****

Содержание

[Введение 5](#_Toc199800919)

[1 Анализ предметной области 6](#_Toc199800920)

[1.1 Обзор аналогов 6](#_Toc199800921)

[1.2 Постановка задачи 8](#_Toc199800922)

[2 Проектирование программного средства 9](#_Toc199800923)

[2.1 Структура программы 9](#_Toc199800924)

[2.2 Проектирование интерфейса программного средства 9](#_Toc199800925)

[2.3 Проектирование функционала программного средства 11](#_Toc199800926)

[3 Проектирование программного средства 16](#_Toc199800927)

[3.1 Поиск файлов 16](#_Toc199800928)

[3.2 Шифрование данных 17](#_Toc199800929)

[3.2 Оптимизация буфера передачи в зависимости от размера файла 17](#_Toc199800930)

[4 Тестирование программного средства 18](#_Toc199800931)

[5 Руководство пользователя 19](#_Toc199800932)

[5.1 Интерфейс программного средства 19](#_Toc199800933)

[5.2 Управление программным средством 24](#_Toc199800934)

[Заключение 27](#_Toc199800935)

[Список использованных источников 28](#_Toc199800936)

[Приложение А. Текст программы 29](#_Toc199800937)

# Введение

Современный мир информационных технологий невозможно представить без обмена данными между компьютерами и серверами. Одним из ключевых протоколов, обеспечивающих передачу файлов в сети, является FTP (File Transfer Protocol). Этот протокол, разработанный еще в 1971 году, остается актуальным и сегодня благодаря своей простоте, надежности и широкой поддержке различными операционными системами и программным обеспечением. FTP позволяет пользователям загружать, скачивать, удалять и управлять файлами на удаленных серверах, что делает его незаменимым инструментом для системных администраторов, веб-разработчиков и обычных пользователей.

Актуальность темы курсовой работы обусловлена необходимостью понимания принципов работы FTP и создания собственного FTP-клиента, который может быть использован для решения практических задач. Несмотря на наличие множества готовых решений, таких как FileZilla, WinSCP и других, разработка собственного клиента позволяет глубже изучить работу протокола, а также адаптировать программу под конкретные требования пользователя. Кроме того, это отличная возможность закрепить знания в области сетевого программирования и разработки приложений.

Целью данной работы является разработка FTP-клиента, который обеспечивает базовые функции для работы с FTP-серверами, такие как подключение к серверу, просмотр содержимого директорий, загрузка и скачивание файлов, а также управление файлами на удаленном сервере. В процессе разработки будут рассмотрены основные аспекты работы протокола FTP, включая его команды, режимы передачи данных и вопросы безопасности.

Таким образом, данная курсовая работа не только позволит углубить знания в области сетевых технологий, но и предоставит практический опыт разработки приложений для работы с сетевыми протоколами. Это важный шаг в профессиональном становлении будущего специалиста в области информационных технологий.

# 1 Анализ предметной области

## 1.1 Обзор аналогов

FileZilla — это один из самых популярных FTP-клиентов с открытым исходным кодом, который поддерживает работу с протоколами FTP, SFTP и FTPS. Он доступен для операционных систем Windows, macOS и Linux, что делает его универсальным инструментом для работы с файлами на удаленных серверах.

