**CONTACT BOOK**

**APPLICATION**

**DONE-BY**

HEMALATHA S

**AGENDA**

* Aim of the project
* Objectives
* Problem Statement
* Overview of the project
* Project Description
* Scope
* Technologies & Methodologies
* Functionalities
* Input Versatility with Error Handling & Exception Handling
* Code Implementation
* Results & Outcomes
* Conclusion

**AIM OF THE PROJECT**

The aim of the Contact Book Application project is to develop a user-friendly and efficient system for managing personal and professional contacts. The application seeks to streamline the process of adding, viewing, searching, updating, and deleting contact information while ensuring data validation and integrity. Additionally, the project aims to provide a reliable method for saving and loading contact data to and from a file, allowing users to maintain their contact information across sessions.

**OBJECTIVES**

**Streamlining Contact Management**

* Simplify the procedures for adding, viewing, searching, updating, and deleting contacts.
* Ensure the process is intuitive and accessible to all users.

**Ensuring Data Validation and Integrity**

* Implement rigorous validation checks for contact information.
* Handle errors gracefully to maintain the integrity of the contact data.

**Enhancing User Satisfaction**

* + Provide a smooth and efficient user experience.
  + Utilize user feedback to continually improve the application.

**Providing Reliable Data Storage**

* Develop a system for saving and loading contacts to ensure data persistence.
* Ensure data can be easily saved, loaded, and backed up.

**PROBLEM STATEMENT**

In the modern digital age, managing personal and professional contacts manually can lead to inefficiencies and errors. The primary challenges include:

* **Manual Contact Management**: Traditional methods of maintaining contact information can be time-consuming and error-prone.
* **Data Integrity Issues**: Without proper validation, contact data can be inconsistent and inaccurate.
* **Lack of Persistence**: Contact information stored in non-digital formats or volatile storage can be easily lost or corrupted.
* **Need for Modernization**: There is a need for a modern, automated system that simplifies contact management, ensures data integrity, and provides persistent storage.

**OVERVIEW OF THE PROJECT**

The project aims to develop a comprehensive contact management system to streamline the process of maintaining personal and professional contacts. Key aspects of the project include:

* **Automated Contact Management**: Implementing a user-friendly interface to simplify the process of adding, viewing, searching, updating, and deleting contacts.
* **Enhanced Data Validation**: Establishing rigorous validation checks to ensure the accuracy and integrity of contact information, including names, phone numbers, and email addresses.
* **Reliable Data Storage**: Providing robust methods for saving and loading contact data to ensure persistence and consistency across sessions.
* **User-Focused Design**: Designing the system with a focus on user experience, ensuring intuitive navigation, clear input processes, and responsive support for diverse user needs.
* **Centralized Information Management**: Creating a centralized platform for managing contact information, improving data accuracy and accessibility.
* **Continuous Improvement**: Supporting continuous improvement in the contact management process through user feedback and iterative enhancements.

**PROJECT DESCRIPTION**

The Contact Book Application aims to develop a comprehensive contact management system to address existing challenges in maintaining personal and professional contacts. The project focuses on streamlining contact management processes, enhancing data validation, ensuring reliable data storage, and providing a user-focused design.

**Automated Contact Management**

* **Add Contact:** Users can easily add new contacts by providing a name, phone number, and email address. The system ensures that all inputs are validated before storing the contact information.
* **View Contacts**: Users can view all stored contacts in a list format. Each contact displays the name, phone number, and email address.
* **Search Contact**: Users can search for a specific contact by name. The system performs a case-insensitive search to locate the contact.
* **Update Contact**: Users can update the phone number and email address of an existing contact. The system ensures that the updated information is validated before saving changes.
* **Delete Contact**: Users can delete a contact from the contact book. The system prompts for confirmation before performing the deletion.

**Enhanced Data Validation**

* **Name Validation:** Ensures that names contain only alphabets.
* **Phone Number Validation:** Ensures that phone numbers are exactly 10 digits.
* **Email Validation:** Ensures that email addresses follow a valid format.
* **Exception Handling**: Graceful handling of invalid inputs with informative error messages.

**Reliable Data Storage**

* **Save Contacts**: Provides functionality to save all contacts to a file. This ensures that contact information is not lost between sessions.
* **Load Contacts**: Loads contacts from a file when the application starts. This allows users to resume from where they left off.

