Simple Chat / Messaging Simulator

Made by: Muddasir

CMS ID: 65069

Section: BS Software engineering-5th semester

Introduction

This project is a desktop-based Simple Chat / Messaging Simulator developed using Python and Tkinter. The primary goal is to demonstrate clean software architecture using multiple design patterns such as Factory, Builder, Decorator, Observer, and Singleton. The system simulates sending/receiving messages, styling messages, user notifications, and chat management.

Design Pattern Usage

Message Factory (Factory pattern)

The Factory Pattern is implemented in message_factory.py. The purpose is to centralize and simplify the creation of Message objects. The factory exposes a create_message() function that accepts text and sender type and returns a Message instance. This solves the problem of maintaining object creation logic inside the UI, making the system modular and easier to extend.

Chat Session Builder (builder pattern)

The Builder Pattern is implemented in builder.py. The ChatSessionBuilder class constructs a chat window step by step. It handles widget creation such as text areas and entry boxes. This pattern solves the problem of placing complex UI construction logic inside app.py. The builder creates clean separation between UI layout and application logic

Decorator Pattern (decorator pattern)

The Decorator Pattern appears in decorator.py. The MessageDecorator and StyledMessage classes modify Message objects by adding extra formatting attributes such as prefixes or styled text. This allows messages to be visually modified without changing the original Message class. The decorator wraps Message instances and adds style dynamically

Observer Pattern (observer pattern)

The Observer Pattern is implemented in observer.py. The NotificationObserver class listens for new incoming messages that are routed through the ChatEngine. When ChatEngine executes notify_observers(), it triggers all observers and updates the UI. This simulates real-time incoming message notification.

Chat Engine (Singleton pattern)

The Singleton Pattern is implemented in chat_engine.py. ChatEngine uses a class-level attribute _instance and a get_instance() method to ensure that only one ChatEngine object exists throughout the program. This singleton centralizes message routing and log management, preventing multiple conflicting engine instances from being created.

Implementation Details

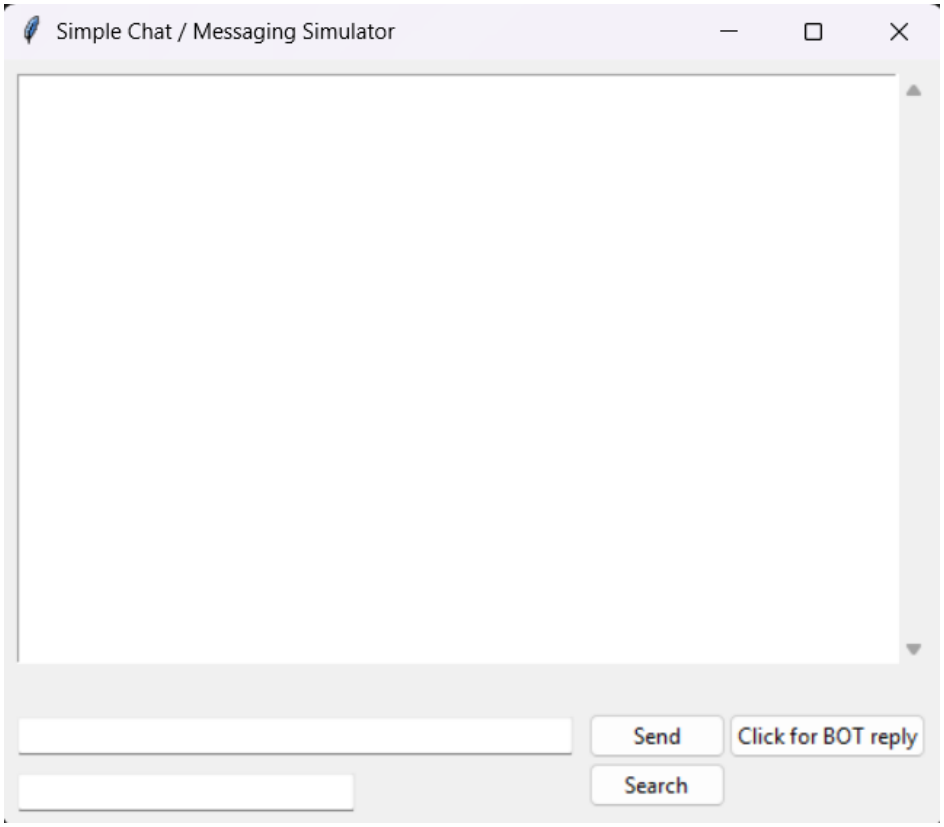
The project is implemented in Python using the Tkinter GUI framework. Object-oriented programming techniques and design patterns are used to ensure modularity. The codebase is structured with separate files for factory, builder, observer, decorator, and engine logic.