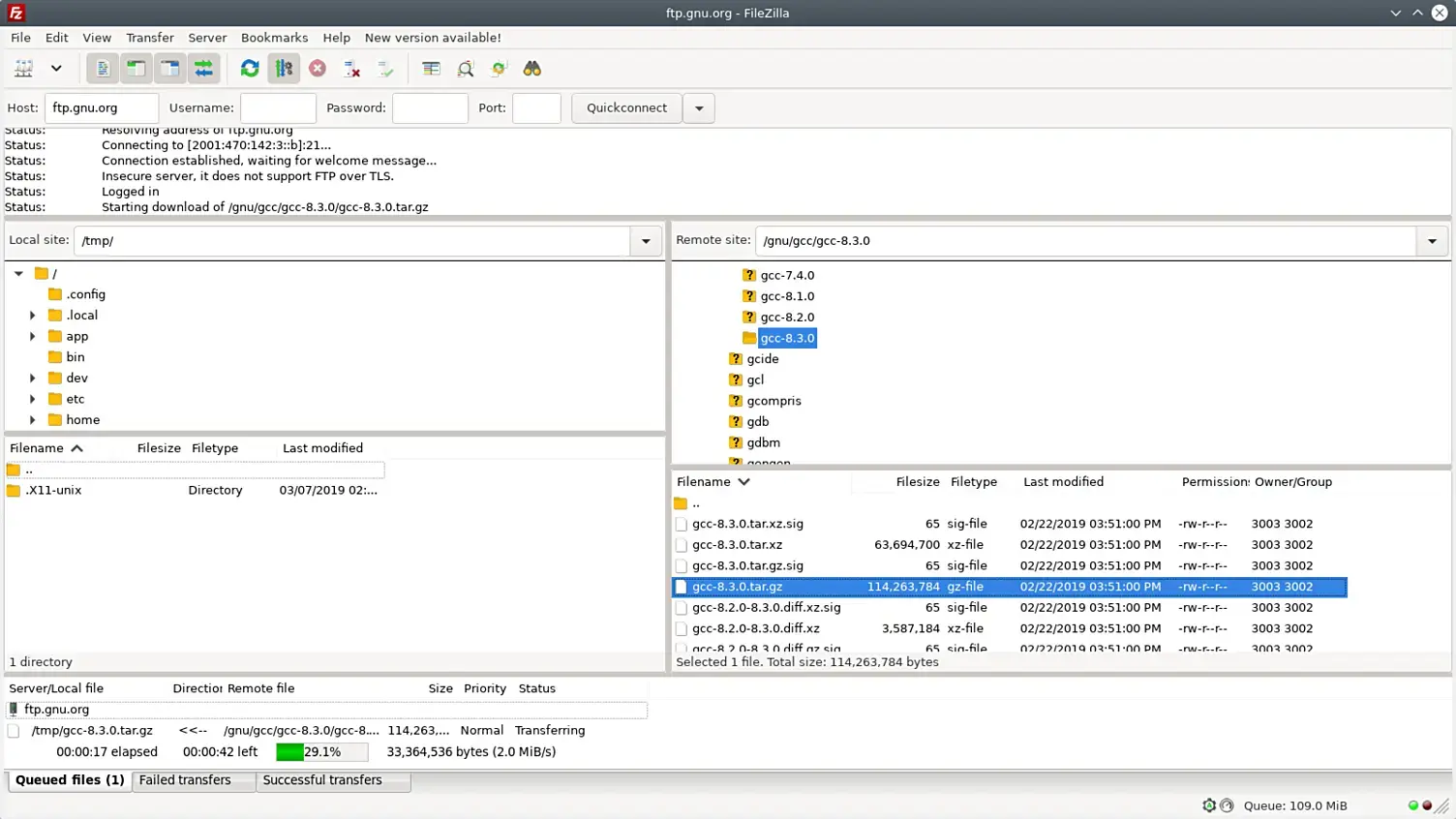


Рисунок 1.1 – Программное средство «FileZilla»

Одним из главных преимуществ FileZilla является его кроссплатформенность, которая позволяет пользователям работать с клиентом на различных операционных системах без необходимости поиска альтернативных решений. Кроме того, FileZilla поддерживает несколько протоколов передачи данных, включая SFTP и FTPS, что обеспечивает безопасную передачу файлов. Интерфейс программы интуитивно понятен и разделен на две части: локальные и удаленные файлы, что упрощает навигацию и управление. Высокая производительность FileZilla позволяет эффективно работать с большими объемами данных, а функция возобновления передачи файлов после разрыва соединения делает его надежным инструментом. Немаловажным преимуществом является и то, что программа распространяется бесплатно, что делает ее доступной для широкого круга пользователей.

