**Table of Content:**

1. INTRODUCTION
2. DEPENDENCIES AND TECHNOLOGY USED
3. ALGORITHM & TECHNICAL DESCRIPTION
4. SCREENSHOTS
5. RESULTS & INFERENCES
6. CONCLUSION

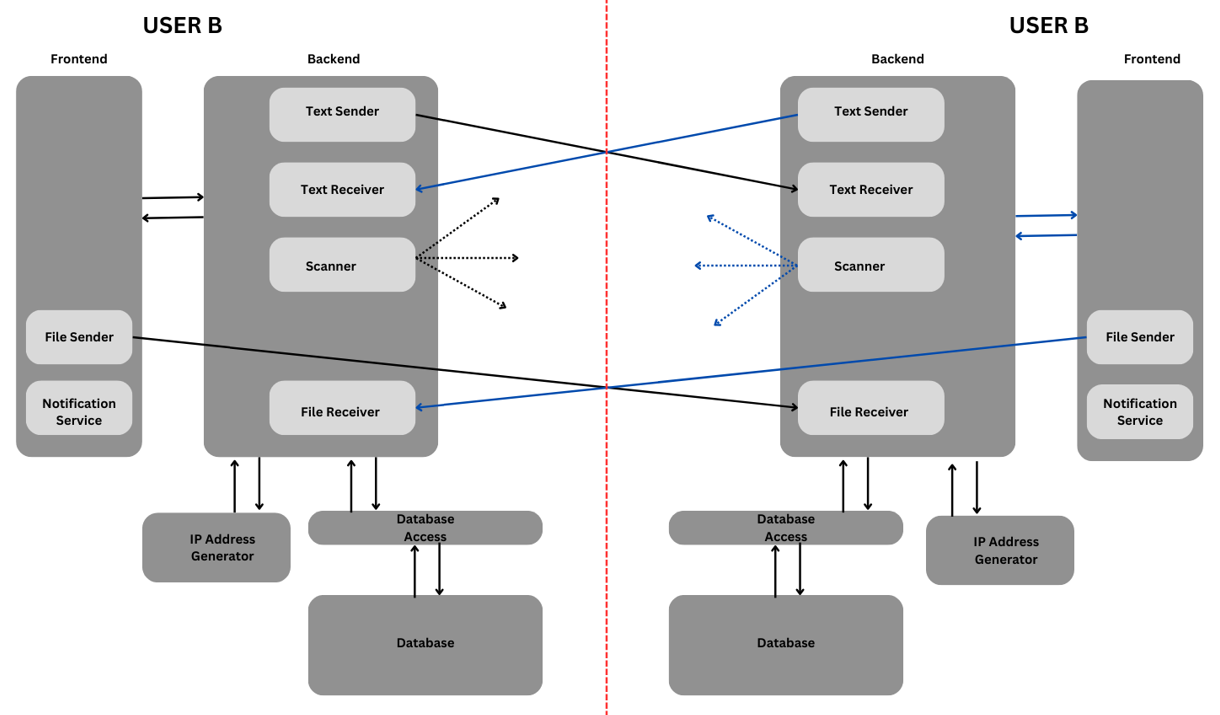
**INTRODUCTION:**

Most often, the central servers in modern communication networks can cause latency, bottlenecks, or breach of privacy. The objective of this project is to address the problems posed by a central server by introducing a P2P communication system. The application lets users send and receive messages and share files over devices connected to the local area network with each other. Here every user acts as a client and a server depending on the task. The system relies on simple HTTP GET and POST requests, JavaScript-based notification services, and multi-threaded scanning to find devices. By separating the front end, API handler, and processing units of the application, the system is both effective and flexible in design and use. This approach enables internal communication in termed networks devoid of changes while allowing other changes in the future.

**DEPENDENCIES AND TECHNOLOGY USED:**

Python : flask

**ALGORITHM & TECHNICAL DESCRIPTION:**



**Fig: Flow Chart of Application Design and Working**

The whole application is designed on the concept of a model in which each user (running application of user system) is both client as well as server depending upon the scenario. For example, during the sending of any test message the sender is a client and receiver is a server.

