Address Book in Python

School of Computer Science and Engineering

GALGOTIAS UNIVERSITY

(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

Supervision of: Balbimdar Kaur

Submitted by: Ritik Ranjan (22SCSE1011036) & Shambhavi Rani

(22SCSE1011027)

Author Note

This project was developed by Ritik Ranjan and Shambhavi Rani, both B.Tech CSE students at Galgotias University, under the guidance of faculty member Balbimdar Kaur. The project was undertaken as part of a cooperative (Coop) education experience.

The AddressBook application showcases our skills in:

1. **Web Development**: Using Flask to create a functional and user-friendly web application.
2. **Data Modeling**: Designing robust data models to represent addresses and address books.
3. **Serialization**: Implementing methods for converting data between different formats for storage and retrieval.
4. **Form Handling**: Utilizing Flask-WTF for managing web forms and user inputs.
5. **Code Organization**: Structuring the project into modular components to enhance readability and maintainability.

This project is a testament to our commitment to continuous learning and excellence in software development. The AddressBook application not only fulfills an academic requirement but also serves as a practical tool that can be extended and improved for real-world use. Special thanks are extended to Balbimdar Kaur for her invaluable support and guidance throughout the project.

Acknowledgement

I extend my sincere gratitude to all those who contributed to the development of this Address Book Web Application. Special thanks to our faculty advisor, Balbimdar Kaur, for her expert guidance, continuous support, and insightful feedback, which were instrumental in shaping the project. I would like to acknowledge Galgotias University for providing the necessary resources and a conducive environment for research and development. The institution's commitment to excellence has been a driving force in the successful realization of the application.

Heartfelt appreciation goes to my partner, Shambhavi Rani, for her collaboration and dedication. The collective effort of our team played a crucial role in overcoming challenges and achieving the project's objectives. I want to express my thanks to the open-source community, particularly the developers behind the libraries and tools we used. Their contributions provided a solid foundation and significantly contributed to the project's success.

Abstract

This report delves into the structure, functionality, and organization of the AddressBook application codebase. The AddressBook application, designed using Python and Flask, serves as a digital address book, enabling users to store, manage, and interact with contact information. Key components of the codebase include data models for addresses, forms for user input, web server configurations for handling requests, and templates for rendering HTML content.

The Address class encapsulates the details of a physical address, providing methods for serialization, comparison, and modification. The AddressBook class maintains a collection of Address instances, supporting addition, removal, and iteration over stored addresses. Form handling is facilitated through the AddressForm class, which leverages Flask-WTF for validation and submission of address data. The Flask-based web server manages routes for displaying the address book, adding new addresses, and deleting existing ones.

Keywords: Flask, Serialization, Web server integration

Table of content

Chapter 1: Introduction

Objective

The main objectives of the AddressBook Web Application project are:

1. **Developing a User-Friendly Platform**: Create an intuitive and easy-to-use web interface for managing contact information, catering to users with varying levels of technical expertise.
2. **Efficient Address Management**: Enable users to store, edit, and delete addresses seamlessly, providing a streamlined solution for organizing contact details.
3. **Form Validation and Data Integrity**: Implement robust form validation mechanisms to ensure data integrity and accuracy, preventing errors and inconsistencies in address entries.
4. **Serialization and Storage**: Integrate serialization methods to facilitate the storage and retrieval of address data in various formats, enhancing flexibility and compatibility with different systems.
5. **Scalability and Maintenance**: Design the application with scalability and maintainability in mind, allowing for future enhancements and updates to meet evolving user needs.

Background Information

The background information for the AddressBook Web Application project revolves around the increasing reliance on digital solutions for managing contact information in both personal and professional settings. Traditional methods of storing addresses, such as physical address books or scattered digital files, often lack efficiency and accessibility. With the proliferation of digital communication and the need for centralized and organized contact databases, there is a growing demand for web-based address management solutions.

Moreover, advancements in web development technologies, particularly frameworks like Flask, have made it easier to develop sophisticated web applications with minimal overhead. These technologies enable developers to create user-friendly interfaces and implement robust backend functionalities, making web-based address management applications feasible and practical.

In this context, the AddressBook project seeks to leverage these advancements to develop a modern, intuitive, and efficient web application for managing contact information. By providing a centralized platform for storing, editing, and accessing addresses, the project aims to address the limitations of traditional address management methods and offer users a seamless and convenient solution for organizing their contact details.

