```python
# 2024 Â© Idan Hazay
# Import libraries

from PyQt6.QtCore import QThread, pyqtSignal
import time, uuid, traceback, os
from modules.config import *
from modules.limits import Limits

class FileSending:
    """Handles file upload management, including queuing and thread handling."""
    def __init__(self, window):
        self.window = window
        self.active_threads = []
        self.file_queue = []

    def send_files(self, cmd="FILS", file_id=None, resume_file_id=None, location_infile=0):
        """Starts a new file upload thread if none are active."""
        if len(self.active_threads) >= 1:
            return
        try:
            self.window.file_upload_progress.show()
        except:
            pass
        try:
            self.window.stop_button.setEnabled(True)
            self.window.stop_button.show()
        except:
            pass

        thread = FileSenderThread(cmd, file_id, resume_file_id, location_infile, self.window, self.file_queue)
        self.active_threads.append(thread)

        thread.finished.connect(thread.deleteLater)
        thread.finished.connect(lambda: self.active_threads.remove(thread))
        thread.finished.connect(self.window.finish_sending)

        thread.progress.connect(self.window.update_progress)
        thread.progress_reset.connect(self.window.reset_progress)  # Connect progress signal to progress bar
        thread.message.connect(self.window.set_message)
        thread.error.connect(self.window.set_error_message)

        thread.start()

    def resume_files_upload(self, id, progress):
        """Resumes file upload from the last known progress point."""
        uploading_files = self.window.json.get_files_uploading_data()
        for file_id, details in uploading_files.items():  # Iterate through stored uploading files
            if id == file_id:
                file_path = details.get("file_path")
                if not os.path.exists(file_path):
                    continue
                self.file_queue.extend([file_path])  # Re-add file to queue
                self.window.protocol.send_files(resume_file_id=file_id, location_infile=int(progress))
                break


class FileSenderThread(QThread):
    """Handles file upload operations in a separate thread."""
    finished = pyqtSignal()  # Signal when file sending is complete
    error = pyqtSignal(str)  # Signal for error messages
    progress = pyqtSignal(int)  # Signal for updating progress bar
    progress_reset = pyqtSignal(int)
    message = pyqtSignal(str)  # Signal for updating the status message

    def __init__(self, cmd, file_id, resume_file_id, location_infile, window, file_queue):
        super().__init__()
        self.files_uploaded = []
        self.cmd = cmd
        self.file_id = file_id
        self.resume_file_id = resume_file_id
        self.running = True
        self.location_infile = location_infile
        self.window = window
        self.file_queue = file_queue

    def run(self):
        """Runs the file upload process for each file in the queue."""
        try:
            for file_path in self.file_queue:
                start = time.time()
                bytes_sent = 0

                try:
                    self.window.stop_button.setEnabled(True)
                except:
                    pass
```

```
                file_name = self.file_id if self.file_id else file_path.split("/")[-1]  # Extract file name
                file_id = uuid.uuid4().hex
                self.window.uploading_file_id = file_id

                if self.resume_file_id is None:
                    print("start upload:", file_id)
                    start_string = f"{self.cmd}|{file_name}|{self.window.user['cwd']}|{os.path.getsize(file_path)}|
{file_id}"
                    self.window.protocol.send_data(start_string.encode())
                    self.window.json.update_json(True, file_id, file_path)
                else:
                    file_id = self.resume_file_id

                if not os.path.isfile(file_path):
                    self.error.emit("File path was not found")
                    return

                size = os.path.getsize(file_path)
                left = size % CHUNK_SIZE
                sent = self.location_infile
                self.progress.emit(sent)
                self.progress_reset.emit(size)
                self.message.emit(f"{file_name} is being uploaded")

                try:
                    with open(file_path, 'rb') as f:
                        f.seek(self.location_infile)  # Resume from last known position
                        for i in range((size - self.location_infile) // CHUNK_SIZE):
                            if not self.running:
                                break

                            location_infile = f.tell()
                            data = f.read(CHUNK_SIZE)

                            current_time = time.time()
                            elapsed_time = current_time - start

                            if elapsed_time >= 1.0:
                                start = current_time
                                bytes_sent = 0

                            self.window.protocol.send_data(f"FILD|{file_id}|{location_infile}|".encode() + data)
                            bytes_sent += len(data)
                            sent += CHUNK_SIZE

                            self.progress_reset.emit(size)
                            self.message.emit(f"{file_name} is being uploaded")
                            self.progress.emit(sent)  # Update progress bar

                            # Ensure upload speed limit
                            if bytes_sent >= (Limits(self.window.user["subscription_level"]).max_upload_speed - 1) *
1_000_000:
                                time_to_wait = 1.0 - elapsed_time
                                if time_to_wait > 0:
                                    time.sleep(time_to_wait)

                        if not self.running:
                            self.running = True
                            continue

                        location_infile = f.tell()
                        data = f.read(left)
                        if data != b"":
                            self.window.protocol.send_data(f"FILE|{file_id}|{location_infile}|".encode() + data)
                            self.progress_reset.emit(size)
                            self.message.emit(f"{file_name} is being uploaded")
                            self.progress.emit(sent)  # Final progress update

                except:
                    print(traceback.format_exc())
                    return

                finally:
                    self.window.json.update_json(True, file_id, file_path, remove=True)

            if self.file_id is not None:
                os.remove(file_path.split("/")[-1])  # Remove temp files after upload completion
            self.finished.emit()

        except:
            print(traceback.format_exc())
            print(type(self.file_queue))


class File:
    """Represents a file being downloaded or uploaded."""
    def __init__(self, window, save_location, id, size, is_view=False, file_name=None):
```

```
        self.save_location = save_location
        self.id = id
        self.size = size
        self.is_view = is_view
        self.file_name = file_name
        self.start_download()
        self.window = window

    def start_download(self):
        """Prepares a file for download by creating an empty placeholder."""
        if not os.path.exists(self.save_location):
            with open(self.save_location, 'wb') as f:
                f.write(b"\0")  # Create an empty file
                f.flush()

    def add_data(self, data, location_infile):
        """Writes received data to the file at the correct position."""
        try:
            self.window.file_upload_progress.show()
        except:
            pass

        self.window.update_progress(location_infile)
        self.window.reset_progress(self.size)
        self.window.set_message(f"File {self.file_name} is downloading")

        try:
            with open(self.save_location, 'r+b') as f:
                f.seek(location_infile)
                f.write(data)
                f.flush()

                self.window.json.update_json(False, self.id, self.save_location, remove=True)
                self.window.json.update_json(False, self.id, self.save_location, file=self, progress=location_infile)
        except:
            self.uploading = False

    def delete(self):
        """Deletes the downloaded file if it exists."""
        if os.path.exists(self.save_location):
            os.remove(self.save_location)
```