فرهاد امان ۹۹۳۱۰۰۶

سرور: این بخش یک سرور HTTP ساده است که از طریق پایتون و استفاده از ماژول http.server ایجاد شده است. سرور قادر است درخواست های GET و POST را پردازش کند و از پایگاه داده Redis برای ذخیره اطلاعات استفاده می‌کند.

- json: برای کار با داده‌های JSON.
- os: برای دسترسی به متغیرهای محیطی مانند REDIS_HOST.
- http.server: برای ایجاد سرور HTTP و پردازش درخواست‌ها.
- urllib.parse: برای تجزیه و بررسی آدرس URL و پارامترهای درخواست.

تنظیمات Redis:
- redis_host: متغیری برای ذخیره IP میزبان Redis، که در صورتی که تنظیم نشده باشد، به مقدار پیش‌فرض 'localhost' را دارد.
- redis_client: یک شی از کلاینت Redis بر اساس IP و Port مشخص شده.

کلاس RequestHandler:
این کلاس یک زیرکلاس از BaseHTTPRequestHandler است که برای پردازش درخواست‌ها و ارسال پاسخ‌ها استفاده می‌شود. این کلاس شامل دو متد do_GET و do_POST است که به ترتیب برای پردازش درخواست‌های GET و POST استفاده می‌شوند.

:do_GET
- بررسی آدرس درخواست و انجام عملیات مربوطه بر اساس آن.
- اگر آدرس /get-all باشد، تمام کلیدهای Redis را دریافت کرده و به صورت JSON به عنوان پاسخ ارسال می‌کند.
- اگر آدرس /get باشد، پارامتر username را درخواست می‌کند و مقدار مربوط به آن کلید را از Redis دریافت کرده و به عنوان پاسخ ارسال می‌کند.
- در غیر این صورت، خطای 404 با پیام "Path not found" ارسال می‌شود.

:do_POST
- بررسی آدرس درخواست و انجام عملیات مربوطه بر اساس آن.
- اگر آدرس /register باشد، داده‌های دریافتی را در قالب JSON بررسی می‌کند.
- اگر نام کاربری (username) یا آدرس (address) موجود نباشد، یا خالی باشد، خطا مربوطه را ایجاد می‌کند.

- در غیر این صورت، نام کاربری و آدرس را در Redis ذخیره می‌کند و پیام "Registration successful" را به عنوان پاسخ ارسال می‌کند.
- در غیر این صورت، خطای 404 با پیام "Path not found" ارسال می‌شود.

شروع سرور HTTP:
- در بخش اصلی کد، پورتی برای سرور تعیین می‌شود (اینجا 80 است) و سپس یک نمونه از کلاس `HTTPServer` با آدرس میزبان و پورت ایجاد می‌شود.
- سپس سرور راه‌اندازی می‌شود.
- سرور به صورت بی‌نهایت درخواست‌ها را پردازش می‌کند و منتظر درخواست‌های جدید می‌ماند.

همتا: این بخش یک برنامه P2P برای به اشتراک گذاشتن فایل بین همتایان (peers) است.

- tkinter: برای ایجاد رابط کاربری گرافیکی.
- numpy: برای کار با آرایه‌ها.
- requests: برای ارسال درخواست‌های HTTP.
- socket: برای ایجاد اتصال TCP و UDP و برقراری ارتباط شبکه.
- threading: برای استفاده از تردها و همروندسازی.
- PIL: برای کار با تصاویر.

تابع address_to_tuple:
- تبدیل یک آدرس شبکه به یک تاپل شامل آدرس IP و پورت مربوطه.

تابع tuple_to_address:
- تبدیل یک تاپل شامل آدرس IP و پورت به یک آدرس شبکه.

تابع create_tcp_server:
- برای ایجاد یک سرور TCP.
- از پورت‌های مشخص شده (TCP_PORT_START تا TCP_PORT_END) استفاده می‌کند تا یک پورت خالی برای ایجاد سرور پیدا کند.
- یک سوکت TCP ایجاد می‌کند و به پورت مشخص شده متصل می‌شود.

تابع create_udp_server:
- برای ایجاد یک سرور UDP.
- از پورت‌های مشخص شده (UDP_PORT_START تا UDP_PORT_END) استفاده می‌کند تا یک پورت خالی برای ایجاد سرور پیدا کند.

- یک سوکت UDP ایجاد می‌کند و به پورت مشخص شده متصل می‌شود.