Methodology

The methodology employed in developing the AddressBook Web Application involves several key steps:

1. **Requirement Analysis**: Conduct a thorough analysis of the requirements and user needs for the application, including features, functionalities, and user interface preferences.
2. **Design Planning**: Create a comprehensive design plan outlining the application architecture, database schema, user interface wireframes, and technology stack selection.
3. **Development**: Implement the design plan using Flask, a Python web framework, for backend development, and HTML, CSS, and JavaScript for frontend development. Develop features such as address storage, editing, deletion, form validation, and serialization.
4. **Testing**: Perform rigorous testing of the application to ensure functionality, usability, and security. Conduct unit tests, integration tests, and user acceptance tests to identify and fix any bugs or issues.
5. **Deployment**: Deploy the application on a web server, ensuring scalability, reliability, and performance. Configure server settings, domain mapping, and security measures to protect user data and ensure smooth operation.
6. **Documentation**: Prepare comprehensive documentation covering various aspects of the application, including installation instructions, usage guidelines, API documentation, and troubleshooting steps.

7. **Feedback and Iteration**: Gather feedback from users and stakeholders to identify areas for improvement. Iterate on the application based on feedback to enhance usability, performance, and user satisfaction.

By following this methodology, the AddressBook Web Application project aims to deliver a high-quality, user-friendly, and efficient solution for managing contact information, meeting the needs of both individual users and organizations.

Chapter 2: Code

File Structure

File structure is as follows:

```
AddressBook
├── main.py
├── README.md
├── requirements.txt
└── src
    ├── address
    │   ├── address_book.json
    │   ├── address.py
    │   ├── book.py
    │   └── __init__.py
    ├── __init__.py
    └── webserver
        ├── forms.py
        ├── __init__.py
        └── templates
            ├── add.html
            └── index.html
```

- The AddressBook application comprises several modules, including `address`, `book`, `webserver`, and `templates`.
- The `address` module defines the `Address` class for representing individual addresses, along with related data structures and methods.
- The `book` module contains the `AddressBook` class for managing collections of addresses, providing functionalities for adding, removing, and retrieving addresses.
- The `webserver` module implements the web application using Flask, defining routes, forms, and templates for user interaction.

- The templates directory contains HTML templates for rendering web pages, such as the index and add address forms.

Address class

Break down the Address class code by code:

1. Import Statements

```
from __future__ import annotations

import doctest
from typing import Any, TypedDict
```

The __future__ import is used to enable forward references of types in type hints. doctest module is imported for running doctests. Any and TypedDict are imported from the typing module for type hinting.

2. Definition of _MissingSentinel class:

```
class _MissingSentinel:
    """
    A sentinel class representing a missing value.
    ...
    """
    __slots__ = ()

    def __eq__(self, other) -> bool:
        return False

    def __bool__(self) -> bool:
        return False

    def __hash__(self) -> int:
        return 0

    def __repr__(self):
        return "..."
```

`_MissingSentinel` is a sentinel class used to indicate a missing value in certain contexts. It overrides the equality, boolean, and hash methods to always return `False` or `0`.

3. Definition of `Address` class:

```
class Address:
     def __init__(
     self,
     *,
     recipient_name: str,
     organization_name: str | None = None,
     building_number: str | None = None,
     street_name: str | None = None,
     apartment_number: str | None = None,
     city: str | None = None,
     state: str | None = None,
     postal_code: str | None = None,
     **kwargs,
     ) -> None:
     """Address class to represent a physical address
     ...
     """
     self.recipient_name: str = recipient_name
     self.organization_name = organization_name
```

```
        self.building_number = building_number
        self.street_name = street_name
        self.apartment_number = apartment_number
        self.city = city
        self.state = state
        self.postal_code = postal_code
```

The `Address` class represents a physical address with attributes like recipient name, organization name, building number, etc. It initializes the address attributes using the provided parameters. It also accepts additional keyword arguments (`**kwargs`) which are not used in the constructor.

`AddressBook` class

1. Import Statements

```
from __future__ import annotations
import logging
from src.address import Address, AddressDict
```

The `__future__` import is used to enable forward references of types in type hints. `logging` module is imported for logging messages. `Address` and `AddressDict` are imported from the `address` module for using the `Address` class and its TypedDict.

2. Class Definition:

```
class AddressBook:
    def __init__(self) -> None:
        """
        Initializes an empty AddressBook object.
        """
        self.__book_holder_name: str = MISSING
        self.__book_holder_id: int = MISSING
        self.__addresses: set[Address] = set()
```

The AddressBook class represents a collection of addresses. It initializes private attributes __book_holder_name, __book_holder_id, and __addresses. __book_holder_name and __book_holder_id are initialized with the MISSING sentinel value, indicating that they are not set yet. __addresses is initialized as an empty set to store Address objects.

