مستندات پروژه WEB SERVER درس شبکه های کامپیوتری

استاد:

جناب دکتر حسین غفاریان

دانشجویان:

دانیال ایرانمهر - امیرحسین مجیدی

۱۴۰۳/۳/۲۹

نیم سال دوم تحصیلی ۰۲-۰۳

فهرست

مقدمه

در این پروژه وب سروری برای پاسخ به درخواست‌های HTTP پیاده‌سازی شده است. همچنین، مفاهیم سیستم‌عاملی همچون چند ریسمانی و ناحیه بحرانی در آن در نظر گرفته شده است.

زبان پیاده‌سازی این پروژه Python بوده است. همچنین به کمک HTML و CSS صفحات وبی طراحی شده است که ما را قادر می‌سازد وب سرور را برروی مرورگر کامپیوتر فراخوانی کنیم.

ارکان اصلی این پروژه درون توابع متنوع پیاده‌سازی شده است که تست و خوانایی آن را افزایش می‌دهد.

در ادامه با توابع و جزئیات عملکرد پروژه آشنا می‌شوید.

کتابخانه‌ها

Socket

کتابخانه socket در پایتون برای برقراری ارتباط شبکه‌ای و انتقال داده بین دستگاه‌ها استفاده می‌شود. این کتابخانه به ما امکان می‌دهد تا از پروتکل‌های مختلف شبکه مانند TCP و UDP برای ارتباطات شبکه‌ای استفاده کنید.

ایجاد سوکت سرور

در اینجا یک سوکت TCP سرور ایجاد شده و به IP و پورت مشخصی متصل شده و برای گوش دادن به اتصالات ورودی آماده می‌شود.

```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((server_ip, server_port))
server_socket.listen(5)
```

پذیرفتن اتصالات ورودی

سرور با فراخوانی متد accept، اتصال یک کلاینت را می‌پذیرد و یک سوکت جدید برای ارتباط با آن کلاینت ایجاد می‌کند.

```
client_socket, client_address = server_socket.accept()
```

ارسال و دریافت داده‌ها

در اینجا داده‌ها از کلاینت دریافت شده (recv) و پاسخ مناسب به کلاینت ارسال می‌شود (sendall).

```
request = client_socket.recv(1024).decode()
client_socket.sendall(response.encode())
```

بستن سوکت‌ها

پس از اتمام ارتباط، سوکت‌های مربوط به سرور و کلاینت بسته می‌شوند.

```
client_socket.close()
server_socket.close()
```

Threading

یکی از کتابخانه‌هایی که ما در این پروژه استفاده کردیم، کتابخانه threading است.این کتابخانه قابلیت برنامه‌نویسی چند ریسمانی را برای ما فراهم کرده که به ما این امکان را داد که وظایف مختلف را به طور همزمان در برنامه وب سرور اجرا کنیم.

در واقع Thread کوچکترین واحد قابل اجرا در یک برنامه است. هر thread می تواند به طور مستقل کد خود را اجرا کند. این کتابخانه امکاناتی برای پیاده‌سازی مفاهیم سیستم عاملی نظیر چند ریسمانی و کنترل ناحیه بحرانی را در اختیار ما می‌گذارد.

برای پیاده‌سازی مفهوم چند ریسمانی از کلاس Thread از این کتابخانه استفاده کردیم. همچنین برای پیاده‌سازی کنترل ناحیه بحرانی از کلاس Lock استفاده کردیم. این کلاس برای محافظت از دسترسی همزمان تردها به منابع مشترک ناحیه بحرانی مورد استفاه قرار می‌گیرد.

مفاهیم سیستم‌عاملی

ناحیه بحرانی

ناحیه‌های بحرانی (critical sections) در جاهایی که دسترسی به منابع مشترک صورت می‌گیرد، با استفاده از قفل (lock) مدیریت شده‌اند. این کار برای جلوگیری از مشکلات رقابتی (race conditions) و اطمینان از دسترسی همزمان ایمن به این منابع مشترک انجام می‌شود. نحوه مدیریت نواحی بحرانی در این پروژه به شرح زیر است:

قبل از پردازش درخواست کلاینت

در تابع handle_client با استفاده از قفل connected_ips_lock، دسترسی به لیست connected_ips را کنترل می‌کند.