The whole application can be divided into three parts as follow: Frontend, API handler and Processor Program

**Frontend:**

This is a basic HTML and JavaScript based web page for user interaction. These are the function performed by it –

1. It receives input from user like text message to be sent, file to be send, receiver ID, display the messages and continuously check for any received file or messages.
2. After receiving any given input from user, it make a simple HTTP Post request to it server to upload the text input
3. **Notification Service** : - It is a function written in JavaScript which continuously make a get request to server and in response it may gets the ID and name of user which send either text or file. This function is written as asynchronous function having infinite while loop repeating after each 1 sec. The asynchronous nature of the function make it to run continuously in a separate thread parallel to the main thread helps in natural execution of rest of the program.
4. **File Sender** :- As the HTTP is no longer a protocol for only sending text, but using post method of HTTP not only longer text can be send but also various files irrespective whether it is a video file, image file, etc. File sender is also a function which send file to the receiver. When user select a file and hit the send button then its functionality starts. After checking basic details like valid file and ID, it fetch the current IP address of the given receiver ID from server, prepare a post request on received IP address with attached file, then make the post request on the receiver. Here it should be noted that, file is not send on the server first to dispatch it to receiver, it is directly send from frontend itself

**API Handler:**

It is a part of server/backend. Its sole purpose it to receive API request (in the form of either http GET or POST).

It has two category of API routes :-

(a). API for interaction with frontend. Like for receiving inputs, sending messages and other information

(b). API for handling core functionality like sending & receiving data and scanning.

There are various HTTP routes for different purpose, routes for implementing core functionality is shown in block diagram and explained below-

1. **Text Sender: -** This route receive the message and ID of the receiver (for sending message) from frontend. Interact with database using a program called Database\_Access to get the current IP address of the receiver ID. Prepare API which include text message and sender ID, then make a get request to the API. (<http://receiverIP:port/receiveText?id=senderID&message=MESSAGE>)
2. **Text Receiver:-** This route listens for any http get request on “/receiveText” route. On receiving such request it extract sender ID and message from the API. Interact with Database to update the message on sender ID. It also extract the IP address from where this API is send and update this current IP address of the sender in it database.
3. **Scanner:**- On receiving any request for scanning from frontend it does following-
   * 1. It generate a list of all possible IP address on the current local network using subnet mask and the IP address of the device with the help of Processor Program called IP\_Address\_Generator
     2. Start making get request by attaching it ID to each IP. With the help of multithreading these requests are send parallelly in each separate thread making the scanning process much faster.
     3. Each request has a timeout of 2-3 seconds. If before the timeout the response is not received for given IP address then it is assumed that that device is unavailable.
     4. For the purpose of making response, “/respond” route is defined. When any device receive request on this route then it not only respond to the sender but also capture the sender IP address and ID(as scanning device attach it ID in get request) and update in its databse.
4. **Files Receiver:-** Unlike Text Receiver, it listens for POST request having a file attached on the request. On receiving such request it extract file and ID. Save the file by naming filename as ID\_originalFileName.
5. **Enabling Notification Feature:-** A simple technique is implemented. For this feature a global variable is maintained. Whenever Receiver route receive something then it override the variable with sender ID otherwise just a empty string. When a request from frontend is received for any update then the variable value is send in response.

