```python
# 2024 Â© Idan Hazay helper.py
# Import libraries

from datetime import datetime
import xml.etree.ElementTree as ET
from PyQt6.QtCore import Qt
from PyQt6.QtGui import QFontMetrics, QGuiApplication
import hashlib, os, json, sys, re

class JsonHandle:
    """Handles file upload/download tracking in JSON format."""
    def __init__(self):
        self.uploading_files_json = f"{os.getcwd()}/cache/uploading_files.json"
        self.downloading_files_json = f"{os.getcwd()}/cache/downloading_files.json"

    def get_files_uploading_data(self):
        """Retrieves data of currently uploading files."""
        if os.path.exists(self.uploading_files_json):
            with open(self.uploading_files_json, 'r') as f:
                return json.load(f)

    def get_files_downloading_data(self):
        """Retrieves data of currently downloading files."""
        if os.path.exists(self.downloading_files_json):
            with open(self.downloading_files_json, 'r') as f:
                return json.load(f)

    def update_json(self, upload, file_id, file_path, remove=False, file=None, progress=0):
        """Updates the JSON tracking file with file upload/download details."""
        json_path = self.uploading_files_json if upload else self.downloading_files_json
        if not os.path.exists(os.getcwd() + "\\cache"):
            os.makedirs(os.getcwd() + "\\cache")
        if not os.path.exists(json_path):
            with open(json_path, 'w') as f:
                json.dump({}, f)  # Initialize as an empty dictionary

        with open(json_path, 'r') as f:
            files = json.load(f)

        if remove:  # Remove the file entry if needed
            if file_id in files:
                del files[file_id]
        else:
            if file is None:
                files[file_id] = {"file_path": file_path}
            else:
                files[file_id] = {
                    "file_path": file_path,
                    "size": file.size,
                    "is_view": file.is_view,
                    "file_name": file.file_name,
                    "progress": progress
                }

        with open(json_path, 'w') as f:
            json.dump(files, f, indent=4)

def force_exit():
    """Forces the application to exit."""
    sys.exit()

def control_pressed():
    """Checks if the Control key is pressed."""
    modifiers = QGuiApplication.queryKeyboardModifiers()
    return modifiers & Qt.KeyboardModifier.ControlModifier

def build_req_string(code, values=[]):
    """Builds a request string from a command code and a list of values."""
    return f"{code}|{'|'.join(values)}".encode()

def format_file_size(size):
    """Formats file size into a human-readable format."""
    if size < 10_000:
        return f"{size:,} B"
    elif size < 10_000_000:
        return f"{size / 1_000:,.2f} KB"
    elif size < 10_000_000_001:
        return f"{size / 1_000_000:,.2f} MB"
    elif size < 10_000_000_000_001:
        return f"{size / 1_000_000_000:,.2f} GB"
    else:
        return f"{size / 1_000_000_000_000:,.2f} TB"

def parse_file_size(size_str):
    """Parses a human-readable file size string into bytes."""
    units = {"B": 1, "KB": 1_000, "MB": 1_000_000, "GB": 1_000_000_000, "TB": 1_000_000_000_000}
    unit = size_str.split(" ")[1]
```

```
    size = size_str.split(" ")[0]
    return int(float(size) * units[unit]) if unit in units else 0

def str_to_date(str):
    """Converts a string to a datetime object."""
    return datetime.strptime(str, "%Y-%m-%d %H:%M:%S.%f") if str else datetime.min

def update_ui_size(ui_file, new_width, new_height):
    """Updates the window size in a .ui XML file."""
    tree = ET.parse(ui_file)
    root = tree.getroot()

    for widget in root.findall(".//widget[@class='QMainWindow']"):
        geometry = widget.find("property[@name='geometry']/rect")
        if geometry is not None:
            width_elem = geometry.find("width")
            height_elem = geometry.find("height")
            if width_elem is not None and height_elem is not None:
                width_elem.text = str(new_width)
                height_elem.text = str(new_height)

    tree.write(ui_file, encoding='utf-8', xml_declaration=True)

def truncate_label(label, text):
    """Truncates text with an ellipsis if it exceeds the label width."""
    font_metrics = QFontMetrics(label.font())
    max_width = int(label.width() // 1.9)

    return font_metrics.elidedText(text, Qt.TextElideMode.ElideRight, max_width) if font_metrics.horizontalAdvance(text) >
max_width else text

def update_saved_ip_port(new_ip, new_port):
    """Updates the saved IP and port values in the config file."""
    file_path = f"{os.getcwd()}/modules/config.py"
    with open(file_path, "r", encoding="utf-8") as file:
        content = file.read()

    content = re.sub(r'SAVED_IP\s*=\s*["\'].*?["\']', f'SAVED_IP = \"{new_ip}\"', content)  # Replace SAVED_IP
    content = re.sub(r'SAVED_PORT\s*=\s*\d+', f'SAVED_PORT = {new_port}', content)  # Replace SAVED_PORT

    with open(file_path, "w", encoding="utf-8") as file:
        file.write(content)

file_types = {
    "zip": ["rar"],
    "png": ["jpg", "jpeg", "jfif", "gif", "ico"],
    "mp3": ["wav"],
    "code": ["py", "js", "cs", "c", "cpp", "jar"],
    "txt": ["css"]
}

def format_file_type(type):
    """Maps file extensions to standardized categories."""
    for extension, variations in file_types.items():
        if type in variations or type == extension:
            return extension
    return type

def compute_file_md5(file_path):
    """Computes the MD5 checksum of a file."""
    hash_func = hashlib.new('md5')
    with open(file_path, 'rb') as file:
        while chunk := file.read(8192):
            hash_func.update(chunk)

    return hash_func.hexdigest()
```