Однако FileZilla имеет и свои недостатки. Одним из них является отсутствие портативной версии, что может быть неудобно для пользователей, которые предпочитают работать с программами без их установки на компьютер. В прошлом FileZilla также подвергалась критике за проблемы с безопасностью, в частности за хранение паролей в открытом виде, что создавало риски для пользователей. Кроме того, интерфейс программы, несмотря на свою функциональность, может показаться перегруженным и сложным для новичков, что затрудняет ее использование для неподготовленных пользователей.

WinSCP — это бесплатный FTP-клиент для Windows, который поддерживает протоколы SFTP, SCP, FTP и WebDAV. Он ориентирован на безопасную передачу данных и часто используется системными администраторами и разработчиками. Одним из ключевых преимуществ WinSCP является его высокая безопасность, так как клиент поддерживает шифрование данных через SFTP и SCP, что делает его надежным инструментом для работы с конфиденциальной информацией. WinSCP также хорошо интегрируется с Windows, поддерживая drag-and-drop и взаимодействие с проводником, что значительно упрощает работу с файлами. Еще одним преимуществом является поддержка сценариев, которая позволяет автоматизировать рутинные задачи, что особенно полезно для системных администраторов. Кроме того, WinSCP доступен в портативной версии, что позволяет использовать его без установки на компьютер, например, с USB-накопителя. Как и FileZilla, WinSCP распространяется бесплатно, что делает его доступным для всех пользователей.

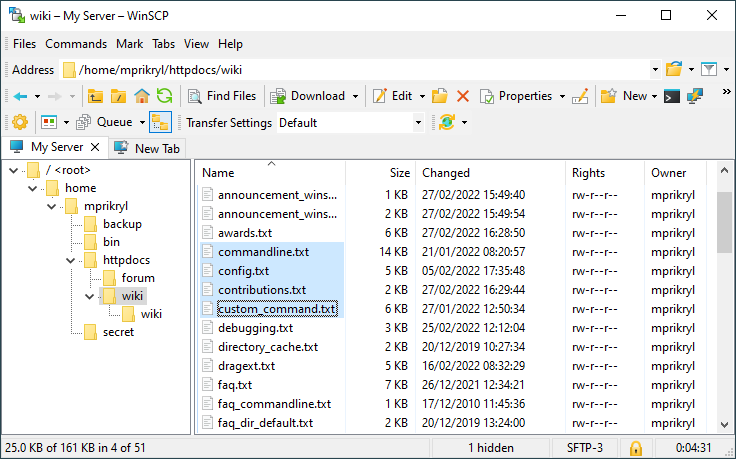


Рисунок 1.2 – Программное средство «WinSCP»

Одним из ключевых преимуществ WinSCP является его высокая безопасность, так как клиент поддерживает шифрование данных через SFTP и SCP, что делает его надежным инструментом для работы с конфиденциальной информацией. WinSCP также хорошо интегрируется с Windows, поддерживая drag-and-drop и взаимодействие с проводником, что значительно упрощает работу с файлами. Еще одним преимуществом является поддержка сценариев, которая позволяет автоматизировать рутинные задачи, что особенно полезно для системных администраторов. Кроме того, WinSCP доступен в портативной версии, что позволяет использовать его без установки на компьютер, например, с USB-накопителя. Как и FileZilla, WinSCP распространяется бесплатно, что делает его доступным для всех пользователей.

Однако WinSCP имеет и свои недостатки. Одним из главных ограничений является его доступность только для Windows, что делает его непригодным для пользователей других операционных систем, таких как macOS или Linux. Интерфейс WinSCP, хотя и функциональный, может показаться менее современным и удобным по сравнению с FileZilla, что может отпугнуть некоторых пользователей. Кроме того, WinSCP не поддерживает протокол FTPS, что может быть существенным недостатком для тех, кому требуется именно этот протокол для работы.