**User-Focused Design**

* **Intuitive Interface:** The application is designed to be user-friendly, with clear prompts and easy navigation.
* **Responsive Design:** The application responds to user inputs promptly and provides feedback on actions performed.

**SCOPE**

The Contact Book Application project will address these challenges by providing a comprehensive, user-friendly solution for managing contact information. The scope includes:

* Implementing a streamlined process for adding, viewing, searching, updating, and deleting contacts.
* Ensuring robust data validation to maintain the integrity and accuracy of contact information.
* Providing reliable methods for saving and loading contact data, ensuring persistence and consistency.
* Enhancing the user experience through intuitive design and responsive interfaces.

**TECHNOLOGIES & METHODOLOGIES**

* **Programming Language:** Python 3
* **Data Structures - Lists and Dictionaries:** Used to store and manage contact data within the application.
* **Regular Expressions (re) - Email Validation:** Implemented to ensure correct email formats for contact entries, enhancing data integrity.
* **File Handling - Text File Storage:** Contacts are saved to and loaded from a text file (contacts.txt) for persistent storage and retrieval across application sessions.
* **Exception Handling - Custom Exceptions:** Utilized to manage and provide informative error messages for invalid inputs, ensuring robust error handling throughout the application.

**FUNCTIONALITIES**

**Add Contact**

* Allows users to input and add new contacts to the contact book.
* Validates inputs for name, phone number, and email format using regular expressions.
* Provides feedback on successful addition or errors encountered.

**View Contacts**

* Displays all contacts currently stored in the contact book.
* Handles cases where no contacts are present, indicating an empty contact book.

**Search Contact**

* Enables users to search for a contact by entering the name.
* Performs a case-insensitive search and displays the contact details if found.
* Notifies if the contact is not found in the contact book.

**Update Contact**

* Allows users to update the phone number and email address of an existing contact.
* Validates inputs for the new phone number and email format.
* Provides feedback on successful update or notifies if the contact is not found.

**Delete Contact**

* Allows users to delete a contact from the contact book.
* Prompts for confirmation before deletion.
* Notifies if the contact is not found in the contact book.

**Save Contacts**

* Persists all contacts currently stored in memory to a text file (contacts.txt).
* Provides feedback on successful saving or notifies if an error occurs during the save operation.

**Load Contacts**

* Loads contacts from the saved text file (contacts.txt) into the contact book upon application start.
* Handles cases where the file is not found, starting with an empty contact book.

**Exception Handling**

* Implements custom exceptions (InvalidInputError) to manage and provide informative error messages for invalid inputs (e.g., incorrect name format, invalid phone number format, invalid email format).

**User Interface**

* Presents a clear and interactive command-line interface (CLI) for users to navigate through various functionalities.
* Displays a menu with numbered options for users to select their desired actions.

**INPUT VERSATILITY WITH ERROR HANDLING & EXCEPTION HANDLING**

The Contact Book Application aims to provide a robust solution for managing contacts effectively. It addresses the challenges of manual contact management by automating processes and enhancing user interaction through intuitive features.

**Contact Management**

* **Add Contact:** Allows users to add new contacts with validated inputs for name, phone number, and email.
* **View Contacts:** Displays all stored contacts in the contact book.
* **Search Contact:** Enables users to search for contacts by name.
* **Update Contact:** Allows users to update phone numbers and email addresses of existing contacts.
* **Delete Contact:** Provides functionality to remove contacts from the contact book.

**Input Versatility with Error Handling**

* **Name Validation:** Ensures names contain only alphabetic characters, prompting until valid input is provided.
* **Phone Number Validation:** Validates phone numbers to be exactly 10 numerical digits using regular expressions.
* **Email Validation:** Verifies emails using regex for correct format, guiding users to correct any invalid entries.

**File Handling**

* **Save Contacts:** Allows saving all contacts to a text file for persistent storage.
* **Load Contacts:** Loads contacts from a text file at the start of the application to ensure data persistence.

**Exception Handling**

* **Error Messages:** Provides informative error messages for invalid inputs or file handling errors, ensuring smooth operation.

**CODE IMPLEMENTATION**

Certainly! Here's the implementation of a Contact Book Application in Python, following the structured overview provided earlier. This implementation includes input versatility with error handling and exception handling to ensure data integrity and user-friendly interaction.