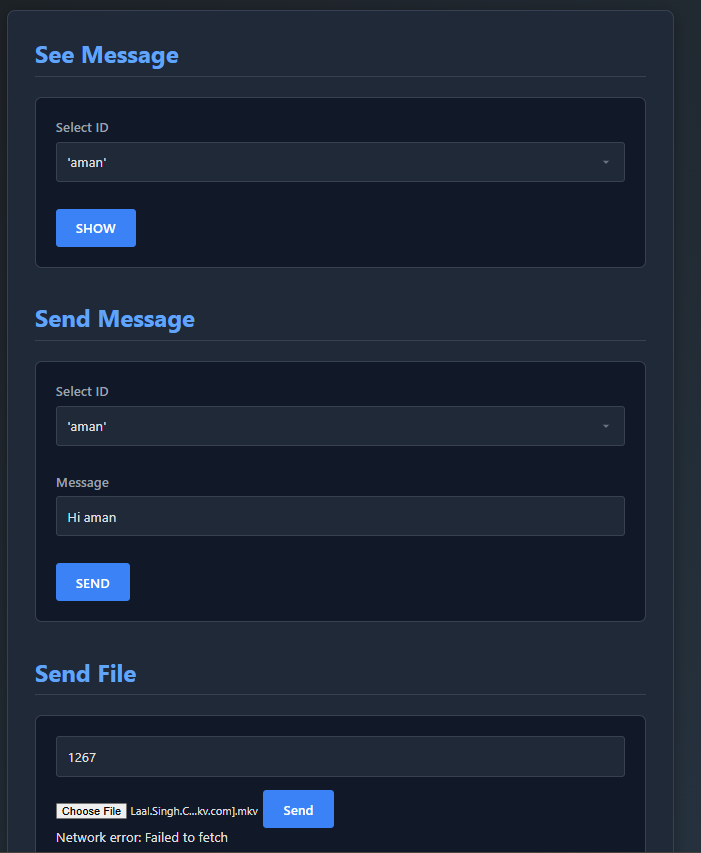
**Processor Files: -**

These are the collection of program files which helps in any additional complex processing, removing the burden of complex computations from API Handler. It is also a part of backend. It has two files:-

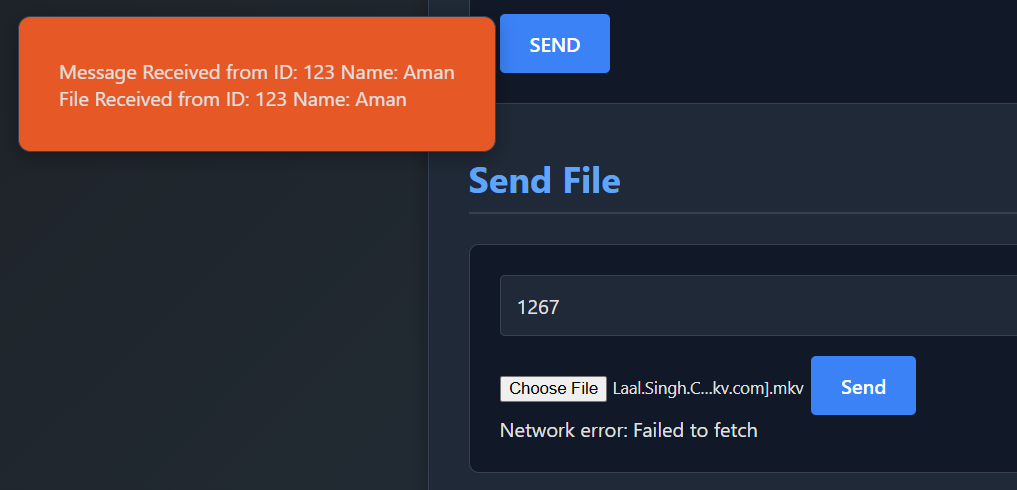
1. **IP\_Address\_Generator :-** Its function is to generate a list of IP addressed after performing computations on subnet mask and user current IP address and return this list to the API Handler
2. **Database\_Access :**- It further smoothens the process of interacting with database. It has several function defined. Like:-
   * 1. **getIP(id) –** return the IP address
     2. **addIP(id, ip, name) –** update/add the database with given details
     3. **getMessage(id) –** return list of all sent/received message on given id
     4. **getName(id) –** return name of given id
     5. **addMessage(id, message, send/receive) –** append the latest message to given id
     6. **getNameIdList():** return a list of all id and corresponding name