3. Methods for Adding and Removing Addresses:

```
def add_address(self, address: Address) -> None:
        self.__addresses.add(address)

def remove_address(self, address: Address) -> None:
        self.__addresses.remove(address)
```

add_address method adds the given Address object to the address book.

remove_address method removes the given Address object from the address book.

`webserver/` package

The `webserver` package consists of several Python files that facilitate the interaction between the user and the address book system through a web interface. Let's explore the contents of this package:

1. **`__init__.py`:**
   - This file initializes the `webserver` package.
   - It may be empty or contain initialization code if necessary.
2. **`forms.py`:**
   - This file contains the definition of Flask-WTF forms used in the web application.
   - It defines an `AddressForm` class representing the form for adding addresses.
   - The form fields correspond to the attributes of an address, such as recipient name, organization name, building number, etc.
   - Each field may have validation rules defined using validators from Flask-WTF.
3. **`templates/` Directory:**
   - This directory contains HTML templates used to render web pages.
   - Templates are written using Jinja2 templating language and contain placeholders for dynamic content.
4. **`add.html` (inside `templates/`):**
   - This HTML template is used for rendering the form to add a new address.
   - It includes form fields for entering address details and a submit button to add the address.

5. **`index.html` (inside `templates/`):**
   - This HTML template is used for rendering the main index page of the address book application.
   - It displays a list of existing addresses with options to view, edit, or delete each address.
   - It also includes a link/button to navigate to the form for adding a new address.
6. **`__init__.py` (inside `webserver/`):**
   - This file initializes the `webserver` module.
   - It may be empty or contain initialization code if necessary.
7. **`__main__.py` (inside `webserver/`):**
   - This file contains the main entry point for running the Flask web application.
   - It imports the Flask app object from the `app` module and runs the application.

Code (`src/webserver/__init__.py`):

```python
from __future__ import annotations

from flask import Flask, redirect, render_template, url_for

from src.address import Address, AddressBook, AddressDict
from src.webserver.forms import AddressForm

app = Flask(__name__)
app.secret_key = "secret_key"

book = AddressBook.from_file(r"src/address/address_book.json")


@app.route("/")
def index():
    return render_template("index.html", address_book=book)


@app.route("/add", methods=["GET", "POST"])
def add():
    form = AddressForm()
    if form.validate_on_submit():
        data: AddressDict = form.data
        book.add_address(Address(**data))
        book.to_file()

        return redirect(url_for("index"))

    return render_template("add.html", form=form)


@app.route("/delete/<address_id>", methods=["GET"])
def delete(address_id):
    for address in book:
    if address.id == address_id:
         book.remove_address(address)
         return redirect(url_for("index"))

    return redirect(url_for("index"))


if __name__ == "__main__":
    app.run()
```

Breakdown of what each part does:

1. **Imports:**
    - The `__future__` import is used to enable the use of annotations in type hints.
    - `Flask` is imported from the `flask` module to create the web application.

   - Other necessary imports include `redirect`, `render_template`, and `url_for` from Flask for routing, as well as custom classes and forms from the `src.address` and `src.webserver.forms` modules.
2. **Flask Application Setup:**
   - An instance of the Flask application is created with the name `app`.
   - The secret key for the Flask session is set to `"secret_key"`.
3. **AddressBook Initialization:**
   - An instance of `AddressBook` is created by calling the `from_file` method, which reads data from a JSON file located at `src/address/address_book.json`.
4. **Route Handlers:**
   - The `/` route is mapped to the `index` function, which renders the `index.html` template, passing the `address_book` object as a parameter.
   - The `/add` route is mapped to the `add` function, which handles both GET and POST requests. It renders the `add.html` template with an `AddressForm` instance for adding new addresses. Upon form submission, it validates the form data, adds a new address to the address book, and redirects to the index page.
   - The `/delete/<address_id>` route is mapped to the `delete` function, which removes an address from the address book based on the provided address ID. It then redirects to the index page.
5. **Main Execution:**

- If the script is executed directly (not imported as a module), the Flask application (`app`) is run using the `run` method.

This script sets up a Flask web application with routes for displaying an address book, adding new addresses, and deleting existing addresses. It utilizes templates for rendering HTML pages and interacts with the `AddressBook` class to manage address data.