## 1.2 Постановка задачи

В рамках данного курсового проекта планируется разработка программного средства «FTP-клиент», предназначенного для работы с FTP-серверами. Программа будет предоставлять пользователю удобный интерфейс для подключения к удаленным серверам, управления файлами и передачи данных.

В процессе реализации будет разработан базовый функционал для работы с протоколом FTP, включая подключение к серверу, просмотр содержимого директорий, загрузку и скачивание файлов, а также управление файлами на удаленном сервере. Особое внимание будет уделено простоте интерфейса и удобству использования, чтобы программа была доступна как для опытных пользователей, так и для новичков.

В FTP-клиенте планируется реализовать следующие функции:

* подключение к FTP-серверу с использованием логина и пароля;
* просмотр списка файлов и директорий на удаленном сервере;
* загрузка файлов с сервера на локальный компьютер;
* выгрузка файлов с локального компьютера на сервер;
* удаление и переименование файлов на удаленном сервере;
* обработка ошибок подключения и передачи данных для повышения надежности работы программы.

Для разработки программного средства будет использоваться язык программирования Python с использованием библиотеки ftplib для работы с FTP-протоколом и библиотеки tkinter для создания графического интерфейса. Это позволит создать кроссплатформенное приложение, которое будет работать на различных операционных системах.

Кроме того, в процессе разработки будут учтены недостатки существующих решений, такие как сложность интерфейса и проблемы с безопасностью, чтобы создать более удобное и надежное программное средство. Результатом работы станет FTP-клиент, который сможет успешно конкурировать с существующими аналогами и будет полезен для широкого круга пользователей.

# 2 Проектирование программного средства

## 2.1 Структура программы

Программа спроектирована с использованием модульной слоистой архитектуры, обеспечивающей четкое разделение функциональных компонентов. Основные слои:

* Core – ядро приложения, содержащее бизнес-логику программы, включая модуль работы с FTP-протоколом, управление настройками программы, обработку файловых операций.
* GUI – графический интерфейс пользователя, обеспечивающий взаимодействие с функционалом Core.
* Utils – вспомогательные модули, отвечающие за шифрование данных, утилиты для работы с файлами и вспомогательные инструменты.

## 2.2 Проектирование интерфейса программного средства

Разрабатываемый графический интерфейс пользователя для FTP-клиента будет реализован на базе библиотеки Tkinter, что обеспечивает кроссплатформенность и простоту интеграции с Python-логикой приложения.

**2.2.1** Главное окно приложения

Главное окно приложения разделено на логические блоки, каждый из которых отвечает за определенный аспект работы с FTP-сервером. Ниже приведена структура с краткими пояснениями:

* параметры подключения – панель, предназначенная для настройки и установки соединения с FTP-сервером;
* мониторинг соединения – блок для визуализации состояния сети и диагностики проблем подключения;
* панель поиска – элементы интерфейса, необходимые для быстрого поиска файлов на локальном и удаленном хранилищах;
* рабочая панель – таблицы, предназначенные для отображения, навигации по файловой системе и управление файлами;
* статус-строка – блок для отображения текущего состояния приложения и выполнения фоновых задач.

Интерфейс главного окна приложения представлен на рисунке 2.1.

Изображение выглядит как снимок экрана, текст, чек, линия

Автоматически созданное описание

Рисунок 2.1 – Интерфейс главного окна

**2.2.2** Окно настроек

Окно настроек FTP-клиента будет организовано в виде вкладок для удобной навигации и разделения параметров по категориям. Каждая вкладка отвечает за определенный аспект работы программы:

* вкладка «Общие» будет отвечать за базовые конфигурации путей и за визуальное отображение файлов;
* вкладка «Подтверждения» послужит защитой от случайных действий пользователя;
* вкладка «Производительность» необходима для настройки скорости передачи данных и эффективности использования ресурсов.