**SCREENSHOTS (REAL TIME):**

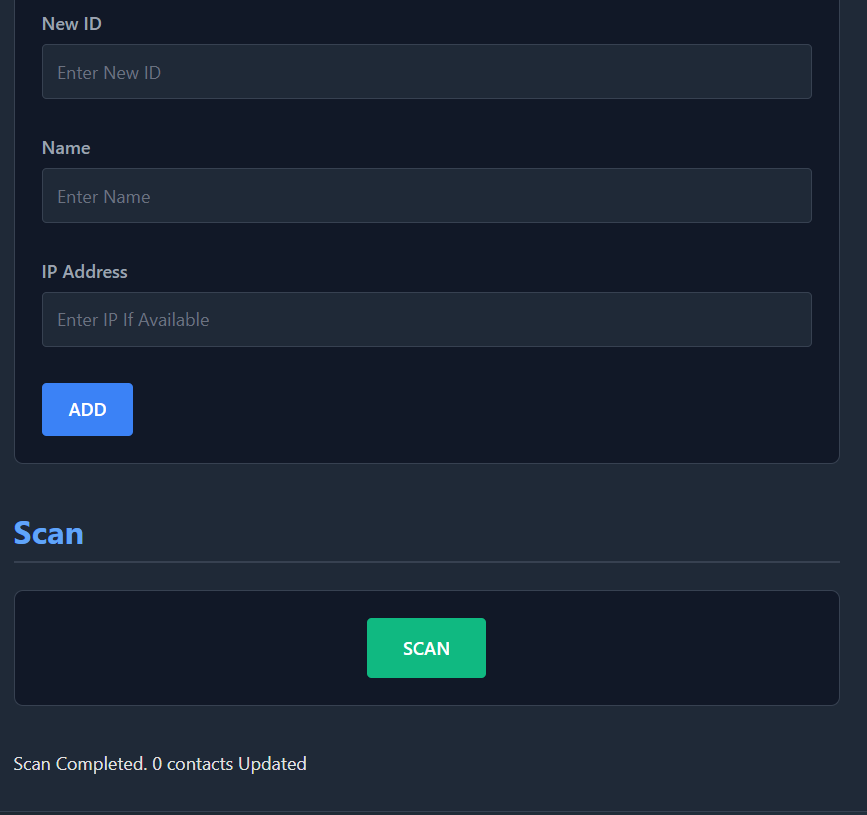
**User Interface: -**



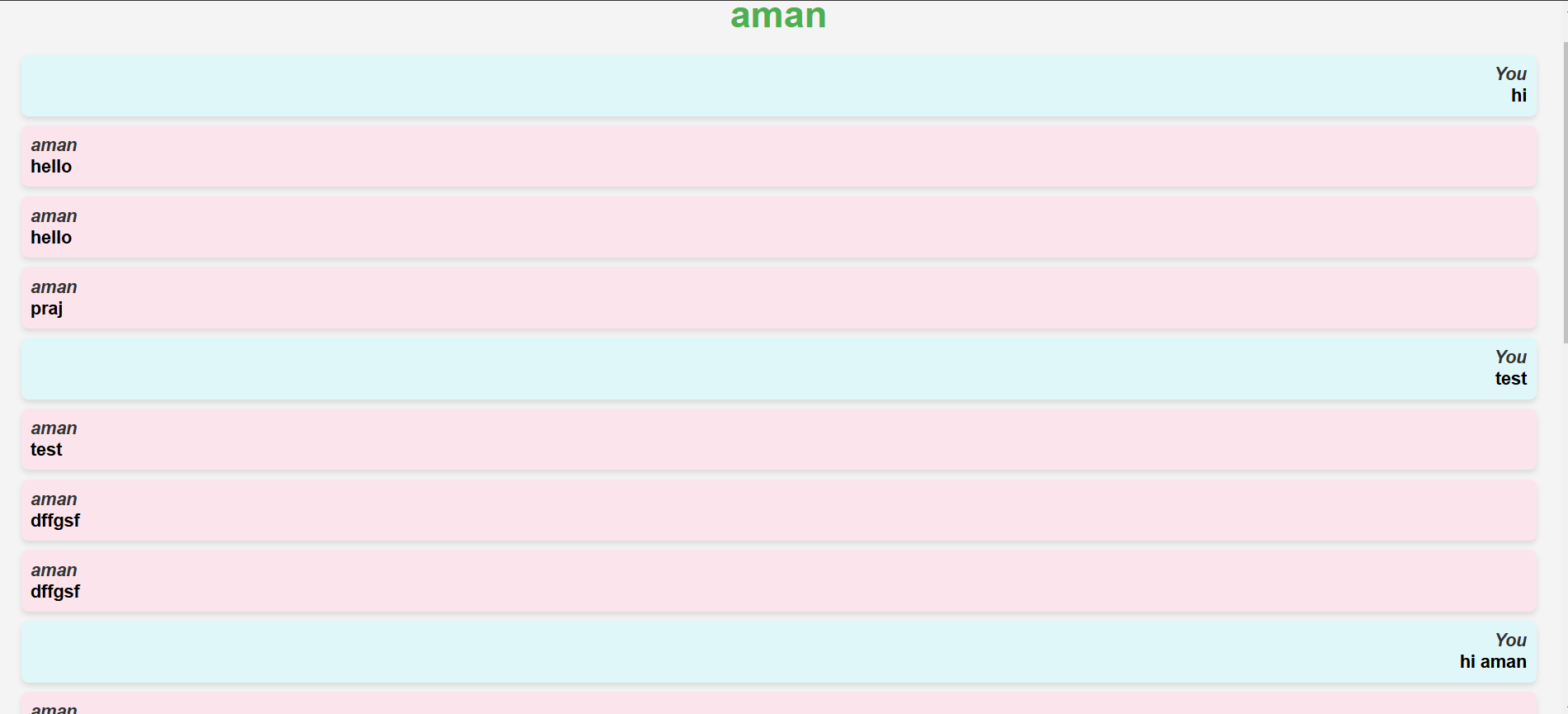
**Users can either view their previous messages, send a new message or file by selecting their saved contact from dropdown.**

