A Micro Project Report On

CONTACT MANAGEMENT SYSTEM

Submitted to the CMR Institute of Technology in partial fulfilment of the requirement of the award of the laboratory of

DATA STRUCTURES THROUGH PYTHON

I-B.Tech. II Semester

DEPARTMENT OF FRESHMAN ENGINEERING

Submitted by

Parinita Malisetty	**(23R01A66X2)**
Nalacheruvula Varsha	**(23R01A66W8)**
G.Vishnu Vardhan	**(23R01A66V2)**
D.Dhanush Rao	**(23R01A66U5)**
Bagotham Harshith	**(23R01A66T2)**
Mohammad Ilyas	**(23R01A66W4)**

Under the Guidance of
Mrs.G Lakshmi Praveena
(Assistant Professor)
[Freshman Engineering]

CMR INSTITUTE OF TECHNOLOGY

(UGC AUTONOMOUS)

(Approved by AICTE, Affiliated to JNTU, Kukatpally, Hyderabad)

Kandlakoya, Medchal Road, Hyderabad. (2023-2024)

CMR INSTITUTE OF TECHNOLOGY

(UGC AUTONOMOUS)

(Approved by AICTE, Affiliated to JNTU, Kukatpally,Hyderabad)

Kandlakoya, Medchal Road, Hyderabad

DEPARTMENT OF FRESHMAN ENGINEERING

CERTIFICATE

This is to certify that a Micro Project entitled with

"CONTACT MANAGEMENT SYSTEM"

is being submitted by

Parinita Malisetty	**(23R01A66X2)**
Nalacheruvula Varsha	**(23R01A66W8)**
G.Vishnu Vardhan	**(23R01A66V2)**
D.Dhanush Rao	**(23R01A66U5)**
Bagotham Harshith	**(23R01A66T2)**
Mohammad Ilyas	**(23R01A66W4)**

In partial fulfilment of the requirement for the award of the laboratory of I-B.Tech. II Semester in CSM towards a record of Bonafide work carried out under guidance and supervision.

Signature of guide Mrs.G Lakshmi Praveena (Assistant Professor)	Signature of Coordinator Dr.M.Ravi (Professor)	Signature of HOD Dr.Radha Krishna Reddy (Head Of Department)

ACKNOWLEDGEMENT

We are extremely grateful to Dr. M. Janga Reddy, Director, Dr. B. Satyanarayana, Principal and Mr.Radha Krishna Reddy, Head of Department, Department of FRESHMAN ENGINEERING, CMR Institute of Technology for their inspiration and valuable guidance during entire duration.

We are extremely thankful to our Data Structures Through Python Laboratory in-charge and Faculty, Computer Science and Engineering Department, CMR Institute of Technology for their constant guidance encouragement, and moral support throughout the project.

We express our thanks to all staff members and friends for all the help and coordination extended in bringing out this Project successfully in time.

Finally, we are very much thankful to our parents and relatives who guided directly or indirectly for successful completion of project.

Parinita Malisetty (23R01A66X2)

Nalacheruvula Varsha (23R01A66W8)

G.Vishnu Vardhan (23R01A66V2)

D.Dhanush Rao (23R01A66U5)

Bagotham Harshith (23R01A66T2)

Mohammad Ilyas (23R01A66W4)

TABLE OF CONTENTS

S.NO	CONTENT	PAGE NO.
1	ABSTRACT	1-2
2	INTRODUCTION	3
3	PROJECT OVERVIEW	4
4	FEATURES OF CONTACT MANAGEMENT SYSTEM	5
7	IMPLEMENTATION AND TESTING OF CONTACT MANAGEMENT SYSTEM	6
8	ALGORITHM	7-8
9	SOURCE CODE	9-16
10	OUTPUT	17-20
11	CONCLUSION	21
12	REFERENCE	21

ABSTRACT

- In today's interconnected world, managing contacts efficiently is paramount for personal and professional success. This abstract outlines the development and implementation of a Contact Management System (CMS) designed to streamline the process of organizing, storing, and accessing contact information. The CMS aims to address the limitations of traditional contact management methods by leveraging technological advancements to provide a user-friendly and feature-rich solution.