Внешний вид окна настроек показан на рисунке 2.2.

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

Рисунок 2.2 – Окно настроек приложения

## 2.3 Проектирование функционала программного средства

Проектирование алгоритмов работы и взаимодействия компонентов является основой работы FTP-клиента. Приложение должно предоставлять пользователю следующий минимальный функционал:

* установление FTP-соединения с сервером;
* получение списка файлов и директорий с FTP-сервера;
* передача файлов между клиентом и сервером;
* мониторинг состояния соединения.

В данном разделе будут рассмотрены ключевые алгоритмы, обеспечивающие надежную работу программного средства. Особое внимание уделено механизмам обработки сетевых взаимодействий, управления передачей данных и обеспечения целостности файловых операций. Реализация этих алгоритмов позволяет создать надежный и удобный инструмент для работы с FTP-серверами.

**2.3.1** Алгоритм установки соединения

Алгоритм подключения к FTP-серверу представляет собой последовательность операций, обеспечивающих установку защищенного соединения и аутентификацию пользователя. Процедура начинается с проверки существующего соединения – при его обнаружении выполняется корректное завершение текущей сессии. Затем создается новый экземпляр FTP-клиента, после чего осуществляется попытка подключения к указанному серверу через заданный порт.

После успешного соединения с сервером выполняется процедура аутентификации с использованием предоставленных учетных данных. В случае успешной проверки подлинности устанавливается параметр кодировки соединения согласно конфигурации приложения. Все параметры подключения сохраняются для возможного последующего восстановления соединения.

Результатом выполнения алгоритма является формирование статуса операции, включающего логический индикатор успешности выполнения и текстовое сообщение с детализацией результата. Особое внимание уделяется обеспечению потокобезопасности операций за счет использования механизма блокировки.

Блок-схема представленного алгоритма отражена на рисунке 2.3.

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание

Рисунок 2.3 – Блок-схема алгоритма установления FTP-соединения с сервером

**2.3.2** Алгоритм получения списка файлов и директорий с FTP-сервера

Алгоритм получения списка файлов реализует комплексный подход к обработке содержимого FTP-сервера. Процедура начинается с проверки наличия активного соединения, что обеспечивает корректность выполнения последующих операций.

Для директорий должен выполняться дополнительный анализ содержимого: временно изменяется текущая рабочая директория, определяется количество вложенных элементов, после чего происходит возврат в исходное положение. Это позволяет предоставить пользователю информацию о наполнении каждой директории.

Особое внимание будет уделено обработке временных меток – выполняется парсинг строкового представления даты в унифицированный формат. Алгоритм включает комплексную обработку исключительных ситуаций на каждом этапе, что обеспечивает стабильную работу даже при частично некорректных данных от сервера.

Результатом выполнения должен быть структурированный список кортежей, содержащий для каждого объекта: наименование, размер,тип, дата последнего изменения.

Блок-схема алгоритма представлена на рисунке 2.4.

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание

Рисунок 2.4 – Блок-схема алгоритма получения списка файлов и директорий с FTP-сервера

**2.3.3** Алгоритм передачи файлов между клиентом и сервером

Алгоритм реализует надежную процедуру скачивания файлов с FTP-сервера с контролем целостности данных. Особое внимание уделено обработке больших файлов через механизм буферизированной передачи, что позволяет минимизировать использование оперативной памяти. В случае обнаружения расхождения размеров файлов автоматически выполняется откат операции. Алгоритм загрузки файлов с сервера аналогичен скачиванию, главное отличие в FTP-командах: для скачивания используется команда RETR с последовательной записью в локальный файл, а для загрузки применяется команда STOR с потоковым чтением исходного файла. Блок-схема алгоритма загрузки файлов с сервера представлена ниже.