****

**Get the notification and update**

****

**Add contact to the database and scan to update its current IP address**



**Interactive User Interface for messages**

**Testing, Bugs and Errors: -**

Current implementation doesn’t pass from industry standard testing. However, a robust error handling is maintained throughout the application to prevent it from crash in case of possible error, instead display informative message. A bug is found as follow – during scanning an existing IP address in network gets skipped, on after restarting the server it works properly.

**RESULTS & INFERENCES:**

1. **Seamless Peer-to-Peer Communication**: The application successfully facilitates direct peer-to-peer communication between devices on the same local network. Users can send and receive text messages and files without routing through a centralized server, minimizing delays and enhancing data privacy.
2. **Reliable Device Detection and Network Scanning**: The IP scanning functionality consistently identified active devices on the local network. Through parallelized HTTP requests, the scanning process proved to be both efficient and responsive, accurately updating the database with the current IP addresses of detected devices.
3. **Real-Time Notification Service**: The notification feature was able to keep users updated on incoming messages or file transfers in near real-time. The asynchronous, non-blocking implementation in JavaScript allowed for uninterrupted user interactions.
4. **File Transfer**: The file sender and receiver functions successfully managed file transfers of various formats and sizes.
5. **Efficiency with Processor Programs**: By separating IP generation and database access to specialized processor programs, the application achieved a high level of efficiency. This modular approach separate computationally intensive tasks from the API handler, reducing latency and improving overall performance.
6. **Scalability and Modularity**: The modular design of the application, with separate roles for frontend, API handler, and processor programs, makes it scalable. Each component can be improved or extended independently, allowing new features or optimizations without affecting the overall system.
7. **Reduced Latency Through Decentralization**: By implementing a decentralized model where each user acts as both client and server, the application minimized data transfer times and network latency. This structure is particularly effective in closed network environments like local networks where real-time, peer-to-peer interactions are crucial

**CONCLUSION:**

The project achieved its objective of designing a local area network with a server-less communication network whereby the users get to connect in real-time and exchange information. The project is divided into three components, Frontend, API handler, and processor, which allows for both development and application of the project in a fast, flexible and scalable way. Despite these advantages, a number of issues were experienced in due course of developing the project. For instance, sometimes in the course of the project, active devices erroneously failed to be detected in the course of IP scanning, which could only be rectified by restarting the server. Moreover, issues like unencrypted message and file communication raise challenges that need to be addressed in the next iterations of the project. Future work could imply encryption mechanism enforcement, improved device verification mechanism or extending the system to offer services to a wider area than just local areas. The present edition is still in the pilot stage but in its structure, there are promising aspects that can be harnessed for practical purposes after some modifications.