- The implementation of the CMS involves several key components, including database design, user interface development, and integration of essential features such as contact categorization, search functionality, and data synchronization across multiple devices. The system is designed to be scalable, allowing for the seamless addition of new features and the accommodation of growing user bases.

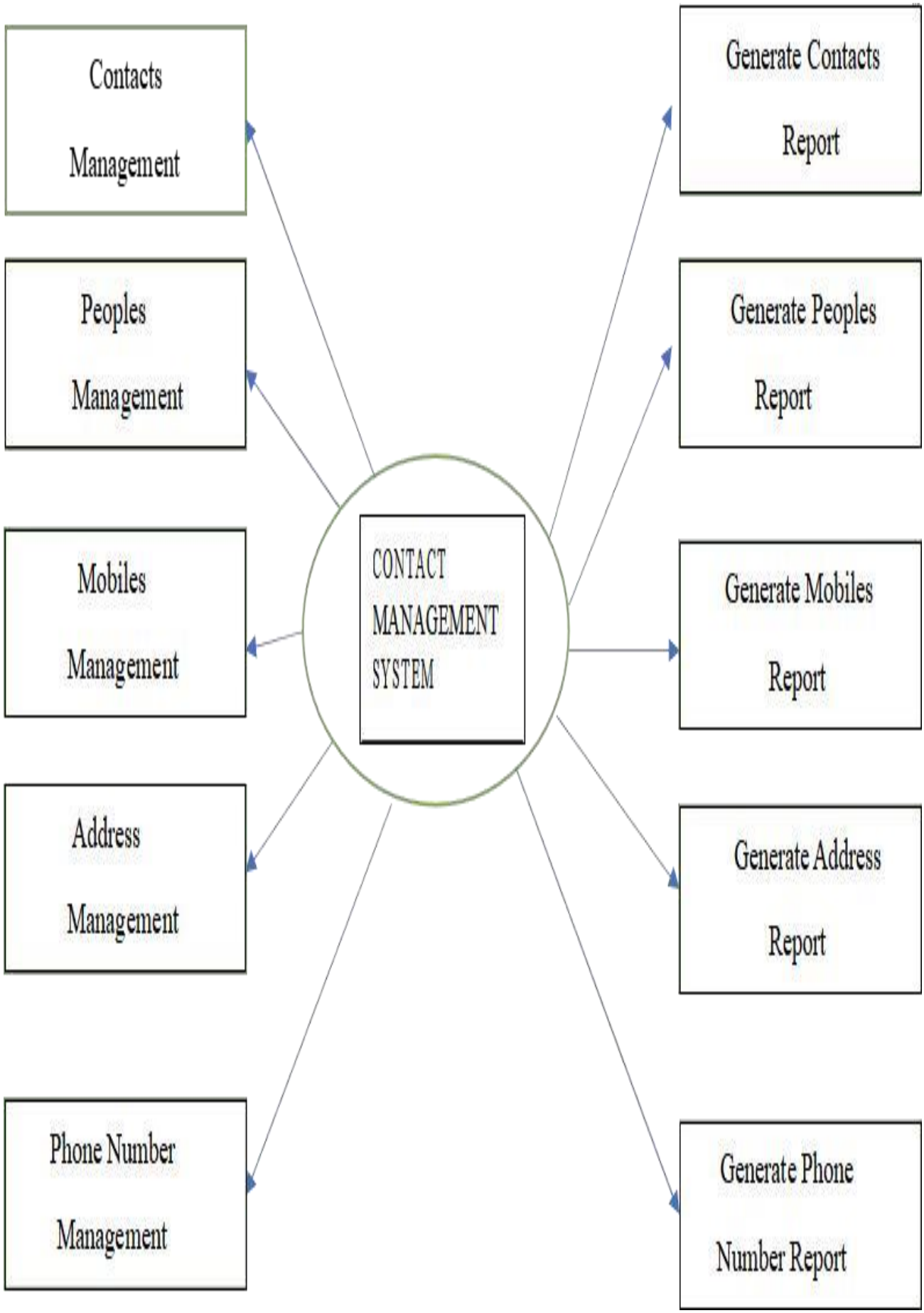
- A Contact Management System project report incorporates a python script (Contact System.py). Contact management System in Python is the process of keeping track of the information about contacts and keeping track of how they interact with a business. This is a simple GUI based mission that's very clean to apprehend and use.