Code (`src/webserver/forms.py`):

```
from __future__ import annotations

from flask_wtf import FlaskForm
from wtforms import StringField, SubmitField
from wtforms.validators import DataRequired


class AddressForm(FlaskForm):
     recipient_name = StringField("Recipient Name",
validators=[DataRequired()])
     organization_name = StringField("Organization Name")
     building_number = StringField("Building Number")
     apartment_number = StringField("Apartment Name")
     street_name = StringField("Street Name")
     city = StringField("City")
     state = StringField("State", validators=[DataRequired()])
     postal_code = StringField("Postal Code")

     submit = SubmitField("Submit")
```

This Python script defines a FlaskForm class called `AddressForm`, which is a form used for adding addresses in a Flask application. Here's a breakdown of its components:

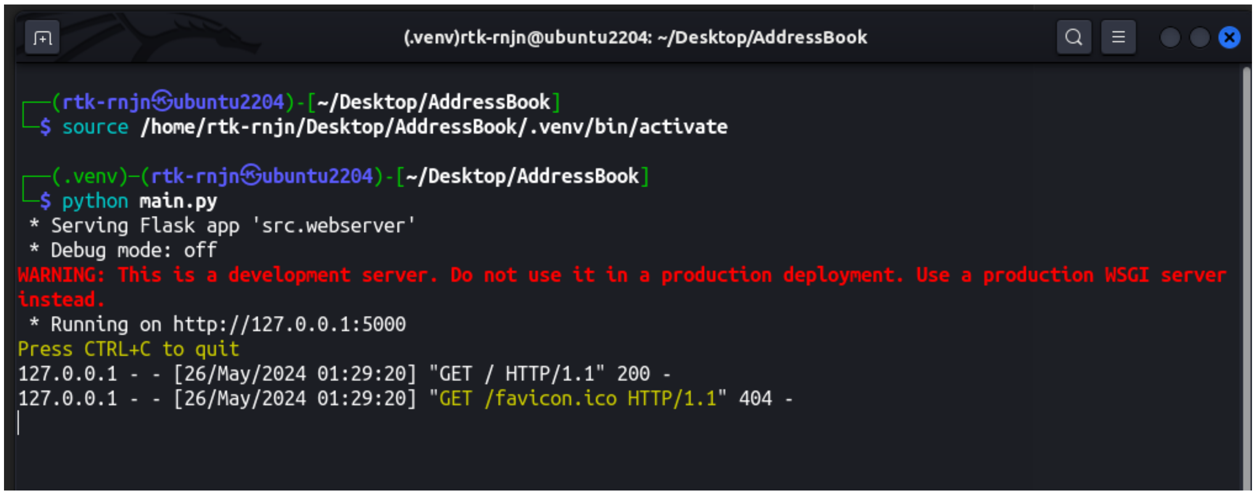
1. **Imports:**
   - `FlaskForm` is imported from `flask_wtf`, which is a Flask extension for handling web forms.
   - Various form field types (`StringField`, `SubmitField`) are imported from `wtforms`.
   - Validators (`DataRequired`) are imported from `wtforms.validators` to perform form validation.
2. **Form Fields:**
   - `recipient_name`: A StringField representing the recipient's name. It has a label "Recipient Name" and is required (`DataRequired`).
   - `organization_name`: A StringField representing the organization's name. It doesn't have any validators.
   - `building_number`: A StringField representing the building number. It doesn't have any validators.
   - `apartment_number`: A StringField representing the apartment number. It doesn't have any validators.
   - `street_name`: A StringField representing the street name. It doesn't have any validators.
   - `city`: A StringField representing the city. It doesn't have any validators.
   - `state`: A StringField representing the state. It is required (`DataRequired`).
   - `postal_code`: A StringField representing the postal code. It doesn't have any validators.

3. **Submit Button:**
    - `submit`: A SubmitField representing the submit button with the label "Submit". When clicked, this button submits the form data.

This form is used to gather information about an address, including recipient name, organization name, building number, apartment number, street name, city, state, and postal code. It ensures that the state field is always provided, while the other fields are optional.

