you can definitely implement a **Client-Server architecture** where the server stores data in a dictionary and provides the requested data when the client sends a request. The server will wait for a specific key from the client and respond with the corresponding value from the dictionary.

Here’s an implementation of a **Server** that stores a dictionary of data and a **Client** that requests data from the server:

**Server Code (server.py)**

import socket

# Define server host and port

HOST = '127.0.0.1' # localhost

PORT = 65432 # Port to listen on

# Create a dictionary to store data

data\_store = {

"name": "Alice",

"age": "25",

"city": "New York",

"occupation": "Engineer"

}

# Create a socket object

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as server\_socket:

# Bind the socket to the host and port

server\_socket.bind((HOST, PORT))

# Start listening for incoming connections

server\_socket.listen()

print(f"Server listening on {HOST}:{PORT}...")

# Accept a client connection

conn, addr = server\_socket.accept()

with conn:

print(f"Connected by {addr}")

while True:

# Receive data from the client (the key to request data)

key = conn.recv(1024).decode()

if not key:

break

print(f"Client requested: {key}")

# Look for the key in the dictionary and send the value back to the client

response = data\_store.get(key, "Key not found!")

conn.sendall(response.encode())

Client Code (client.py)

import socket

# Define server address and port

HOST = '127.0.0.1' # The server's hostname or IP address

PORT = 65432 # The port used by the server

# Create a socket object

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as client\_socket:

# Connect to the server

client\_socket.connect((HOST, PORT))

# Ask the user for a key to request data

key = input("Enter the key to request from server (e.g., name, age, city, occupation): ")

# Send the key to the server

client\_socket.sendall(key.encode())

# Receive the server's response (value)

data = client\_socket.recv(1024)

# Print the server's response

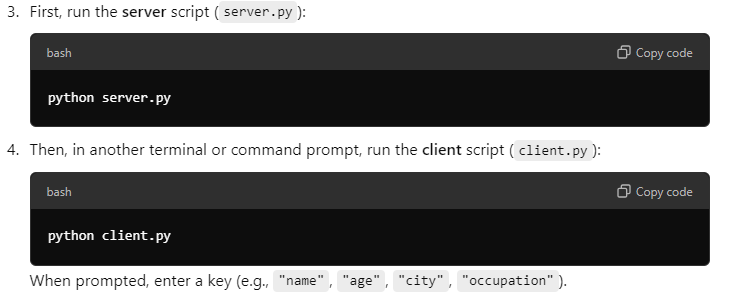
print(f"Received from server: {data.decode()}")

**How It Works:**

1. The **server** stores some predefined data in a dictionary (data\_store).
2. The **client** sends a request for a specific key (e.g., "name", "age", "city") to the server.
3. The **server** looks up the key in its dictionary and sends the corresponding value back to the client.
4. If the key is not found, the server sends a "Key not found!" message.

**How to Run:**

1. Save the **server code** in a file called server.py.
2. Save the **client code** in a file called client.py.



A screenshot of a computer

Description automatically generated

use a web framework like **Flask** for the server-side logic and basic **HTML/CSS/JavaScript** for the front-end.

Here's how you can integrate the **Client-Server architecture** with a simple web interface:

**Steps:**

1. Use **Flask** to handle HTTP requests (front-end interacts with Flask server).
2. Create a simple **HTML form** where users can input their requests.
3. Flask will receive the user input, look up the data in the dictionary, and send back the result.

**Server Code (Flask with Python)**

First, install Flask if you don’t have it installed:

bash

Copy code

pip install flask

Now, modify your server to handle HTTP requests and serve a web page.

**server.py (Flask Server)**

python

Copy code

from flask import Flask, request, render\_template

app = Flask(\_\_name\_\_)

# Dictionary to store data

data\_store = {

"name": "Alice",

"age": "25",

"city": "New York",

"occupation": "Engineer"

}

# Define the route for the homepage

@app.route('/')

def home():

return render\_template('index.html') # Renders the front-end HTML page

# Define the route to handle data requests

@app.route('/get-data', methods=['POST'])

def get\_data():

key = request.form.get('key') # Get the key from the user input

value = data\_store.get(key, "Key not found!") # Lookup key in dictionary

return render\_template('index.html', result=value) # Return the result to front-end

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**Front-End (HTML) Code**

Create a folder called templates in the same directory as your server.py file, and inside it, create an index.html file:

**templates/index.html**

html

Copy code

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Client-Server Data Request</title>

<style>

body {

font-family: Arial, sans-serif;

margin: 0;

padding: 0;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

background-color: #f4f4f4;

}

.container {

text-align: center;

padding: 20px;

background-color: white;

border-radius: 8px;

box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);

}

input[type="text"] {

padding: 10px;

width: 200px;

margin: 10px 0;

}

input[type="submit"] {

padding: 10px 20px;

background-color: #4CAF50;

color: white;

border: none;

border-radius: 4px;

cursor: pointer;

}

input[type="submit"]:hover {

background-color: #45a049;

}

.result {

margin-top: 20px;

font-size: 18px;

color: #333;

}

</style>

</head>

<body>

<div class="container">

<h1>Request Data from Server</h1>

<form method="POST" action="/get-data">

<input type="text" name="key" placeholder="Enter key (e.g., name, age)" required>

<br>

<input type="submit" value="Get Data">

</form>

<div class="result">

{% if result %}

<p>Result: {{ result }}</p>

{% endif %}

</div>

</div>

</body>

</html>

**How It Works:**

1. The Flask server hosts a webpage (index.html) with a form that allows users to input a key (like name, age, city, etc.).
2. When the user submits the form, the server looks up the key in the data\_store dictionary.
3. The server returns the result (or an error message if the key is not found) and displays it on the same page.

**How to Run:**

1. Save your server.py file and the index.html file in a templates folder.
2. Run the Flask server:

bash

Copy code

python server.py

1. Open a browser and go to http://127.0.0.1:5000/.
2. Enter a key (like name, age, city, occupation) into the input field and click **Get Data**. The result will be displayed on the page.

**Example Interaction:**

* **User Input**: Enter "name" and submit the form.
* **Server Response**: The server will look up "name" in the dictionary and return "Alice".

**Output Example:**

* The webpage will display:

makefile

Copy code

Result: Alice

This approach extends the original **Client-Server architecture** to a web-based front-end where users can easily interact with the server via a browser. You can further enhance the front-end using CSS/JavaScript for a better user experience!