Abstract

The project chat application is already in existence of great many companies like apple's messages, google, Microsoft, Facebook, WhatsApp , Instagram etc. It's really needed because of various reasons. Mostly of increasing number of developers and their products.

This project mainly concerns of same user experience throughout the operating systems and platforms. This project can be deployed on windows, mac, linux, android, apple iOS.

Which basically is every platform people use today. This helps in improving time management by reducing the time to correct errors which occur in one and implement the same across other operating systems and platforms.

TABLE OF CONTENTS

Chapter 1 Introduction

This project will give user a unique experience chat with other people. We use socket programming to connect to server and chat with each other through the application. This project is developed in python and it's modules.

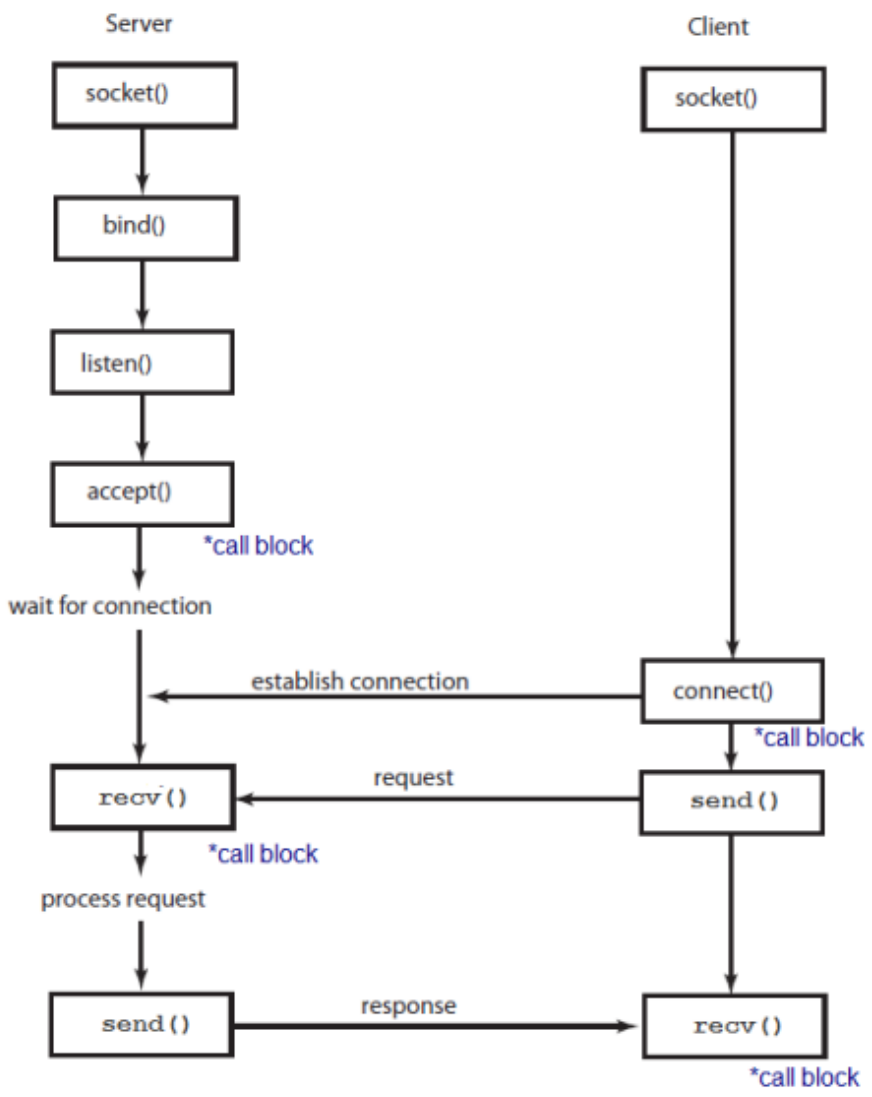
Sockets can be thought of as endpoints in a communication channel that is bi-directional, and establishes communication between a server and one or more clients. Here, we set up a socket on each end and allow a client to interact with other clients via the server. The socket on the server side associates itself with some hardware port on the server side. Any client that has a socket associated with the same port can communicate with the server socket.