در این بخش، دسترسی به لیست connected_ips و مجموعه blocked_ips با استفاده از قفل مدیریت می‌شود تا از دسترسی همزمان ناایمن جلوگیری شود.

```
with connected_ips_lock:
    if ip in blocked_ips:
        print(f"Blocked IP {ip} tried to connect.")
        client_socket.close()
        return
    list_size = len(set(connected_ips))
    connected_ips.append(ip)
```

بعد از پردازش درخواست کلاینت

در اینجا نیز برای حذف آی‌پی از لیست connected_ips، از قفل استفاده شده است.

```
finally:
    with connected_ips_lock:
        if ip in connected_ips:
            connected_ips.remove(ip)
    client_socket.close()
```

هنگام پذیرش اتصال جدید

در تابع start_server، دسترسی به لیست connected_ips و مجموعه firewall_ips با استفاده از قفل مدیریت می‌شود.

```
while True:
    client_socket, client_address = server_socket.accept()
    ip = client_address[0]
    if ip not in connected_ips:
        connected_ips.append(client_address[0])
    with connected_ips_lock:
        if ip in firewall_ips:
            connected_ips.remove(ip)
            print(f"\033[91mFirewall blocked IP {ip} from connecting.\033[00m")
            client_socket.close()
            continue
```

همچنین توابع show_connected_ips، disconnect_and_block_ip، unblock_ip و show_blocked_ips نیز برای دسترسی به IPهای مختلف از قفل استفاده می‌کنند.

چند ریسمانی

همانطور که گفته شد ما در این کد از کتابخانه threading استفاده کرده ایم. این کتابخانه دارای کلاسی به نام Thread است که با استفاده از آن ما می‌توانیم تردهای جداگانه برای کلاینت‌هایی که به سرور متصل می‌شوند ایجاد کنیم تا هر کلاینت بتواند به صورت جداگانه به سرور متصل شود و ما بتوانیم در وب سرور خود به صورت همزمان چند کلاینت را پاسخگو باشیم.

در کنار آن، وب سرور ما قابلیت مدیریت کردن را نیز دارد و همزمان که کلاینت‌ها می‌توانند به سرور متصل شوند، امکان مدیریت سرور نیز برای مدیر فراهم است که این موضوع نیاز دارد که برای مدیریت وب سرور هم یک ترد جداگانه در نظر بگیریم تا کارهایی مانند نمایش منو و دریافت دستورات مدیریتی به صورت موازی فراهم شود. این مورد در خطوط شماره ۲۶۱ و ۲۶۲ به صورت زیر صورت گرفته است:

```
server_thread = threading.Thread(target=start_server, args=("127.0.0.1", 8080))
server_thread.start()
```

تابع handle_client برای مدیریت هر اتصال کلاینت طراحی شده است. این تابع در یک ریسمان جداگانه برای هر کلاینت اجرا می‌شود تا بتوان همزمان چندین کلاینت را مدیریت کرد.

ایجاد تردهای جداگانه برای هر کلاینت در تابع start_server و در خطوط شماره ۱۷۰ و ۱۷۱ صورت می‌گیرد. در این قسمت پس از پذیرش کلاینت، یک ترد (ریسمان) جداگانه به آن کلاینت اختصاص داده می‌شود.

```
client_handler = threading.Thread(target=handle_client, args=(client_socket, client_address))
client_handler.start()
```

آرگومان‌های مورد نیاز که شامل سوکت کلاینت و آدرس کلاینت است، به تابع handle_client منتقل می‌شود تا در آن قسمت پاسخ مناسب به کلاینت ارسال شود.