Technologies used:

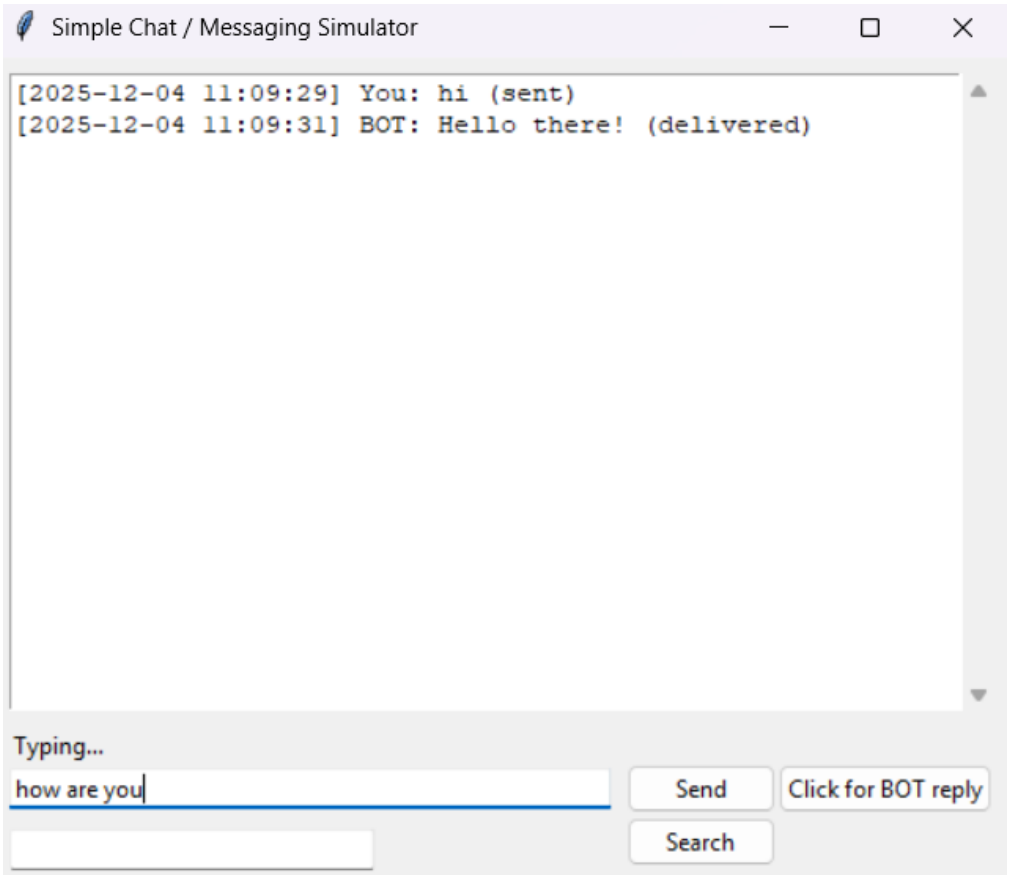
- Python 3.12
- Tkinter GUI in python
- Object-Oriented Design Patterns in python