- The Features Used in this Contact Management System are

1. **Add Contact**
2. **remove an exist contact**
3. **Delete all contacts**
4. **Search for a contact**
5. **Display all contacts**
6. **Exit ( )**

- This GUI based Contact Management system provides the simplest management of contact details. In short, this projects mainly focus on CRUD. There's an external database connection file used in this mini project to save user's data permanently. In order to run the project, you must have installed python on your PC. This is a simple GUI Based system, specially written for the beginners.

- In conclusion, the implementation of a Contact Management System offers substantial benefits in terms of efficiency, productivity, and organization. By leveraging technology to streamline contact management processes, individuals and organizations can enhance their ability to connect, collaborate, and succeed in today's dynamic digital landscape.


Contacts Management
Peoples Management
Mobiles Management
Address Management
Phone Number Management
CONTACT MANAGEMENT SYSTEM
Generate Contacts Report
Generate Peoples Report
Generate Mobiles Report
Generate Address Report
Generate Phone Number Report

INTRODUCTION

- Contact management is the process of recording contacts' details and tracking their interactions with a business. Such systems have gradually evolved into an aspect of customer relationship management (CRM) systems, which allow businesses to improve sales and service levels leveraging a wider range of data.
- By combining the tracking of contacts, their interactions with the business, and their preferences and service issues, it became possible to create a single, unified view of the customer -- vital data not only for a successful sales team, but also for the delivery of excellent customer service.
- To maximize the value of this data, what was needed was a fundamental move beyond a sophisticated contact book, and towards processes and systems that track the entire customer journey from start to end – and then connect those to products.
- Contact management evolved into customer relationship management as a result of two key developments:

  1. The shift in focus from contact details to relationships.
  2. The move from individual desktop databases to sharing information businesswide.
- Free contact management software, or packages that came bundled with other business utilities such as spreadsheets, made it easy to store and retrieve contact information.
- But contact management tools and programmes have been integrated into and superseded by CRM systems that can track everything from customers and sales leads to marketing campaigns and sales team performance.

PROJECT OVERVIEW

PROJECT NAME	Contact Management System Project in Python
ABSTRACT	This is a GUI-based program in python that basically includes the use of the Tkinter and Sqlite3 database for the execution.
LANGUAGES/TECHNOLOGIES USED	Python
IDE	pycharm (Recommended)
DATA BASE	SQLite3
PYTHON VERSION	3.8 or 3.9
CATEGORY	Data Structures using Python

➢ HOW TO RUN:-

- First you need to install Python
- Install some module: - Os, csv, datetime
- Download the code
- Save code with .py extension
- Create a CSV file
- Save both files in the same folder
- Now run the code

➢ FEATURES OF CONTACT MANAGEMENT SYSTEM:

1.Creating and Changing Issues at Ease

2.Query Issue List to any Depth

3.Reporting and charting in more Comprehensive way

4.User Accounts to Control the access and Maintain Security

5.Simple Status and Resolutions

6.Multilevel Priorities and Severities

7.Targets and Milestones for guiding the Programmers.