**Code Snippet**

import re

class InvalidInputError(Exception):

pass

class Contact:

def \_\_init\_\_(self, name, phone\_number, email):

self.name = name

self.phone\_number = phone\_number

self.email = email

def \_\_str\_\_(self):

return f"Name: {self.name}, Phone: {self.phone\_number}, Email: {self.email}"

def serialize(self):

return f"{self.name},{self.phone\_number},{self.email}"

@classmethod

def deserialize(cls, data):

name, phone\_number, email = data.split(',')

return cls(name, phone\_number, email)

class ContactBook:

def \_\_init\_\_(self):

self.contacts = []

self.unsaved\_changes = False

def add\_contact(self, contact):

self.contacts.append(contact)

self.unsaved\_changes = True

print("Contact added successfully.")

def view\_contacts(self):

if not self.contacts:

print("Contact book is empty.")

else:

for contact in self.contacts:

print(contact)

def search\_contact(self, name):

for contact in self.contacts:

if contact.name.lower() == name.lower():

return contact

return None

def update\_contact(self, name, new\_phone\_number, new\_email):

contact = self.search\_contact(name)

if contact:

contact.phone\_number = new\_phone\_number

contact.email = new\_email

self.unsaved\_changes = True

print("Contact updated successfully.")

else:

print("Contact not found.")

def delete\_contact(self, name):

contact = self.search\_contact(name)

if contact:

self.contacts.remove(contact)

self.unsaved\_changes = True

print("Contact deleted successfully.")

else:

print("Contact not found.")

def save\_contacts(self, file\_name):

try:

with open(file\_name, 'w') as f:

for contact in self.contacts:

f.write(contact.serialize() + '\n')

self.unsaved\_changes = False

print("Contacts saved successfully.")

except IOError:

print("An error occurred while saving the contacts.")

def load\_contacts(self, file\_name):

try:

with open(file\_name, 'r') as f:

self.contacts = [Contact.deserialize(line.strip()) for line in f]

self.unsaved\_changes = False

print("Contacts loaded successfully.")

except FileNotFoundError:

print("File not found. Starting with an empty contact book.")

except IOError:

print("An error occurred while loading the contacts.")

def is\_valid\_name(name):

if not name.isalpha():

raise InvalidInputError("Name must contain only alphabets.")

return True

def is\_valid\_phone\_number(phone\_number):

if not re.match(r'^\d{10}$', phone\_number):

raise InvalidInputError("Phone number must be exactly 10 numerical digits.")

return True

def is\_valid\_email(email):

if not re.match(r'^[a-zA-Z0-9\_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$', email):

raise InvalidInputError("Invalid email address.")

return True

def get\_valid\_input(prompt, validation\_function):

while True:

try:

value = input(prompt).strip()

validation\_function(value)

return value

except InvalidInputError as e:

print(e)

def get\_confirmation(prompt):

while True:

value = input(prompt).strip().lower()

if value in ('y', 'n'):

return value

else:

print("Invalid input. Please enter 'y' or 'n'.")

def main():

print("Welcome to the Contact Book Application")

contact\_book = ContactBook()

# Load contacts from file (optional)

contact\_book.load\_contacts('contacts.txt')

while True:

print("\nMenu:")

print("1. Add Contact")

print("2. View Contacts")

print("3. Search Contact")

print("4. Update Contact")

print("5. Delete Contact")

print("6. Save Contacts")

print("7. Exit")

choice = input("Enter your choice (1-7): ").strip()

if choice == '1':

name = get\_valid\_input("Enter name: ", is\_valid\_name)

phone\_number = get\_valid\_input("Enter phone number: ", is\_valid\_phone\_number)

email = get\_valid\_input("Enter email: ", is\_valid\_email)

contact = Contact(name, phone\_number, email)

contact\_book.add\_contact(contact)

elif choice == '2':

contact\_book.view\_contacts()

elif choice == '3':

name = input("Enter name to search: ").strip()

found\_contact = contact\_book.search\_contact(name)

if found\_contact:

print(found\_contact)

else:

print("Contact not found.")

