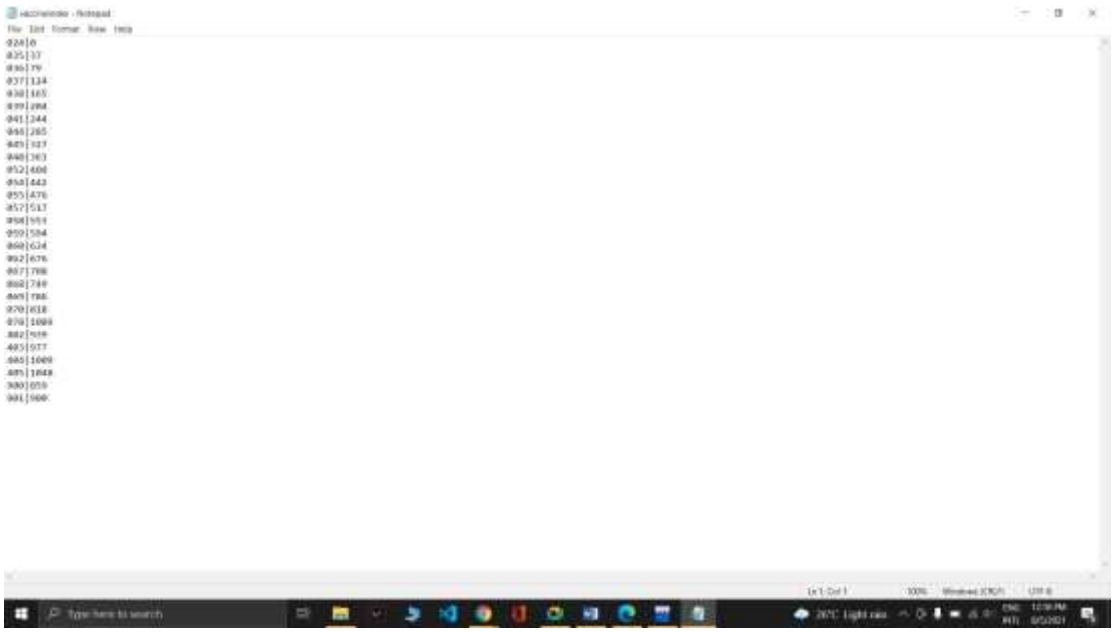

Fig. 8.8.2 Displaying vaccinated people from index file

- **SEARCH USER DETAILS**

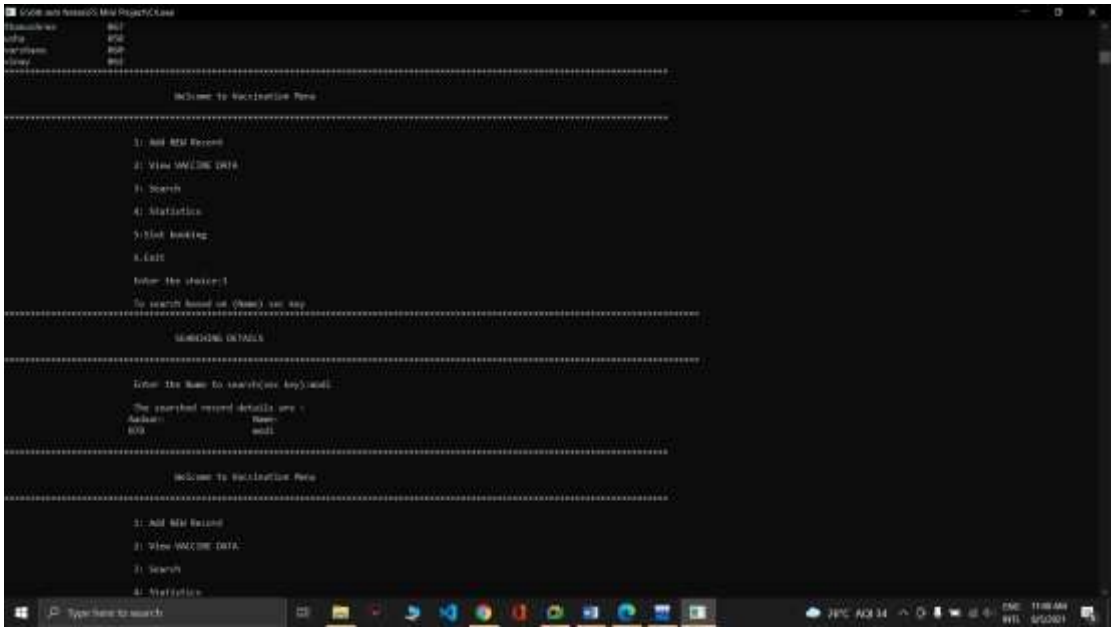

Fig. 8.9 Searching people who are vaccinated

- **STATISTICS**

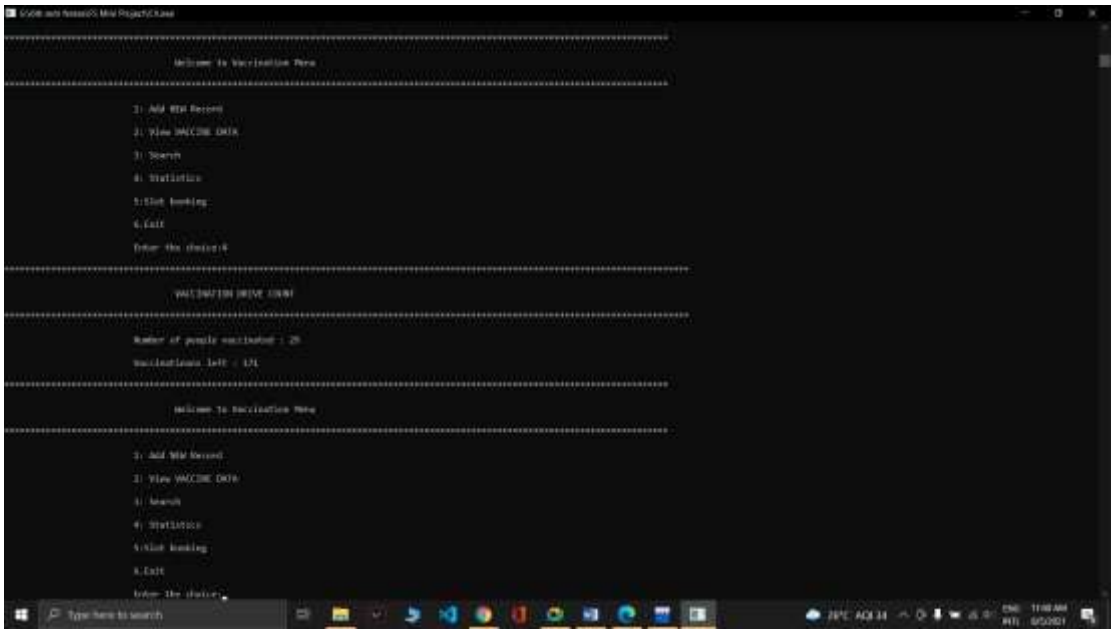

Fig. 8.10 No of people vaccinated and available vaccinations

- **BOOKING SLOTS**

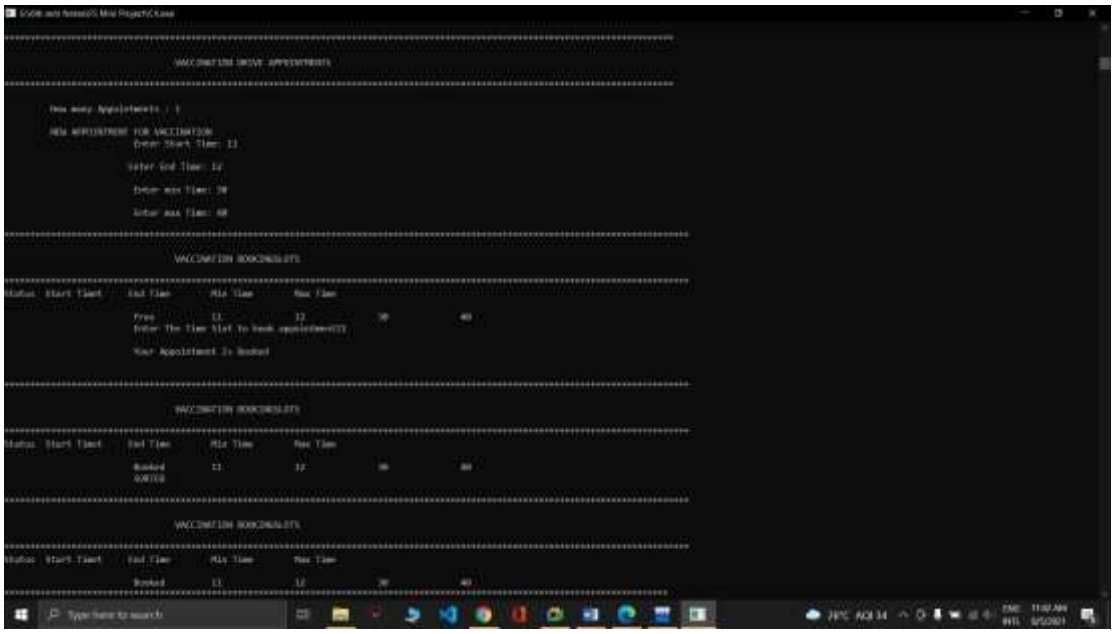

Fig. 8.11 Slot booking

# CHAPTER 9

# CONCLUSION

From a proper analysis of positive points and constraints on the project, it can be concluded that the application makes the entire process online. It is a highly efficient UI based component which meets all the requirements of the society. Admin can manage entities to store the patient records by searching, inserting or deleting or viewing the records. User can add their personal information to book the vaccination and its appointments and view the statistics of the drive. Admin can see details of a particular patient whether he/she is positive. User can see the details of the people vaccinated.

We can enhance the system by including more facilities like:

- Updating the records if wrong entry
- Automate the discharge date
- Enhanced UI design
- More entities if required

# REFERENCES

[1]: Michael J. Folk, Bill Zoellick, Greg Riccardi: File Structures-An Object Oriented Approach with C++, 3rd Edition, Pearson Education, 1998.

[2]: K.R. Venugopal, K.G. Srinivas, P.M. Krishnaraj: File Structures Using C++, Tata McGraw-Hill, 2008.

[3]: Scot Robert Ladd: C++ Components and Algorithms, BPB Publications, 1993.

[4]: Raghu Ramakrishan and Johannes Gehrke: Database Management Systems, 3rd Edition, McGraw Hill, 2003.

[5]:www.w3schools.com

[6]:www.geeksforgeeks.com

[7]:www.tutorialpoints.com

[8]:www.youtube.com