8.Attachments and Additional Comments for more information

9.Robust database back-end

10.Various Number of Reports Available with number of filter criteria's

11.It contains better storage capacity

12. Accuracy in work

13. Easy and Fast Retrieval of Information

14.Well Designed Reports

15.Access of Information Individually

➢ FUNCTIONALITIES PROVIDED BY THE CONTACT MANAGEMENT SYSTEM ARE AS FOLLOWS:

1.Provides the searching facilities based on various factors. Such as Contract, Telephone, Profile, Emails

2.Contact Management System also manage the mobile details online for Profile details, Email details, Contacts.

3.It tracks all the information of Credential, Mobile, Profile, etc.

4.Shows the information and description of the contact, Telephone

5.To increase efficiency of managing the contact, credential

6.It deals with monitoring the information and transactions of profile.

7.Managing the information of contact

8.Editing, adding, and uploading of Records is improved which results in proper resource management of contact data.

IMPLEMENTATION AND TESTING OF CONTACT MANAGEMENT SYSTEM:

Implementing and testing a Contact Management System (CMS) in Python involves several steps, including setting up the development environment, designing the system architecture, writing code for CRUD operations, implementing user authentication, and conducting comprehensive testing. Below is a general outline of the implementation and testing process:

1. Setting Up the Development Environment:

- Install Python: Download and install Python from the official website.
- Choose a Web Framework: Select a web framework like Flask or Django for building the CMS.
- Set Up Virtual Environment: Create a virtual environment to manage dependencies and isolate the project environment.

2. **Designing System Architecture**:

- Define Data Model: Determine the structure of the contact data (e.g., name, email, phone number) and create a database schema.
- Design API Endpoints: Plan the API endpoints for performing CRUD operations on contacts.
- User Authentication: Decide on the authentication mechanism (e.g., session-based, token-based) and design user authentication flows.

3. **Writing Code for CRUD Operations**:

- Create Flask Application: Initialize a Flask application and configure routes for CRUD operations.
- Implement Database Operations: Write functions to interact with the database for creating, reading, updating, and deleting contacts.
- Form Validation: Implement form validation to ensure data integrity and prevent malicious inputs.

4. Implementing User Authentication:

- Set Up User Model: Define a user model to store user credentials and other relevant information.
- Authentication Routes: Create routes for user registration, login, logout, and password management.
- Authentication Middleware: Implement middleware to protect routes that require authentication.

5. Conducting Comprehensive Testing:

- Unit Testing: Write unit tests for individual components such as database operations, authentication logic, and API endpoints.
- Integration Testing: Test the interaction between different components of the CMS to ensure seamless functionality.
- End-to-End Testing: Conduct end-to-end tests to simulate user interactions and verify the overall system behavior.
- Security Testing: Perform security testing to identify and mitigate vulnerabilities such as SQL injection, cross-site scripting (XSS), and session hijacking.

6. Documentation and Maintenance:

- Document the CMS architecture, API endpoints, and deployment instructions for reference.
- Regularly update and maintain the CMS to address bugs, security vulnerabilities, and user feedback.

Throughout the implementation and testing process, it's essential to follow best practices such as code modularity, error handling, and adherence to coding standards to ensure the reliability, scalability, and security of the Contact Management System.

ALGORITHM

Designing an algorithm for a **Contact Management System (CMS)** in Python involves outlining the steps required to perform various operations such as adding, updating, deleting, and searching contacts. Below is a high-level algorithm for implementing basic CRUD operations in a CMS:

1. Initialization:

- Initialize the CMS application.
- Set up the database connection.

2. User Authentication:

- Prompt the user to log in or register.
- Verify user credentials against the database.
- Provide options for password recovery or account management.

3. CRUD Operations:

a. Create Contact:

- Accept input for contact details (name, email, phone number, etc.).
- Validate input data to ensure correctness and completeness.
- Insert the new contact into the database.

b. Read Contact:

- Display options for viewing all contacts or searching for a specific contact.
- Retrieve contact information from the database based on user input.
- Display contact details to the user.

c. Update Contact:

- Prompt the user to select a contact to update.
- Allow the user to modify contact details.
- Update the corresponding record in the database with the new information.

d. Delete Contact:

- Prompt the user to select a contact to delete.
- Confirm the deletion request.
- Remove the selected contact from the database.