A thread is sub process that runs a set of commands individually of any other thread. So, every time a user connects to the server, a separate thread is created for that user and communication from server to client takes place along individual threads based on socket objects created for the sake of identity of each client.

We will require two scripts to establish this chat room. One to keep the serving running, and another that every client should run in order to connect to the server.

The server side script will attempt to establish a socket and bind it to an IP address and port specified by the user (windows users might have to make an exception for the specified port number in their firewall settings, or can rather use a port that is already open). The script will then stay open and receive connection requests, and will append respective socket objects to a list to keep track of active connections. Every time a user connects,
a separate thread will be created for that user. In each thread, the server awaits a message, and sends that message to other users currently on the chat. If the server encounters an error while trying to receive a message from a particular thread, it will exit that thread.

Chapter 2 System design



Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.

They are the real backbones behind web browsing. In simpler terms there is a server and a client.

Socket programming is started by importing the socket library and making a simple socket.

```
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Here we made a socket instance and passed it two parameters. The first parameter is AF_INET and the second one is SOCK_STREAM. AF_INET refers to the address family ipv4. The SOCK_STREAM means connection oriented TCP protocol.

Now we can connect to a server using this socket.

Server :

A server has a bind() method which binds it to a specific ip and port so that it can listen to incoming requests on that ip and port.A server has a listen() method which puts the server into listen mode. This allows the server to listen to incoming connections. And last a server has an accept() and close() method. The accept method initiates a connection with the client and the close method closes the connection with the client.

- First of all we import socket which is necessary.
- Then we made a socket object and reserved a port on our pc.
- After that we binded our server to the specified port. Passing an empty string means that the server can listen to incoming connections from other computers as well. If we would have passed 127.0.0.1 then it would have listened to only those calls made within the local computer.
- After that we put the server into listen mode.5 here means that 5 connections are kept waiting if the server is busy and if a 6th socket trys to connect then the connection is refused.
- At last we make a while loop and start to accept all incoming connections and close those connections after a thank you message to all connected sockets.

Client :

Now we need something with which a server can interact. We could tenet to the server like this just to know that our server is working.

Chapter 3 Implementation

3.1 Description of tools Used

Python

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

The Python interpreter is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C). Python is also suitable as an extension language for customisable applications.

Reasons to Python

- Scalability and Flexibility
- Expressive language.
- Interpreted language
- Management Ease
- Available to all platforms
- Easy to install
- Easy to learn
- Supports Object oriented programming
- Free and open source
- Large standard library

3.2 Description of Development Environment

Kivy

Kivy is a free and open source Python library for developing mobile apps and other multitouch application software with a natural user interface (NUI). It is distributed under the terms of the MIT License, and can run on Android, iOS, Linux, OS X, and Windows.

Kivy is the main framework developed by the Kivy organisation,[2] alongside Python for Android,[3] Kivy iOS,[4] and several other libraries meant to be used on all platforms. In 2012, Kivy got a $5000 grant from the Python Software Foundation for porting it to Python 3.3.[5] Kivy also supports the Raspberry Pi which was funded through Bountysource.[6]

The framework contains all the elements for building an application such as:

- extensive input support for mouse, keyboard, TUIO, and OS-specific multitouch events,
- a graphic library using only OpenGL ES 2, and based on Vertex Buffer Object and shaders,
- a wide range of widgets that support multitouch,
- an intermediate language (Kv)[7] used to easily design custom widgets.

Kivy is the evolution of the PyMT project, and is recommended for new projects.[8]

Chapter -4 Methodology

Snapshots of the code

1. Main.py