توابع

handle_client

ورودی‌های تابع

- client_socket: سوکتی که اتصال کلاینت را نمایندگی می‌کند.
- client_address: آدرس کلاینت که شامل IP و شماره پورت است.

عملکرد تابع

تابع handle_client برای مدیریت اتصال کلاینت‌ها به سرور طراحی شده است. این تابع درخواست کلاینت را دریافت می‌کند، آن را پردازش می‌کند تا کد وضعیت HTTP مناسب و پیام را تعیین کند، پاسخ را ایجاد کرده و به کلاینت ارسال می‌کند. همچنین اگر IP کلاینت مسدود شده باشد، اتصال بلافاصله بسته می‌شود. در ادامه، به توضیح کامل این تابع می‌پردازیم.

با استفاده از قفل connected_ips_lock اگر IP در لیست مسدود شدگان (blocked_ips) باشد، اتصال بسته می‌شود و تابع خاتمه می‌یابد.

در غیر این صورت، IP به لیست connected_ips(لیستی از IPهای متصل) اضافه می‌شود و اندازه لیست متصل شدگان محاسبه می‌شود.

```
with connected_ips_lock:
    if ip in blocked_ips:
        print(f"Blocked IP {ip} tried to connect.")
        client_socket.close()
        return
    list_size = len(set(connected_ips))
    connected_ips.append(ip)
```

این بخش از تابع handle_client مسئول دریافت و پردازش درخواست HTTP کلاینت است. در ابتدا، درخواست کلاینت به صورت باینری از سوکت دریافت و به رشته تبدیل می‌شود. سپس، هدرهای درخواست با جدا کردن خطوط (بر اساس کاراکتر newline) تجزیه می‌شوند. هدر اول که شامل اطلاعات اصلی درخواست HTTP است (مانند متد و مسیر درخواست) به اجزای خود تقسیم شده و متد و مسیر درخواست استخراج می‌شود. در صورتی که هرگونه خطای ایندکس یا متغیر تعریف نشده رخ دهد، با استفاده از بلاک except مدیریت می‌شود و برنامه بدون متوقف شدن به کار خود ادامه می‌دهد. این قسمت از کد به منظور آماده‌سازی درخواست برای پردازش بیشتر و تعیین پاسخ مناسب استفاده می‌شود.

```
request = client_socket.recv(1024).decode()
    try:
        headers = request.split('\n')
        first_header_item = headers[0].split()
        method = first_header_item[0]
        path = first_header_item[1]
    except IndexError or UnboundLocalError:
        pass
```

در خارج از این تابع لیستی از کدهای وضعیت مختلف در قالب یک دیکشنری به نام http_status_codes تعبیه شده است.

```
http_status_codes = {
    200: "OK",
    201: "Created",
    202: "Accepted",
    204: "No Content",
    301: "Moved Permanently",
    302: "Found",
    304: "Not Modified",
    400: "Bad Request",
    401: "Unauthorized",
    403: "Forbidden",
    404: "Not Found",
    500: "Internal Server Error",
    501: "Not Implemented",
    503: "Service Unavailable"
}
```

سپس client با توجه به مسیر path، کد وضعیت مناسب به متغییر status_code انتساب داده می‌شود. سپس در لیستی به نام bad_status_codes، کدهای وضعیتی که نمایانگر وجود مشکل در اتصال هستند، قرار داده شده است و اگر مقدار متغییر status_code در این لیست وجود داشت، فایل Browser2.html فراخوانی می‌شود و در غیراین صورت فایل Browser.html.