4. Search Functionality:

- Provide options for searching contacts by name, email, phone number, etc.
- Retrieve matching contacts from the database.
- Display search results to the user.

5. Error Handling:

- Implement error handling mechanisms to handle exceptions gracefully.
- Display appropriate error messages to the user in case of invalid input or database errors.

6. **User Interface**:
- Design a user-friendly interface to interact with the CMS.
- Incorporate navigation menus, input forms, and feedback messages for a smooth user experience.

7. **Security Measures**:
- Implement security measures such as password hashing and encryption to protect user data.
- Guard against common security threats such as SQL injection and cross-site scripting (XSS).

8. **Testing**:
- Test each CRUD operation to ensure that it functions as expected.
- Conduct unit tests, integration tests, and end-to-end tests to validate the CMS functionality.

9. Documentation:
- Document the CMS algorithm, including the steps for each operation and any dependencies.
- Provide instructions for setting up and using the CMS.

10. Deployment:
- Deploy the CMS to a production environment for use by end-users.
- Monitor performance and security metrics to ensure the system's reliability and stability.

This algorithm serves as a roadmap for developing a basic Contact Management System in Python. Depending on specific requirements and features, additional steps and functionalities may need to be included in the algorithm.

SOURCE CODE:

```
# importing the module
import sys
# this function will be the first to run as soon as the main function executes
def initial_phonebook():
    rows, cols = int(input("Please enter initial number of contacts: ")), 5

    # We are collecting the initial number of contacts the user wants to have in
the
    # phonebook already. User may also enter 0 if he doesn't wish to enter any.
    phone_book = []
    print(phone_book)
    for i in range(rows):
        print("\nEnter contact %d details in the following order (ONLY):" %
(i+1))
        print("NOTE: * indicates mandatory fields")
        print(".....")
        temp = []
        for j in range(cols):

        # We have taken the conditions for values of j only for the
personalized fields
        # such as name, number, e-mail id, dob, category etc
            if j == 0:
                temp.append(str(input("Enter name*: ")))

                # We need to check if the user has left the name empty as
its mentioned that
                # name & number are mandatory fields.
                # So implement a condition to check as below.
                if temp[j] == '' or temp[j] == ' ':
                    sys.exit(
                        "Name is a mandatory field. Process exiting
due to blank field...")
                    # This will exit the process if a blank field is
encountered.
            if j == 1:
                temp.append(int(input("Enter number*: ")))
                # We do not need to check if user has entered the number
because int automatically
```

```
                    # takes care of it. Int value cannot accept a blank as that
counts as a string.
                    # So process automatically exits without us using the sys
package.
                if j == 2:
                    temp.append(str(input("Enter e-mail address: ")))
                    # Even if this field is left as blank, None will take the
blank's place
                    if temp[j] == '' or temp[j] == ' ':
                        temp[j] = None

                if j == 3:
                    temp.append(str(input("Enter date of birth(dd/mm/yy): ")))
                    # Whatever format the user enters dob in, it won't make a
difference to the compiler
                    # Only while searching the user will have to enter query
exactly the same way as
                    # he entered during the input so as to ensure accurate
searches
                    if temp[j] == '' or temp[j] == ' ':

                    # Even if this field is left as blank, None will take the
blank's place
                        temp[j] = None
                if j == 4:
                    temp.append(
                        str(input("Enter
category(Family/Friends/Work/Others): ")))
                    # Even if this field is left as blank, None will take the
blank's place
                    if temp[j] == "" or temp[j] == ' ':
                        temp[j] = None

            phone_book.append(temp)
            # By this step we are appending a list temp into a list phone_book
            # That means phone_book is a 2-D array and temp is a 1-D array