```python
import kivy  # importing main package
import socket_client
from kivy.app import App  # required base class for your app.
from kivy.uix.label import Label  # uix element that will hold text
from kivy.uix.gridlayout import GridLayout  # one of many layout structures
from kivy.uix.textinput import TextInput  # allow for ...text input.
from kivy.uix.button import Button # to add a button
from kivy.uix.screenmanager import ScreenManager, Screen
from kivy.clock import Clock
from kivy.core.window import Window
from kivy.uix.scrollview import ScrollView
kivy.require("1.10.1")  # make sure people running py file have right version

# An actual app is likely to consist of many different
# "pages" or "screens." Inherit from GridLayout
class ConnectPage(GridLayout):…


# Simple information/error page
class InfoPage(GridLayout):…

class ChatPage(GridLayout):…

# Our simple app. NameApp  convention matters here. Kivy
# uses some magic here, so make sure you leave the App bit in there!
class EpicApp(App):…

# This class is an improved version of Label
# Kivy does not provide scrollable label, so we need to create one
class ScrollableLabel(ScrollView):…

# Error callback function, used by sockets client
# Updates info page with an error message, shows message and schedules exit in 10 seconds
# time.sleep() won't work here - will block Kivy and page with error message won't show up
def show_error(message):…

# Run the app.
if __name__ == "__main__":…
```

2. socket_server.py

```python
import socket
import select

HEADER_LENGTH = 10

IP = "127.0.0.1"
PORT = 1234

# Create a socket
# socket.AF_INET - address family, IPv4, some other possible are AF_INET6, AF_BLUETOOTH, AF_UNIX
# socket.SOCK_STREAM - TCP, conection-based, socket.SOCK_DGRAM - UDP, connectionless, datagrams, socket.SOCK_RAW - raw IP packets
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# SO_ - socket option
# SOL_ - socket option level
# Sets REUSEADDR (as a socket option) to 1 on socket
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

# Bind, so server informs operating system that it's going to use given IP and port
# For a server using 0.0.0.0 means to listen on all available interfaces, useful to connect locally to 127.0.0.1 and remotely to LAN i
server_socket.bind((IP, PORT))

# This makes server listen to new connections
server_socket.listen()

# List of sockets for select.select()
sockets_list = [server_socket]

# List of connected clients - socket as a key, user header and name as data
clients = {}

print(f'Listening for connections on {IP}:{PORT}...')

# Handles message receiving
def receive_message(client_socket):…

while True:…
```

3. socker_client.py

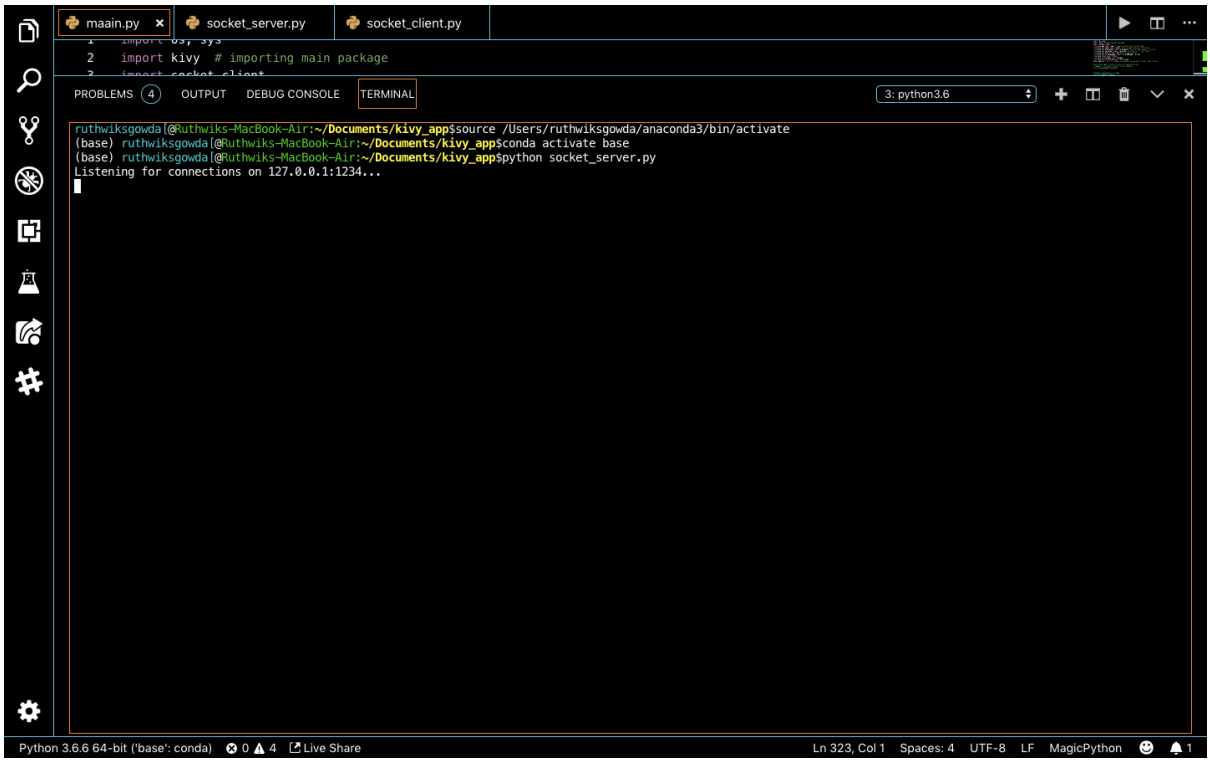
maain.py | socket_server.py | socket_client.py