Изображение выглядит как текст, диаграмма, План, Технический чертеж

Автоматически созданное описание

Рисунок 2.5 – Блок-схема алгоритма загрузки файлов с сервера

**2.3.4** Алгоритм мониторинга состояния соединения

Алгоритм мониторинга состояния соединения представляет собой фоновый процесс, непрерывно отслеживающий доступность FTP-сервера и качество подключения. Процедура начинается с инициализации параметров подключения, включая адрес хоста и порт назначения. После запуска система создает отдельный поток выполнения, который работает параллельно основному процессу приложения, не блокируя пользовательский интерфейс.

Основной цикл мониторинга функционирует по следующему принципу: на каждом шаге итерации фиксируется точное время начала проверки, после чего осуществляется попытка установить TCP-соединение с целевым сервером. Для этого создается новый сокет с предустановленным таймаутом подключения, что позволяет избежать бесконечного ожидания при недоступности сервера. В случае успешного соединения система рассчитывает задержку передачи данных как разницу между временем отправки запроса и получения ответа, преобразуя этот показатель в миллисекунды для удобства восприятия. Одновременно фиксируется нулевой процент потерь пакетов, свидетельствующий о стабильном канале связи.

При неудачной попытке подключения или возникновении исключительной ситуации алгоритм автоматически устанавливает максимальное значение потерь пакетов и обнуляет показатель задержки, что служит четким индикатором проблем с соединением. Независимо от результата проверки, система сохраняет временную метку последнего контроля, обеспечивая возможность анализа истории подключений. Между последовательными проверками выдерживается пауза в одну секунду, что позволяет поддерживать баланс между оперативностью обнаружения проблем и нагрузкой на систему. Процесс продолжается циклически до явного указания на необходимость прекращения мониторинга, после чего фоновый поток корректно завершает свою работу, освобождая все занятые ресурсы.

Блок-схема алгоритма представлена на рисунке 2.6.

Изображение выглядит как текст, чек, диаграмма, снимок экрана

Автоматически созданное описание

Рисунок 2.6 – Блок-схема алгоритма мониторинга состояния соединения

# 3 Проектирование программного средства

## 3.1 Поиск файлов

Алгоритм умного поиска файлов реализует механизм фильтрации локальных и удаленных файлов с учетом пользовательских настроек. При вызове функции поиска система анализирует входную строку поиска и применяет её к текущему содержимому директорий. Поиск может осуществляться как в локальной, так и в удаленной файловой системе, при этом учитывается регистр символов и возможность поиска внутри папок. Код алгоритма представлен ниже.

def \_on\_search(self, text: str, scope: str, case\_sensitive: bool, search\_in\_folders: bool):

if not text:

self.\_refresh\_lists()

return

def matches\_search(name: str) -> bool:

// Реализация умного сравнения с учетом регистра

if not case\_sensitive:

return text.lower() in name.lower()

return text in name

if scope in ["local", "both"]:

try:

items = []

current\_dir = self.settings.get('default\_local\_dir')

for item in os.listdir(current\_dir):

try:

path = os.path.join(current\_dir, item)

is\_dir = os.path.isdir(path)

// Проверка соответствия условиям поиска

if matches\_search(item) or (is\_dir and search\_in\_folders):

stat = os.stat(path)

// Специальная обработка для директорий

if is\_dir:

try:

size = f"{len(os.listdir(path))} элем."

except:

size = "Нет доступа"

else:

size = humanize.naturalsize(stat.st\_size)

// Форматирование времени модификации

modified = datetime.fromtimestamp(stat.st\_mtime).strftime(

self.settings.get('date\_format', "%Y-%m-%d %H:%M"))

items.append({

'name': item,

'size': size,

'type': "Папка" if is\_dir else "Файл",

'modified': modified

})

except Exception:

continue

// Применение фильтров и сортировки

items = filter\_hidden\_files(items, self.settings.get('show\_hidden\_files'))

items = sort\_items(items, self.settings.get('sort\_folders\_first'))