Chapter 3: Result

Starting the Server:

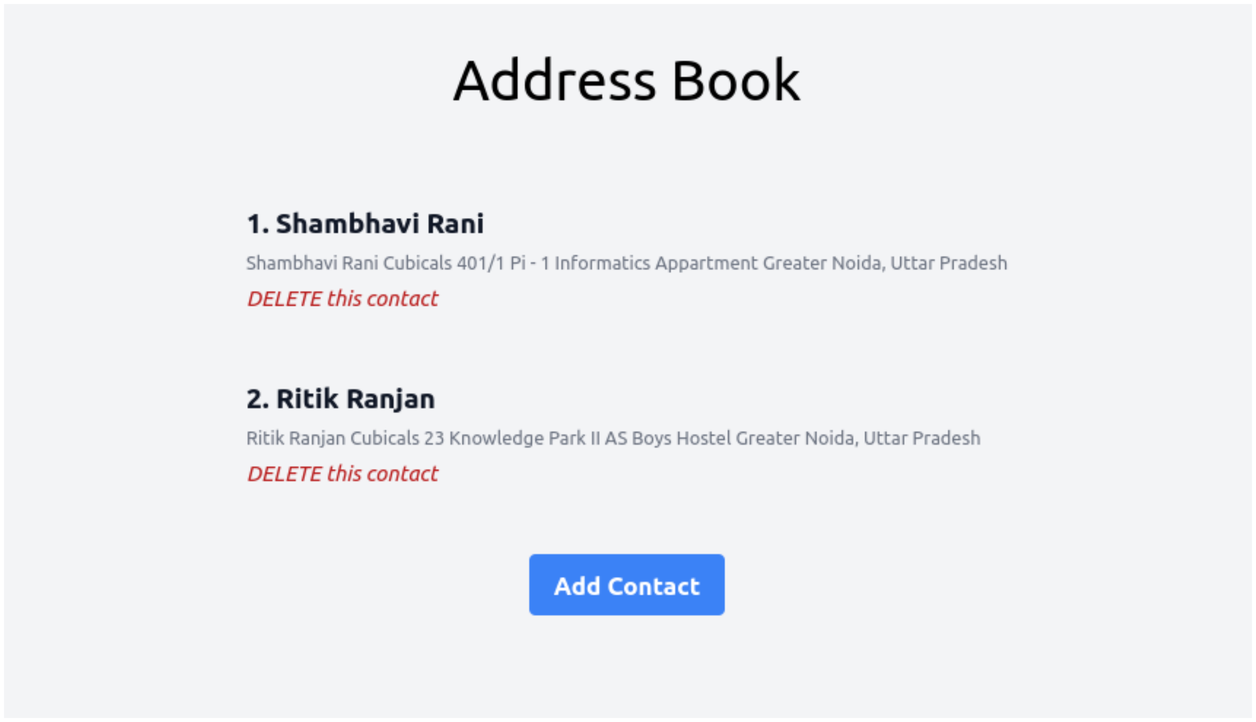


```
(.venv)rtk-rnjn@ubuntu2204: ~/Desktop/AddressBook

┌──(rtk-rnjn㉿ubuntu2204)-[~/Desktop/AddressBook]
└─$ source /home/rtk-rnjn/Desktop/AddressBook/.venv/bin/activate

┌──(.venv)─(rtk-rnjn㉿ubuntu2204)-[~/Desktop/AddressBook]
└─$ python main.py
 * Serving Flask app 'src.webserver'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server
instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [26/May/2024 01:29:20] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [26/May/2024 01:29:20] "GET /favicon.ico HTTP/1.1" 404 -
```

Server Running at `localhost`



Address Book

1. Shambhavi Rani

Shambhavi Rani Cubicals 401/1 Pi - 1 Informatics Appartment Greater Noida, Uttar Pradesh