        print(phone_book)
        return phone_book
```

```
def menu():
    # We created this simple menu function for
    # code reusability & also for an interactive console

# Menu func will only execute when called
    print("**************************************************
**********")
    print("\t\t\tSMARTPHONE DIRECTORY", flush=False)
    print("**************************************************
**********")
    print("\tYou can now perform the following operations on this
phonebook\n")
    print("1. Add a new contact")
    print("2. Remove an existing contact")
    print("3. Delete all contacts")
    print("4. Search for a contact")
    print("5. Display all contacts")
    print("6. Exit phonebook")

    # Out of the provided 6 choices, user needs to enter any 1 choice among
the 6
    # We return the entered choice to the calling function wiz main in our case
    choice = int(input("Please enter your choice: "))

    return choice

def add_contact(pb):
    # Adding a contact is the easiest because all you need to do is:
    # append another list of details into the already existing list
    dip = []
    for i in range(len(pb[0])):
        if i == 0:
            dip.append(str(input("Enter name: ")))
        if i == 1:
            dip.append(int(input("Enter number: ")))
        if i == 2:
            dip.append(str(input("Enter e-mail address: ")))
        if i == 3:
            dip.append(str(input("Enter date of birth(dd/mm/yy): ")))
        if i == 4:
            dip.append(
```

```
                    str(input("Enter category(Family/Friends/Work/Others): ")))
        pb.append(dip)
        # And once you modify the list, you return it to the calling function wiz main, here.

return pb

def remove_existing(pb):
        # This function is to remove a contact's details from existing phonebook
        query = str(
                input("Please enter the name of the contact you wish to remove: "))
        # We'll collect name of the contact and search if it exists in our phonebook

        temp = 0
        # temp is a checking variable here. We assigned a value 0 to temp.

        for i in range(len(pb)):
                if query == pb[i][0]:
                        temp += 1
                        # Temp will be incremented & it won't be 0 anymore in this function's scope

                        print(pb.pop(i))
                        # The pop function removes entry at index i

                        print("This query has now been removed")
                        # printing a confirmation message after removal.
                        # This ensures that removal was successful.
                        # After removal we will return the modified phonebook to the calling function
                        # which is main in our program

                        return pb
        if temp == 0:
                # Now if at all any case matches temp should've incremented but if otherwise,
                # temp will remain 0 and that means the query does not exist in this phonebook
                print("Sorry, you have entered an invalid query.\
        Please recheck and try again later.")
```

```
        return pb

def delete_all(pb):
        # This function will simply delete all the entries in the phonebook pb
        # It will return an empty phonebook after clearing
        return pb.clear()

def search_existing(pb):
        # This function searches for an existing contact and displays the result
        choice = int(input("Enter search criteria\n\n\
        1. Name\n2. Number\n3. Email-id\n4. DOB\n5. Category(Family/Friends/Work/Others)\
        \nPlease enter: "))
        # We're doing so just to ensure that the user experiences a customized search result

        temp = []
        check = -1

        if choice == 1:
        # This will execute for searches based on contact name
                query = str(
                        input("Please enter the name of the contact you wish to search: "))
                for i in range(len(pb)):
                        if query == pb[i][0]:
                                check = i
                                temp.append(pb[i])

        elif choice == 2:
        # This will execute for searches based on contact number
                query = int(
                        input("Please enter the number of the contact you wish to search: "))
                for i in range(len(pb)):
                        if query == pb[i][1]:
                                check = i
                                temp.append(pb[i])

        elif choice == 3:
```