## 3.2 Шифрование данных

Алгоритм шифрования паролей реализует механизм безопасного хранения конфиденциальных данных для закладок FTP-подключений. Процесс начинается с генерации случайной криптографической соли длиной 16 байт, которая обеспечивает уникальность каждого зашифрованного значения даже для одинаковых паролей. Затем происходит формирование криптографического ключа с использованием алгоритма PBKDF2 (Password-Based Key Derivation Function 2), где в качестве основы используется мастер-ключ приложения. Для усиления криптостойкости применяется 100000 итераций хеширования с использованием алгоритма SHA-256. Полученный ключ кодируется в base64-формат для дальнейшего использования в алгоритме симметричного шифрования Fernet. После шифрования исходного текста пароля, результат объединяется с солью и кодируется в base64-строку для безопасного хранения. Такой подход обеспечивает надежную защиту сохраненных паролей от несанкционированного доступа, при этом позволяя восстанавливать их при наличии корректного мастер-ключа. Код алгоритма представлен ниже.

def encrypt(self, text: str) -> str:

// Генерация случайной соли для каждого пароля

salt = os.urandom(16)

// Создание ключа на основе мастер-ключа и соли

kdf = PBKDF2HMAC(

algorithm=hashes.SHA256(),

length=32,

salt=salt,

iterations=100000,

)

key = base64.urlsafe\_b64encode(kdf.derive(self.master\_key))

// Шифрование пароля с использованием Fernet

f = Fernet(key)

encrypted = f.encrypt(text.encode())

// Комбинируем соль и зашифрованные данные для хранения

return base64.urlsafe\_b64encode(salt + encrypted).decode()

## 3.2 Оптимизация буфера передачи в зависимости от размера файла

Алгоритм оптимизации буфера передачи реализует механизм динамической настройки размера блока данных при операциях передачи файлов через FTP-соединение. В основе алгоритма лежит принцип баланса между эффективностью использования памяти и скоростью передачи данных. При работе с файлами размером менее 1 мегабайта используется минимальный буфер в 8 килобайт, что оптимально для небольших файлов и экономит ресурсы системы. Для файлов среднего размера, от 1 до 10 мегабайт, размер буфера увеличивается до 32 килобайт, обеспечивая более эффективную передачу. При работе с крупными файлами, превышающими 10 мегабайт, применяется максимальный размер буфера в 64 килобайта, что позволяет достичь оптимальной скорости передачи для больших объемов данных. Такой адаптивный подход обеспечивает эффективное использование сетевых ресурсов и оперативной памяти, учитывая особенности передачи файлов различного размера. Код алгоритма представлен ниже.

def \_get\_optimal\_buffer\_size(self, file\_size: int) -> int:

// Динамическая настройка размера буфера в зависимости от размера файла

if file\_size < 1024 \* 1024: // < 1MB

return 8192 // 8KB для маленьких файлов

elif file\_size < 10 \* 1024 \* 1024: // < 10MB

return 32768 // 32KB для средних файлов

else: // >= 10MB

return 65536 // 64KB для больших файлов

# 4 Тестирование программного средства

При разработке FTP-клиента было обнаружено несколько проблем, связанных с управлением состоянием соединения и обработкой потери связи с сервером.

В процессе тестирования был обнаружен ряд проблем при внезапном разрыве соединения с FTP-сервером:

1. Зависание интерфейса при попытке выполнения операций с файлами после потери соединения.

2. Отсутствие автоматического обнаружения разрыва соединения.

3. Некорректное состояние интерфейса после потери связи.

Основная проблема заключалась в том, что при внезапной потере соединения программа продолжала считать себя подключенной к серверу, что приводило к зависанию при попытке выполнить любую операцию с удаленными файлами. Кроме того, пользователь не получал своевременного уведомления о потере связи.