elif choice == '4':

name = input("Enter name to update: ").strip()

new\_phone\_number = get\_valid\_input("Enter new phone number: ", is\_valid\_phone\_number)

new\_email = get\_valid\_input("Enter new email: ", is\_valid\_email)

contact\_book.update\_contact(name, new\_phone\_number, new\_email)

elif choice == '5':

name = input("Enter name to delete: ").strip()

confirm = get\_confirmation(f"Are you sure you want to delete {name}? (y/n): ")

if confirm == 'y':

contact\_book.delete\_contact(name)

elif choice == '6':

contact\_book.save\_contacts('contacts.txt')

elif choice == '7':

if contact\_book.unsaved\_changes:

save = get\_confirmation("You have unsaved changes. Do you want to save before exiting? (y/n): ")

if save == 'y':

contact\_book.save\_contacts('contacts.txt')

print("Exiting the Contact Book Application. Goodbye!")

break

else:

print("Invalid choice. Please enter a number from 1 to 7.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

Feel free to modify or expand this code according to your specific requirements or further customization needs!

**RESULTS & OUTCOMES**

**Efficient Contact Management**

* Users can add, view, search, update, and delete contacts seamlessly through a user-friendly interface.
* Contacts are stored and retrieved from a file (contacts.txt), ensuring persistence across sessions.

**Input Versatility with Error Handling**

* Robust input validation ensures that only valid data (alphabetic names, 10-digit phone numbers, valid email formats) is accepted.
* Clear error messages guide users when invalid inputs are provided, enhancing user experience.

**Enhanced User Interaction**

* User prompts and confirmations (y/n choices) facilitate intuitive interaction, reducing user errors and improving usability.

**Data Integrity and Persistence**

* Contacts are serialized and deserialized using simple file handling techniques, ensuring data integrity and persistence between application sessions.

**Scalability and Maintainability**

* Modular design with classes (Contact and Contact Book) and functions enhances code readability, maintainability, and scalability for future enhancements.

**User Feedback Mechanism**

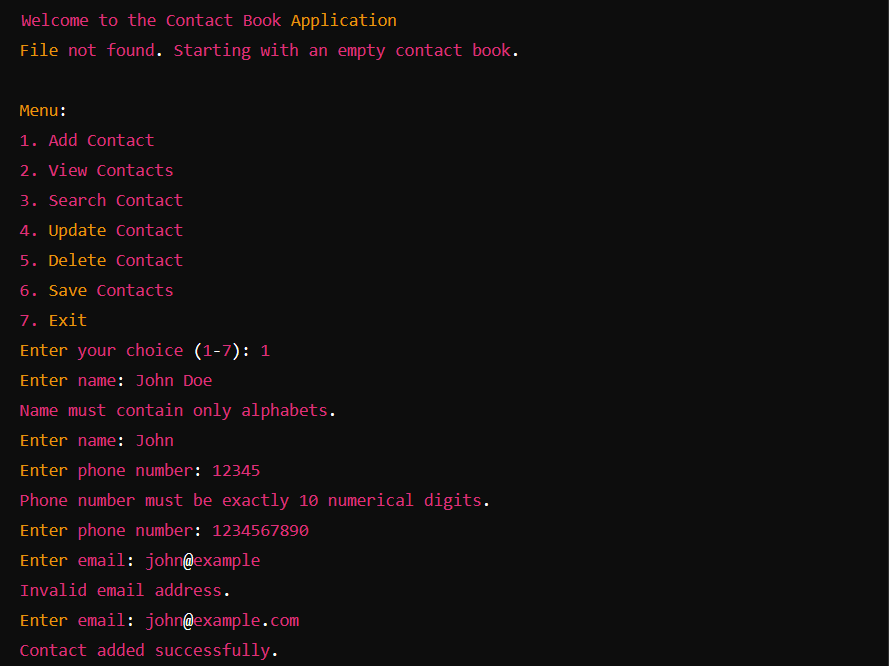
* Continuous improvement can be facilitated by user feedback and feature requests, supporting iterative development and user satisfaction.