این بخش از کد مسئول تولید محتوای بدنه (body) پاسخ HTTP است که به کلاینت ارسال می‌شود. این کار با استفاده از یک قالب (template) انجام می‌گیرد که شامل متغیرهای قالب‌بندی شده است.

```
response_body = template.format(
            status_code=status_code,
            status_message=status_message,
            list_size=list_size
        )
```

این بخش نیز مسئول تولید خطوط سرآیند و بدنه پاسخ HTTP به کلاینت است.

```
response = (
            f"HTTP/1.1 {status_code} {status_message}\r\n"
            "Content-Type: text/html\r\n"
            f"Content-Length: {len(response_body)}\r\n"
            "\r\n"
            f"{response_body}"
        )
```

پاسخ HTTP با استفاده از قالب و مقادیر مناسب ساخته شده و با استفاده از sendall به کلاینت ارسال می‌شود.

```
client_socket.sendall(response.encode())
```

این بخش از کد، در بخش finally قرار دارد که همواره پس از اجرای بخش‌های try و except اجرا می‌شود.

```
    finally:
        with connected_ips_lock:
            if ip in connected_ips:
                connected_ips.remove(ip)
        client_socket.close()
```

start_server

این تابع وظیفه راه‌اندازی سرور و مدیریت اتصالات ورودی کلاینت‌ها را بر عهده دارد. این تابع برای اطمینان از قابلیت همزمانی و مدیریت موثر منابع، از چند ریسمانی استفاده می‌کند. این تابع در ابتدا آدرس IP و شماره پورت سرور (یعنی آی پی و شماره پورت سیستم خودمان) را به عنوان آرگومان ورودی دریافت می‌کند.

```
def start_server(server_ip, server_port):
```

این تابع در خط 261 پس از ایجاد ریسمان مربوط به سرور فراخوانی می‌شود. ما به طور پیش‌فرض آی پی سرور را 127.0.0.1 (آدرس Loopback) و شماره پورت 8080 قرار داده‌ایم.

```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((server_ip, server_port))
server_socket.listen(5)
```

سپس سوکت سرور ایجاد می‌شود. سوکتی که ما ایجاد کردیم از نوعIP نسخه۴ است و برای اتصال از پروتکل TCP استفاده می‌کند. سپس سوکت را به آی پی و پورت مشخص شده متصل کرده و سرور در حالت گوش دادن برای پذیرفتن کلاینت‌ها قرار می‌گیرد.

```
while True:
    client_socket, client_address = server_socket.accept()
    ip = client_address[0]
    if ip not in connected_ips:
        connected_ips.append(client_address[0])

    with connected_ips_lock:
        if ip in firewall_ips:
            connected_ips.remove(ip)
            print(f"\033[91mFirewall blocked IP {ip} from connecting.\033[00m")
            client_socket.close()
            continue
    print("\033[93m\n--------------------------------------------------------------------------------
\033[00m")
```

```
    print(f"\033[93mA client with IP Address {client_address[0]} and Socket Number {client_address[1]} has connected.\033[00m")

    print("\033[93m--------------------------------------------------------------------------------\033[00m")

    client_handler = threading.Thread(target=handle_client, args=(client_socket, client_address))

    client_handler.start()
```

در ادامه در یک حلقه بی‌نهایت، سرور به طور مداوم منتظر اتصالات ورودی کلاینت‌ها است. با پذیرش هر اتصال جدید، سوکت و آدرس کلاینت در متغیرهای client_socket و client_address ذخیره می‌شوند.

پس از پذیرفتن کلاینت، آدرس IP آن در متغیر ip ذخیره شده و اگر آی پی تکراری نبود(با توجه به اینکه کلاینت با یک آی پی و چند شماره سوکت مختلف می‌تواند به سرور متصل شود) به لیست connected_ips که یک لیست از آی پی‌های متصل به سرور است، اضافه می‌شود.

برای مدیریت ناحیه بحرانی و جلوگیری از تداخل همزمان ناشی از دسترسی به لیست connected_ips، ما از یک قفل استفاده کرده‌ایم. سپس چک می‌شود که اگر آی پی کلاینت در لیست فایروال است، آی پی او از لیست آی‌پی‌های متصل حذف و سوکت کلاینت بسته می‌شود تا از دسترسی او به وب سرور جلوگیری کنیم.