```python
import socket
import errno
from threading import Thread

HEADER_LENGTH = 10
client_socket = None

# Connects to the server
def connect(ip, port, my_username, error_callback):...

# Sends a message to the server
def send(message):...

# Starts listening function in a thread
# incoming_message_callback - callback to be called when new message arrives
# error_callback - callback to be called on error
def start_listening(incoming_message_callback, error_callback):...

# Listens for incomming messages
def listen(incoming_message_callback, error_callback):...
```

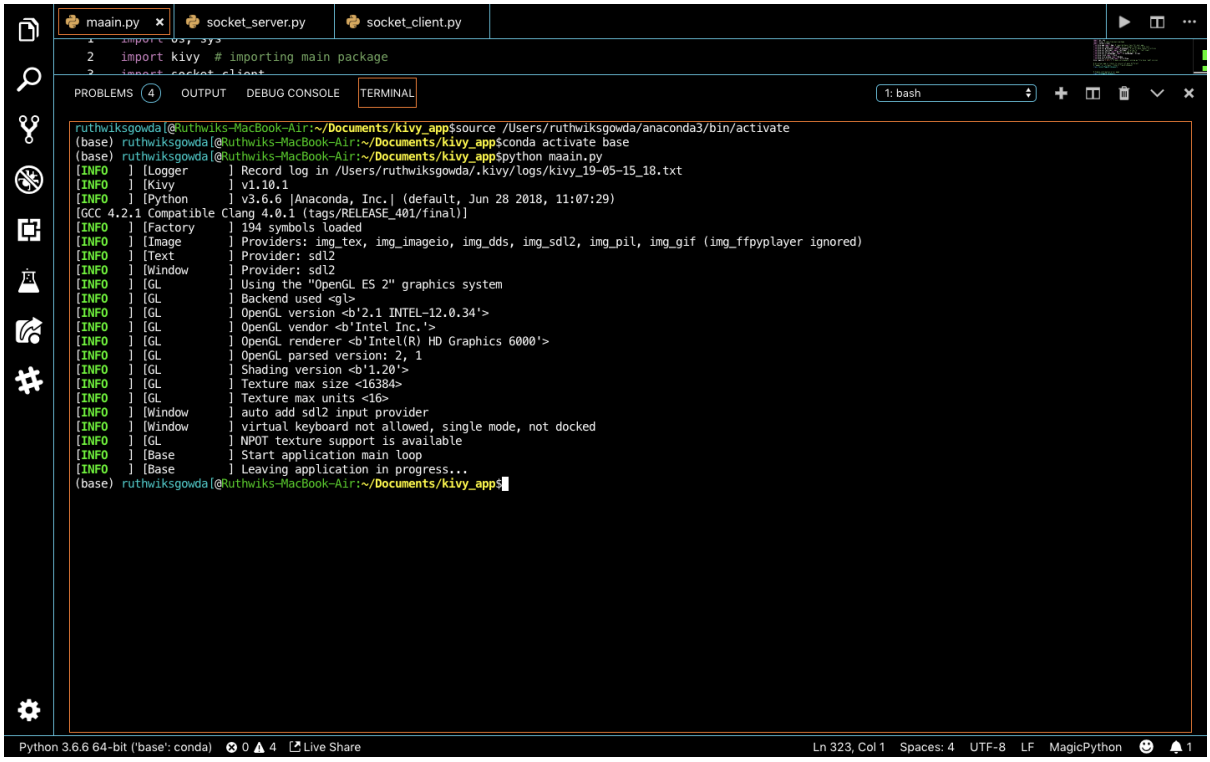