متغیرها:
- SERVER_IP و SERVER_PORT: آدرس و پورت سرور اصلی که برنامه از آن استفاده می‌کند.
- GET_ALL_API ، REGISTER_API و GET_API: آدرس‌های API برای ثبت‌نام، دریافت همه کاربران و دریافت کاربر با نام کاربری مشخص.
- TCP_PORT_START و TCP_PORT_END: محدوده پورت‌های مورد استفاده برای اتصالات TCP.
- UDP_PORT_START و UDP_PORT_END: محدوده پورت‌های مورد استفاده برای اتصالات UDP.

تابع send_image:
- برای ارسال تصویر به طرف مقابل از پروتکل UDP استفاده می‌کند.
- از سوکت UDP استفاده می‌کند تا تصویر را برای طرف مقابل ارسال کند.
- تصویر را از فایل مشخص شده بارگیری می‌کند و به آرایه numpy تبدیل می‌کند.
- ابعاد تصویر را به عنوان یک پیام (message) به طرف مقابل ارسال می‌کند.
- تصویر را به بخش‌های کوچکتر (chunk) تقسیم کرده و هر بخش را به طرف مقابل ارسال می‌کند.
- پس از ارسال تمامی بخش‌ها، پیام "Finished" را برای اعلام اتمام ارسال ارسال می‌کند.

تابع send_text:
- برای ارسال فایل متنی به طرف مقابل از پروتکل TCP استفاده می‌کند.
- فایل متنی را از فایل مشخص شده می‌خواند و به طرف مقابل ارسال می‌کند.

تابع downloader:
- برای دریافت فایل از طرف مقابل استفاده می‌شود.
- از سوکت TCP برای برقراری ارتباط با طرف مقابل استفاده می‌کند.
- اطلاعات مربوط به درخواست (آدرس IP و پورت UDP مقصد و آدرس فایل) را به طرف مقابل ارسال می‌کند.
- اگر فایل متنی باشد، داده‌ها را دریافت کرده و در فایل مورد نظر ذخیره می‌کند.
- اگر تصویر باشد، از سوکت UDP برای دریافت بخش‌های تصویر استفاده می‌کند.
- پس از دریافت تمامی بخش‌های تصویر، آن را با استفاده از آرایه numpy مجدداً ساختاردهی می‌کند و در فایل مورد نظر ذخیره می‌کند.

تابع listener:
- برای پذیرش اتصالات و دریافت درخواست‌ها از طرف مقابل استفاده می‌شود.
- پس از برقراری ارتباط با طرف مقابل، درخواست را دریافت می‌کند.
- درخواست شامل اطلاعات مقصد (آدرس IP و پورت) و نام فایل است.
- با استفاده از تابع handle_incoming_request از کاربر درخواست تایید دریافت می‌شود.
- در صورت تایید، فایل متنی یا تصویر برای طرف مقابل ارسال می‌شود.

تابع start_gui:
- تابعی برای شروع و نمایش رابط کاربری گرافیکی است.
- یک پنجره جدید با استفاده از tkinter ایجاد می‌شود.
- در این پنجره، المان‌های گرافیکی مختلف ایجاد می‌شوند، از جمله برچسب‌ها و دکمه‌ها.
- عملکرد دکمه‌ها با استفاده از توابع مشخص شده تعیین می‌شود.
- همچنین یک نخ جداگانه برای شروع گوش دادن به اتصالات و دریافت درخواست‌ها ایجاد می‌شود.

توابع handle_init, handle_get_all, handle_get, handle_request:
- توابعی برای پردازش عملیات کاربری هستند که در رابط کاربری صدا زده می‌شوند.
- بر اساس عملیات مورد نظر، درخواست مربوطه را به سرور ارسال می‌کنند و نتیجه را نمایش می‌دهند.

تابع handle_incoming_request:
- تابعی برای نمایش پیام درخواست و پرسش از کاربر برای تایید دریافت درخواست است.
- با استفاده از پنجره پیام tkinter، پیام و درخواست را نمایش می‌دهد.
- با استفاده از دکمه‌های پیام، کاربر می‌تواند درخواست را تایید یا رد کند.

تابع main:
- تابع اصلی برنامه است.
- ابتدا یک سرور TCP و یک سرور UDP ایجاد می‌شوند.
- سپس رابط کاربری گرافیکی شروع می‌شود.