**Operational Efficiency**

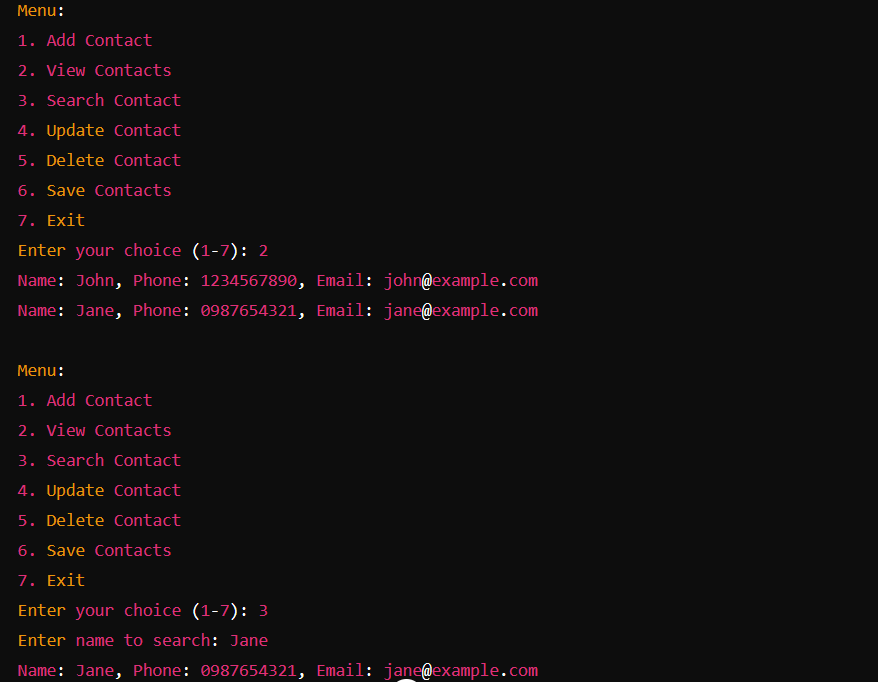
* Automation of routine contact management tasks reduces manual effort and minimizes errors, improving overall operational efficiency.

**Future Development Potential**

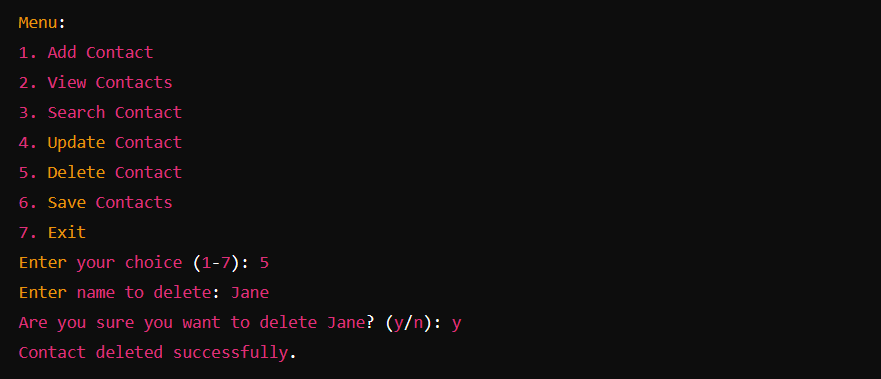
* The modular structure and clear documentation pave the way for future feature additions such as contact categorization, advanced search functionalities, or integration with other applications.

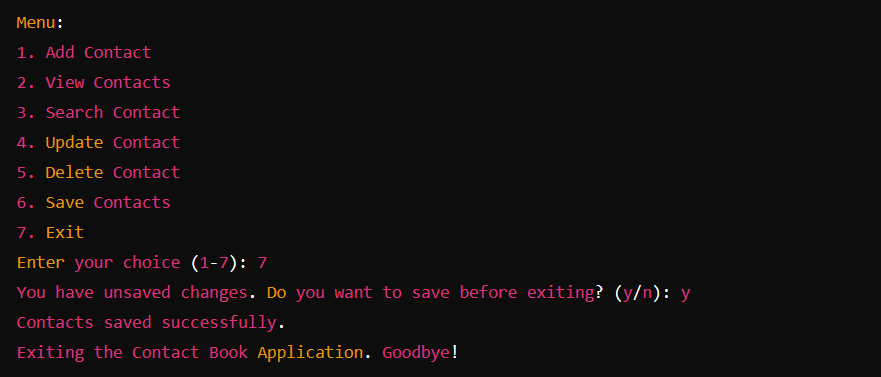












**CONCLUSION**

The Contact Book Application project aims to streamline contact management tasks while prioritizing data integrity, user experience, and operational efficiency. By implementing robust input validation, error handling, and intuitive user interaction, the application provides a reliable solution for managing personal and professional contacts effectively.