Python 3.6.6 64-bit ('base': conda) 0 4 Live Share Ln 9, Col 52 Spaces: 4 UTF-8 LF MagicPython 1

Chapter 5 Interpretation of Results

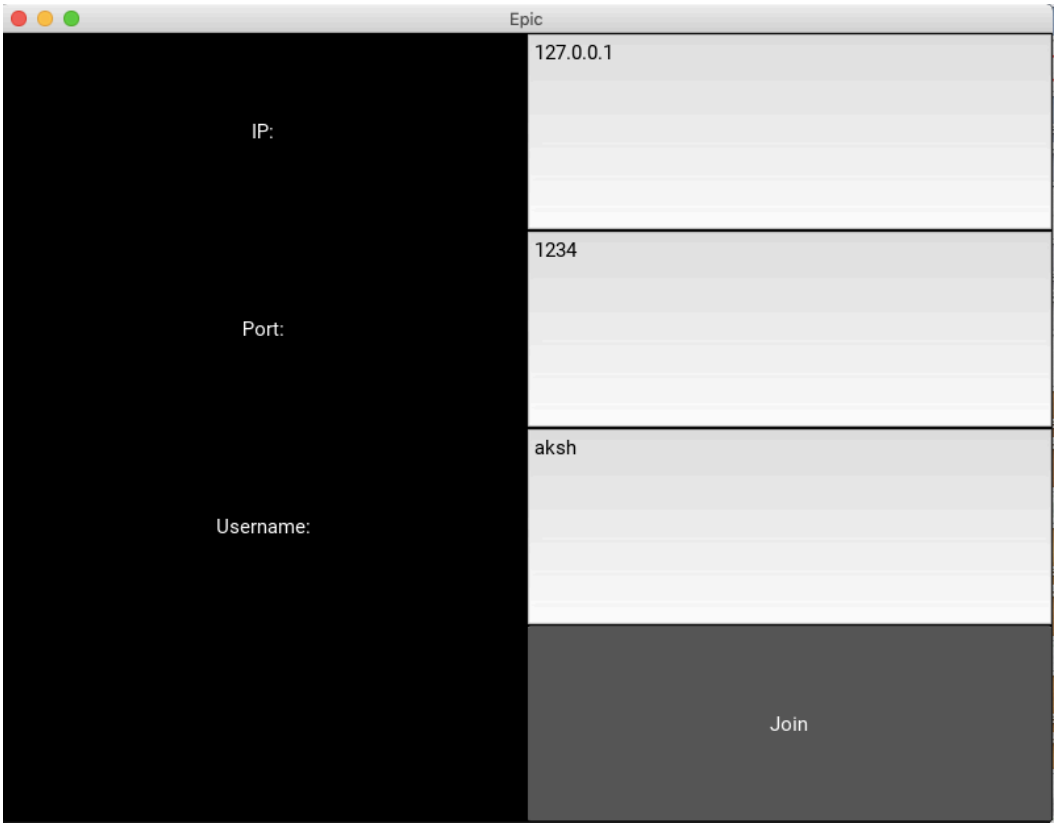
Starting the server



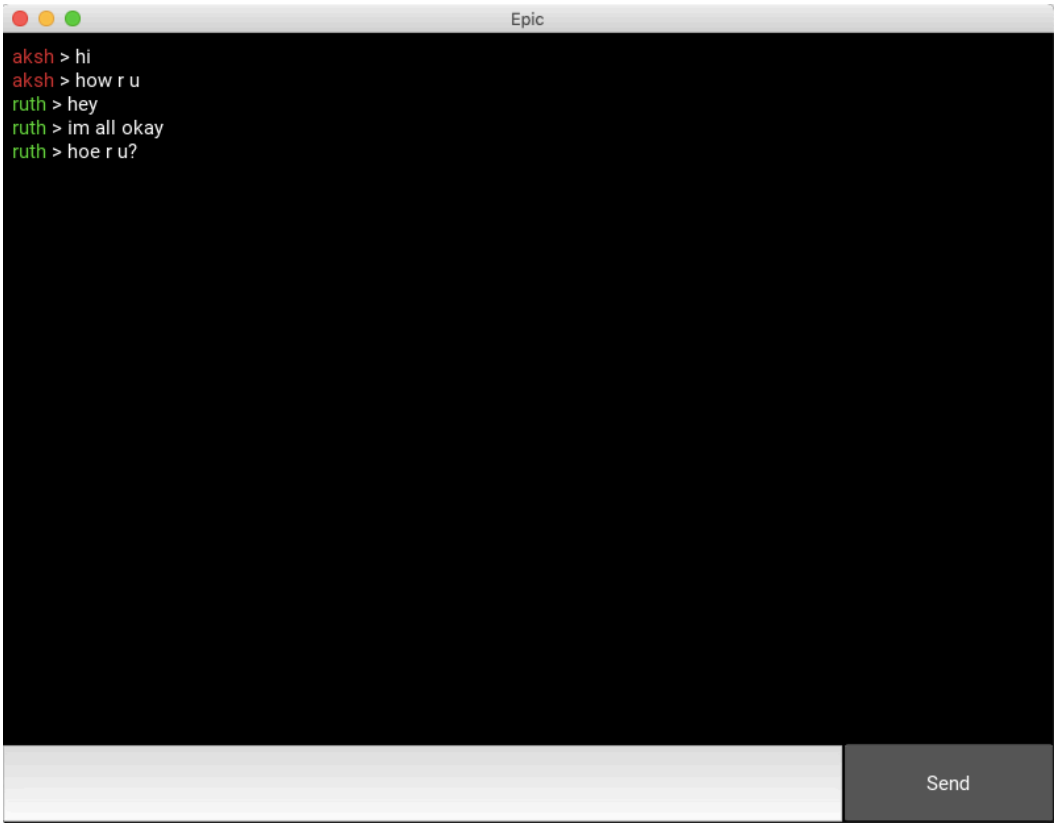
Running python maain.py



First page after running maain.py (info page)



Second page after clicking join button (chat page)



Conclusion

Once this project is completed it offers users the following functionalities:

Multiple login can be done to the same chatroom. With infinite text and infinite scrolling. This allows users to converse with ease as it shows the name go the user which was entered in the beginning.

This application can be deployed on any platforms which makes it easy to make changes to all the platforms at a time and as it is same to all platforms, there's only one UI experience which helps people understand the application even when they change their platforms or operating systems.

References

- StackOverflow: www.stackoverflow.com
- https://kivy.org/doc/stable/
- https://pythontips.com/2013/08/06/python-socket-network-programming/
- www.tutorailspoint.com
- https://www.geeksforgeeks.org
- www.youtube.com
- www.wikipedia.org