در ادامه پیامی در قالب رنگی، چاپ می‌کنیم که کلاینت به سرور متصل شده است و یک ترد(ریسمان) به این کلاینت اختصاص می‌دهیم تا شرایط چندریسمانی بودن را فراهم کنیم و چند کلاینت را به طور همزمان در وب سرورمان پاسخگو باشیم.

server_management

این تابع وظیفه مدیریت سرور را برعهده دارد و امکاناتی نظیر نمایش پیام‌های مدیریتی، گرفتن ورودی از کیبورد مدیر سرور(برای مدیریت سرور و انتخاب امکانات)، نمایش آی‌پی‌های متصل، مسدودسازی آی‌پی، رفع مسدودیت آی‌پی، نمایش آی‌پی‌های مسدود شده و بستن سرور را دارد. در واقع توابع مدیریتی که قبل‌تر توضیح داده شد، توسط این تابع فراخوانی می‌شوند.

این تابع در یک حلقه بی‌نهایت، ابتدا پیام‌های مدیریتی را نمایش می‌دهد تا مدیر بتواند گزینه مورد نظرش را انتخاب کند. سپس از مدیر یک ورودی که یک عدد است را دریافت می‌کند. این عدد ورودی باید بین ۱ تا ۵ باشد.

هرعدد مربوط به فراخوانی یک تابع می‌شود و در یک ساختار شرطی، عدد وارد شده توسط مدیر مقایسه می‌شود و گزینه مورد نظر انتخاب می‌شود. برخی از توابع که نیاز به آی‌پی دارند(برای مثال برای مسدود کردن یا رفع مسدودیت یک آی پی)، این آی‌پی را از مدیر دریافت می‌کنند و سپس آن را به عنوان آرگومان به تابع مربوطه پاس می‌دهند.

show_connected_ips

عملکرد تابع

این تابع به منظور نمایش لیست آدرس‌هایIPهایی که در حال حاضر به سرور متصل هستند، طراحی شده است.

درصورتی که لیست connected_ips هیچ عنصری نداشته باشد، پیامی مناسب نمایش داده می‌شود.

با حلقه‌ای برروی لیست connected_ips که تبدیل به set شده(به جهت جلوگیری از وجود IP تکراری) پیمایش می‌کنیم و در هر پیمایش، اگر آن IP در لیست blocked_ips(لیستی برای ذخیره IPهای مسدود شده) نبود، آن را با فرمتی مناسب نمایش می‌دهیم.

```
global connected_ips
    with connected_ips_lock:
        print("\033[96mConnected IPs:\033[00m")
        if len(connected_ips) == 0:
            print("\033[91mThere is no Connected IP!\033[00m")
        cc = 0
        for ip in set(connected_ips):
            if ip not in blocked_ips:
                if cc == len(set(connected_ips))-1:
                    print("\033[92m" + ip + "\033[00m")
                else:
                    print("\033[92m" + ip + "\033[00m", end=" - ")
                cc += 1
```

disconnect_and_block_ip

این تابع وظیفه قطع اتصال کلاینت و قراردادن او در لیست آی‌پی‌های مسدود شده را دارد تا از اتصال مجدد او به سرور جلوگیری کنیم.

```
def disconnect_and_block_ip(ip):


    global connected_ips, blocked_ips
    with connected_ips_lock:
        if ip in connected_ips:
            connected_ips.remove(ip)
            blocked_ips.add(ip)
            print("\033[96mDisconnected and blocked IP: \033[00m" + "\033[92m" + ip + "\033[00m")
        else:
            print(f"\033[91mIP {ip} not found in connected IPs.\033[00m")
```