Screen Shots/Outputs

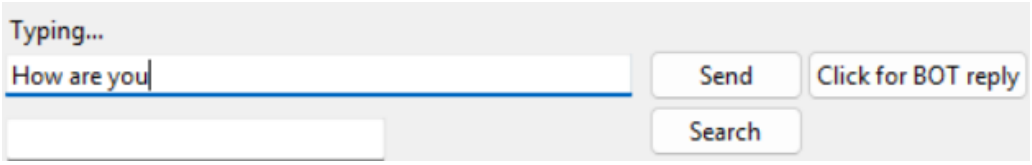
Main screen



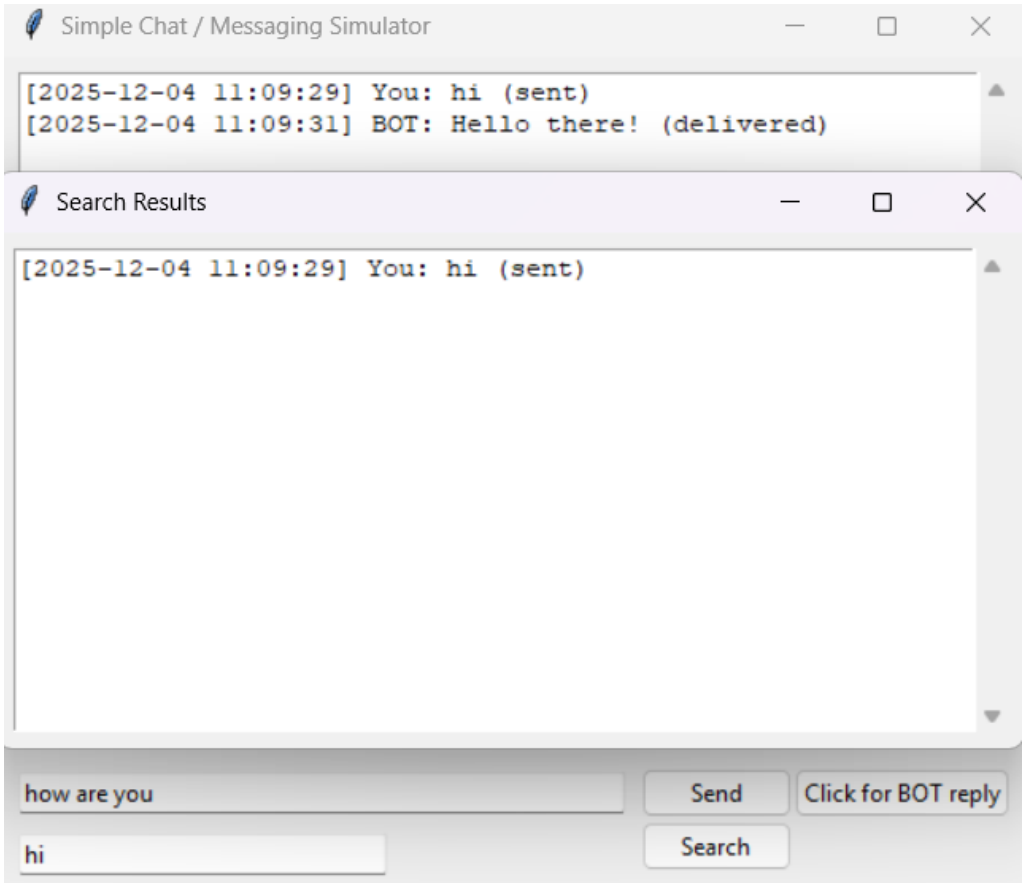
User messaging and bot replying



Typing indicator



Searching message



Conclusion

This project demonstrates practical implementation of core software engineering design patterns in a functional chat simulation application. The architecture ensures clean separation of concerns and scalability for future expansions such as multi-user support, enhanced UI themes, or persistence.