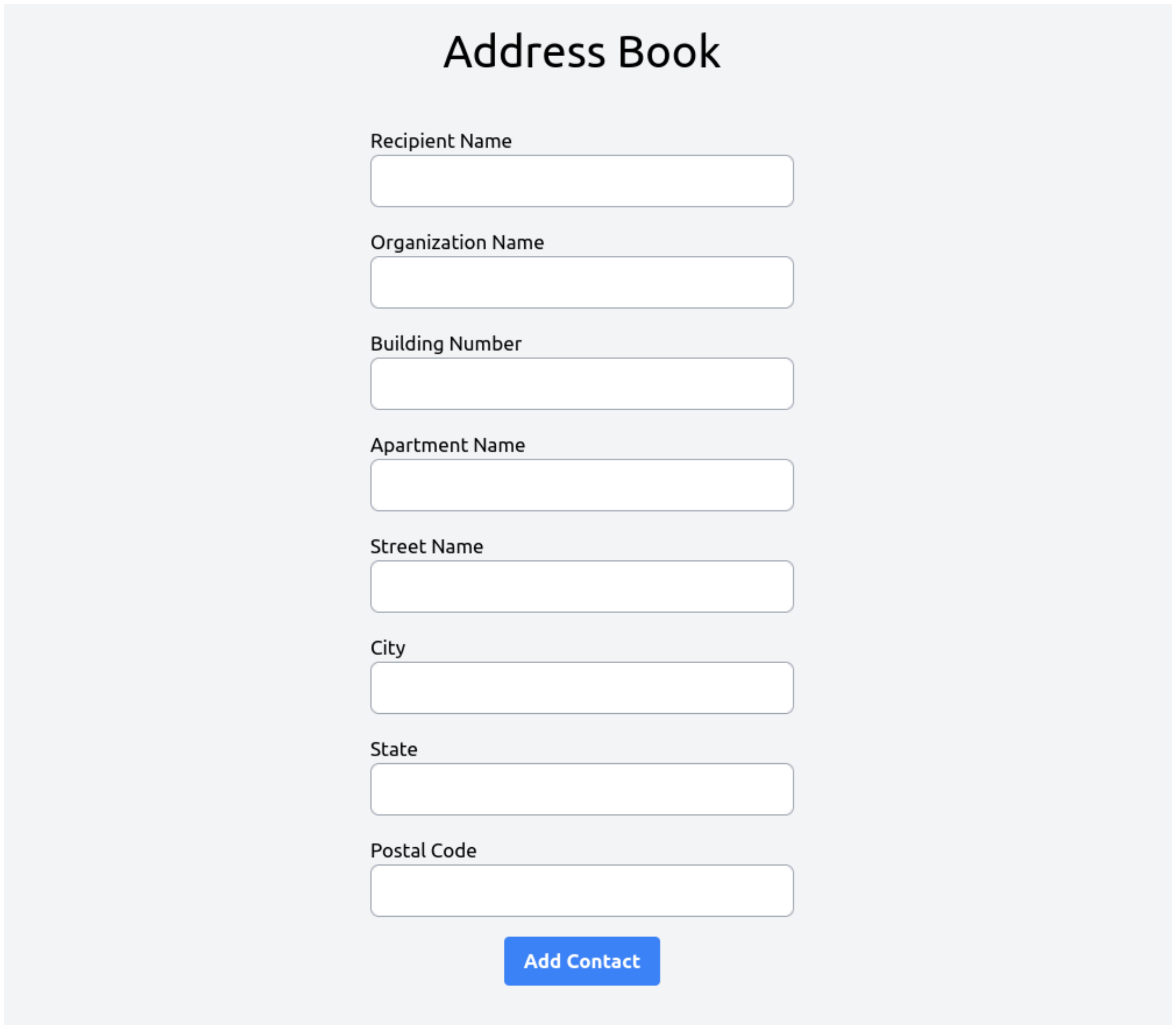
DELETE this contact

2. Ritik Ranjan

Ritik Ranjan Cubicals 23 Knowledge Park II AS Boys Hostel Greater Noida, Uttar Pradesh

DELETE this contact

Adding More contacts



1. **Project Objectives Recap:**
    - Briefly restate the main objectives of the Address Book project, emphasizing the goals set out in the introduction.
2. **Methodologies Overview:**

   - Summarize the methodologies employed to achieve the project objectives, including the development approach, technologies used, and implementation strategies.
3. **Key Findings and Results:**
   - Highlight the main findings and results obtained during the project implementation, focusing on the Address Book system's functionality, usability, and performance.
4. **Implications and Insights:**
   - Discuss the implications of the project findings and insights gained from the development process, including the impact on address book management systems, web application development, and user experience design.
5. **Achievements and Contributions:**
   - Outline the project's achievements and contributions to the field, emphasizing any innovative features, solutions to challenges, or advancements made.
6. **Limitations and Challenges:**
   - Acknowledge any limitations or challenges encountered during the project, such as technical constraints, time constraints, or scope limitations.
7. **Future Directions:**
   - Suggest potential areas for future research, development, or enhancements to the Address Book system, considering user feedback, emerging technologies, and evolving user needs.

Chapter 5: References

1. Python Documentation: https://docs.python.org/
2. Flask Documentation: https://flask.palletsprojects.com/
3. WTForms Documentation: https://wtforms.readthedocs.io/
4. Flask-WTF Documentation: https://flask-wtf.readthedocs.io/