این تابع در آرگومان ورودی خود، یک آی‌پی که در قالب رشته است را دریافت می‌کند. این آی‌پی همان آی‌پی است که قصد داریم آن را مسدود کنیم. در اینجا نیز با استفاده از قفل‌ها از ناحیه بحرانی محافظت می‌شود. اگر آی‌پی داده شده در لیست آی‌پی‌های متصل باشد، ابتدا از آن لیست حذف شده و سپس در لیست blocked_ips که مربوط به آی‌پی‌های مسدود شده است درج می‌شود و پیغام مربوط به آن چاپ می‌شود. اگر که آی‌پی در لیست آی‌پی‌ها نباشد هم پیغام خطا را چاپ می‌کنیم.

unblock_ip

این تابع وظیفه رفع مسدودیت یک آی‌پی را برعهده دارد. در واقع این تابع می‌تواند یک آی‌پی را هم از لیست مسدودیت‌ها و هم از لیست فایروال حذف کند.

این تابع در آرگومان ورودی خود یک آی‌پی را دریافت می‌کند که این آی‌پی همان آی‌پی است که قصد داریم مسدودیتش را رفع کنیم. در اینجا نیز برای مدیریت ناحیه بحرانی از یک قفل استفاده کرده‌ایم.

```
def unblock_ip(ip):
    global connected_ips, blocked_ips
    with connected_ips_lock:
        if ip in blocked_ips or ip in firewall_ips:
            try:
                firewall_ips.remove(ip)
            except:
                blocked_ips.discard(ip)
            print("\033[96mIP \033[00m" + "\033[92m" + ip + "\033[00m" + "\033[96m has been unblocked from Blocked-List and Firewall.\033[00m")
        else:
            print(f"\033[91mIP {ip} is not in Blocked-List or Firewall!\033[00m")
```

در ابتدا این آی‌پی چک می‌شود که آیا در لیست مسدودیت(blocked_ips) یا لیست فایروال (firewall_ips) قرار دارد یا خیر. سپس با استفاده از متد try-except برای پیشگیری از خطاهای احتمالی تلاش می‌شود که اگر آی‌پی در فایروال و یا لیست مسدودیت قرار دارد حذف شود. پس از آزاد شدن آی‌پی از این لیست‌ها، پیغام موفقیت این عمل چاپ می‌شود و کلاینت می‌تواند به سرور متصل شود.

در صورتی که آی‌پی در این لیست‌ها وجود نداشته باشد، پیغام خطا چاپ می‌شود.

show_blocked_ips

این تابع وظیفه‌ی نمایش IPهای مسدود شده را به عهده دارد.

در کد، لیستی به نام firewall_ips وجود دارد که در آن IPهایی که بصورت پیش فرض مسدود در نظر گرفته می‌شوند، ذخیره شده است. همچنین لیستی به نام blocked_ips نیز وجود دارد که IPهایی که بصورت دستی مسدود شده اند را ذخیره می‌کند.

این تابع هم لیست IPهای موجود در firewall_ips و هم لیست IPهای موجود در blocked_ip را نمایش می‌دهد.

نمایش firewall_ips:

```
lobal firewall_ips, blocked_ips
    print("\033[96mFirewall-IPs:\033[00m")
    if len(firewall_ips) == 0:
        print("\033[91mThere is no IP in Firewall!\033[00m")
    else:
        cf = 0
        for ip in firewall_ips:
            if cf == len(firewall_ips)-1:
                print("\033[92m" + ip + "\033[00m")
            else:
                print("\033[92m" + ip + "\033[00m", end=' - ')
            cf += 1
```

نمایش blocked_ip:

```
print("\033[96mBlocked-IPs:\033[00m")
    if len(blocked_ips) == 0:
        print("\033[91mThere is no IP in Blocked List!\033[00m")
    else:
        cb = 0
        for ip in blocked_ips:
            if cb == len(blocked_ips)-1:
                print("\033[92m" + ip + "\033[00m")
            else:
                print("\033[92m" + ip + "\033[00m", end=' - ')
            cb += 1
```