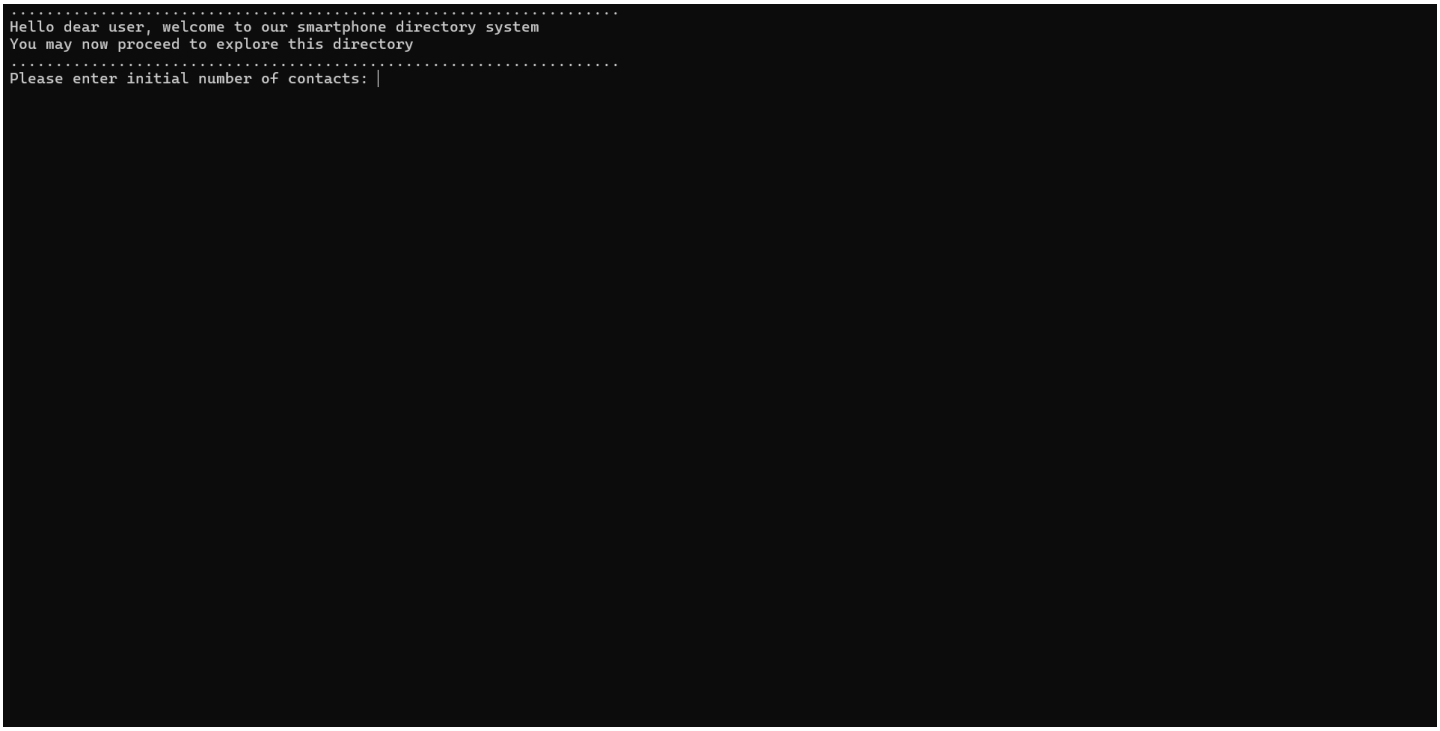
```
        # This will execute for searches based on contact's e-mail address
                query = str(input("Please enter the e-mail ID\
                of the contact you wish to search: "))
                for i in range(len(pb)):
                        if query == pb[i][2]:
                                check = i
                                temp.append(pb[i])

        elif choice == 4:

# This will execute for searches based on contact"s date of birth
                query = str(input("Please enter the DOB (in dd/mm/yyyy format ONLY)\
                        of the contact you wish to search: "))
                for i in range(len(pb)):
                        if query == pb[i][3]:
                                check = i
                                temp.append(pb[i])

        elif choice == 5:
        # This will execute for searches based on contact category
                query = str(
                        input("Please enter the category of the contact you wish to search: "))
                for i in range(len(pb)):
                        if query == pb[i][4]:
                                check = i
                                temp.append(pb[i])
                # All contacts under query category will be shown using this feature

        else:
        # If the user enters any other choice then the search will be unsuccessful
                print("Invalid search criteria")
                return -1
        # returning -1 indicates that the search was unsuccessful

        # all the searches are stored in temp and all the results will be displayed with
        # the help of display function

        if check == -1:
```

```
            return -1
            # returning -1 indicates that the query did not exist in the directory
        else:
            display_all(temp)
            return check
            # we're just returning a index value wiz not -1 to calling function just
to notify
            # that the search worked successfully

# this function displays all content of phonebook pb
def display_all(pb):

if not pb:
        # if display function is called after deleting all contacts then the len will be
0
        # And then without this condition it will throw an error
            print("List is empty: []")
        else:
            for i in range(len(pb)):
                print(pb[i])

def thanks():
# A simple gesture of courtesy towards the user to enhance user experience
        print("**********************************************************
**********")
        print("Thank you for using our Smartphone directory system.")
        print("Please visit again!")
        print("**********************************************************
**********")
        sys.exit("Goodbye, have a nice day ahead!")

#       Main        function        code
print("..................................................................")
print("Hello dear user, welcome to our smartphone directory system")
print("You may now proceed to explore this directory")
print("..................................................................")
# This is solely meant for decoration purpose only.
# You're free to modify your interface as per your will to make it look
interactive

ch = 1
```

```
pb = initial_phonebook()
while ch in (1, 2, 3, 4, 5):
        ch = menu()
        if ch == 1:
                pb = add_contact(pb)
        elif ch == 2:
                pb = remove_existing(pb)
        elif ch == 3:
                pb = delete_all(pb)
        elif ch == 4:
                d = search_existing(pb)
                if d == -1:
                        print("The contact does not exist. Please try again")

elif ch == 5:
                display_all(pb)
        else:
                thanks()
```