فایل‌های HTML

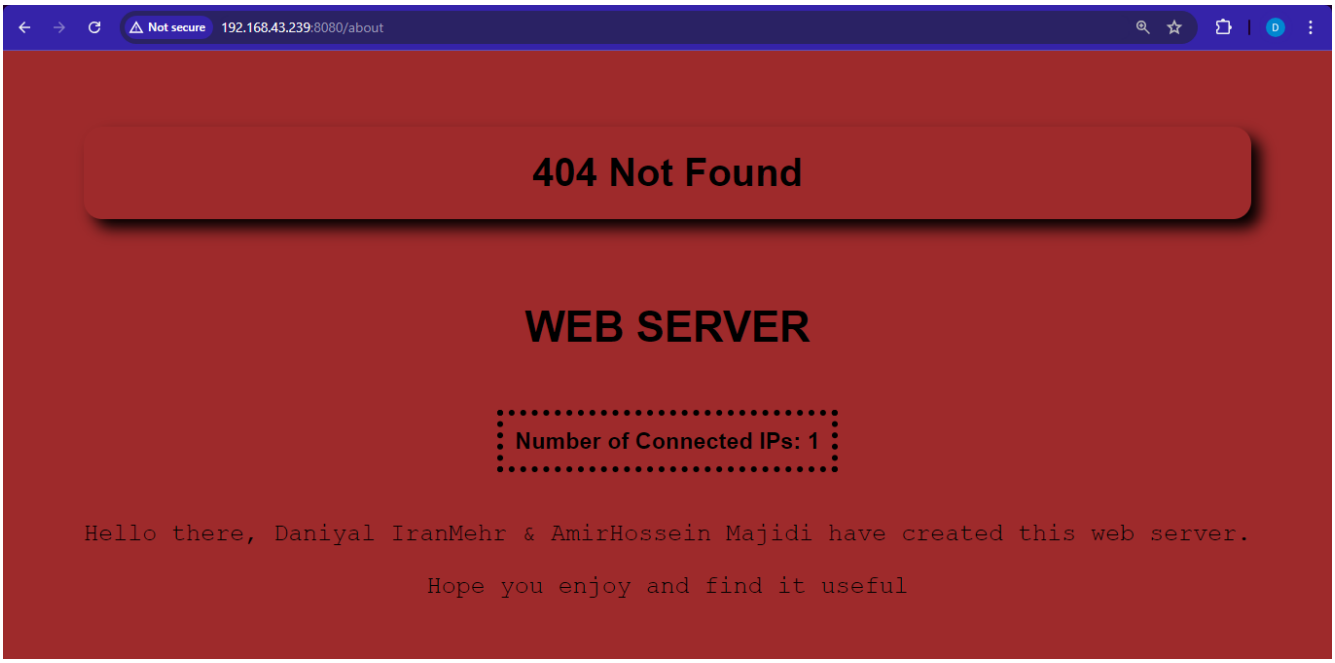
در فایل‌های html کدهای HTML و CSS ادقام شده‌اند تا طراحی زیبا و مناسبی را پدید آورند.

در فایل‌های HTML، کد وضعیت به همراه پیام وضعیت در باکس سه‌بعدی نمایش داده می‌شود. سپس عنوان WEB SERVER و تعداد IPهای متصل به وب سرور نمایش داده می‌شود. در نهایت پیامی از طرف دانشجویان انجام دهنده‌ی این پروژه قرار داده شده است.

فایل Browser2.html

این فایل هنگامی فراخوانی می‌شود که کد وضعیت مورد نظر جز کدهای مشکل ساز باشد.

تم رنگی که به کمک CSS در مرورگر نمایش داده می‌شود، #9e2a2b است که قرمز بودن آن نمایانگر وجود ایراد در اتصال است.



فایل Browser.html

این فایل هنگامی فراخوانی می‌شود که کد وضعیت مورد نظر جز کدهای مشکل ساز نباشد و client با موفقیت به وب سرور متصل شود.

تم رنگی که به کمک CSS در مرورگر نمایش داده می‌شود، lightblue است که آبی بودن آن نمایانگر اتصال صحیح به وب سرور است.