OUTPUT


****
SMARTPHONE DIRECTORY
****
You can now perform the following operations on this phonebook
1. Add a new contact
2. Remove an existing contact
3. Delete all contacts
4. Search for a contact
5. Display all contacts
6. Exit phonebook
Please enter your choice:


.....
Hello dear user, welcome to our smartphone directory system
You may now proceed to explore this directory
.....
Please enter initial number of contacts:

```
.............................................................
Hello dear user, welcome to our smartphone directory system
You may now proceed to explore this directory
.............................................................
Please enter initial number of contacts: 2
[]

Enter contact 1 details in the following order (ONLY):
NOTE: * indicates mandatory fields
.............................................................
Enter name*: sai
Enter number*: 7816784567
Enter e-mail address: sai@123
Enter date of birth(dd/mm/yy): 18/10/2005
Enter category(Family/Friends/Work/Others): family
```


****
SMARTPHONE DIRECTORY
****
You can now perform the following operations on this phonebook
1. Add a new contact
2. Remove an existing contact
3. Delete all contacts
4. Search for a contact
5. Display all contacts
6. Exit phonebook
Please enter your choice: 5
['sai', 7816784567, 'sai@123', '18/10/2005', 'family']
['ram', 8918255679, 'ram19@gmail.com', '18/10/2005', 'work']

CONCLUSION

The Contact Management System Project in Python is a easy to understand graphical user interface based mission which may be very comprehensible and utilizes.

Speaking about the system, it consists of all the referred to as for features that consist of adding the contact, viewing, doing away with and additionally upgrading contact lists.

While including the call of a person, he/she has to provide given name, middle name, last name, gender, age, home address, religion, nationality cope with and make contact with information

REFERENCE

- https://pyseek.com/2022/03/contact-management-system-project-in-python/
- https://www.geeksforgeeks.org/implementing-a-contacts-directory-in-python/
- https://copyassignment.com/contact-management-system-project-in-python/
- Data Structures Through Python Lab Manual