Для решения этой проблемы был реализован механизм мониторинга соединения в классе «ConnectionMonitor» и внедрена система обработки состояния соединения.

В коде это выглядит следующим образом:

1. Создан отдельный поток для мониторинга состояния соединения:

def start\_connection\_monitor(self, callback):

self.monitor\_thread = Thread(target=self.\_monitor\_connection, args=(callback,))

self.monitor\_thread.daemon = True

self.monitor\_thread.start()

2. Реализована функция периодической проверки соединения:

def \_monitor\_connection(self, callback):

while self.ftp:

try:

self.ftp.voidcmd("NOOP")

except:

callback()

break

time.sleep(5)

3. Добавлен механизм асинхронного обновления интерфейса при потере соединения:

def \_on\_connection\_lost(self):

self.schedule\_update(lambda: [

self.status\_bar.set\_status("Связь с сервером потеряна", error=True),

messagebox.showwarning("Соединение", "Связь с сервером потеряна"),

self.\_disconnect()

])

Внедрение этого решения позволило:

* автоматически обнаруживать потерю соединения;
* корректно обрабатывать разрыв связи без зависания интерфейса;
* своевременно уведомлять пользователя о проблемах с соединением;
* автоматически переводить интерфейс в корректное состояние при потере связи.

Дополнительно была реализована система мониторинга качества соединения через класс «ConnectionStatsPanel», которая отслеживает задержки и потери пакетов, что позволяет пользователю видеть состояние соединения в реальном времени и предвидеть возможные проблемы со связью.

Это решение значительно повысило надежность и удобство использования приложения, особенно при работе с нестабильным сетевым соединением.

# 5 Руководство пользователя

## 5.1 Интерфейс программного средства

**5.1.1** Главное окно

Главное окно приложения разделено на логические блоки, каждый из которых отвечает за определенный аспект работы с FTP-сервером.

Панель подключения располагается в верхней части интерфейса и обеспечивает управление соединением с FTP-сервером. Для корректной установки соединения предусмотрены поля «сервер», «порт», «пользователь», «пароль». Также для удобства представлены элементы управления:

* кнопка с изображением глаза переключает режим отображения символов пароля;
* кнопка «Подключиться» инициирует соединение с сервером. При успешном подключении кнопка изменяет статус на «Отключиться».

Блок статистики располагается непосредственно под панелью подключения и отображает следующие метрики в реальном времени:

* задержка в миллисекундах – время отклика сервера;
* потери пакетов в процентах – процент потерянных данных при передаче;
* последняя проверка – временная метка последнего обновления статистики.

Панель поиска расположена в центральной части интерфейса и обеспечивает фильтрацию контента по следующим критериям:

* область поиска – переключатель между поиском в локальном, удаленном хранилищах либо вместе одновременно;
* расширенные параметры – параметры для активации чувствительности к регистру символов и включения рекурсивного поиск во вложенных директориях.

Панель инструментов управления расположена между областями локальных и удаленных файлов. Предусмотрены следующие операции:

* наверх: переход в родительский каталог для активной файловой области;
* обновить: повторная загрузка содержимого текущей директории.
* папка: создание новой директории на сервере или локальном хранилище.
* загрузить: передача выделенных файлов на FTP-сервер;
* скачать: загрузка выделенных файлов с сервера;
* удалить: удаление выделенных объектов с запросом подтверждения.

Далее располагается рабочая область. Секция отображает содержимое локального и удаленного хранилищ в табличном формате, где каждый столбец отображает метаданные файла, такие как: наименование, размер, тип, дата последнего изменения.

В нижней части интерфейса расположена информационная панель, которая включает в себя:

* текстовый индикатор: отображает текущее состояние системы;
* прогресс-бар: визуализирует выполнение текущей операции;
* цифровой индикатор: отображает процент выполнения задачи;

Внешний вид главного окна представлен на рисунке 5.1.

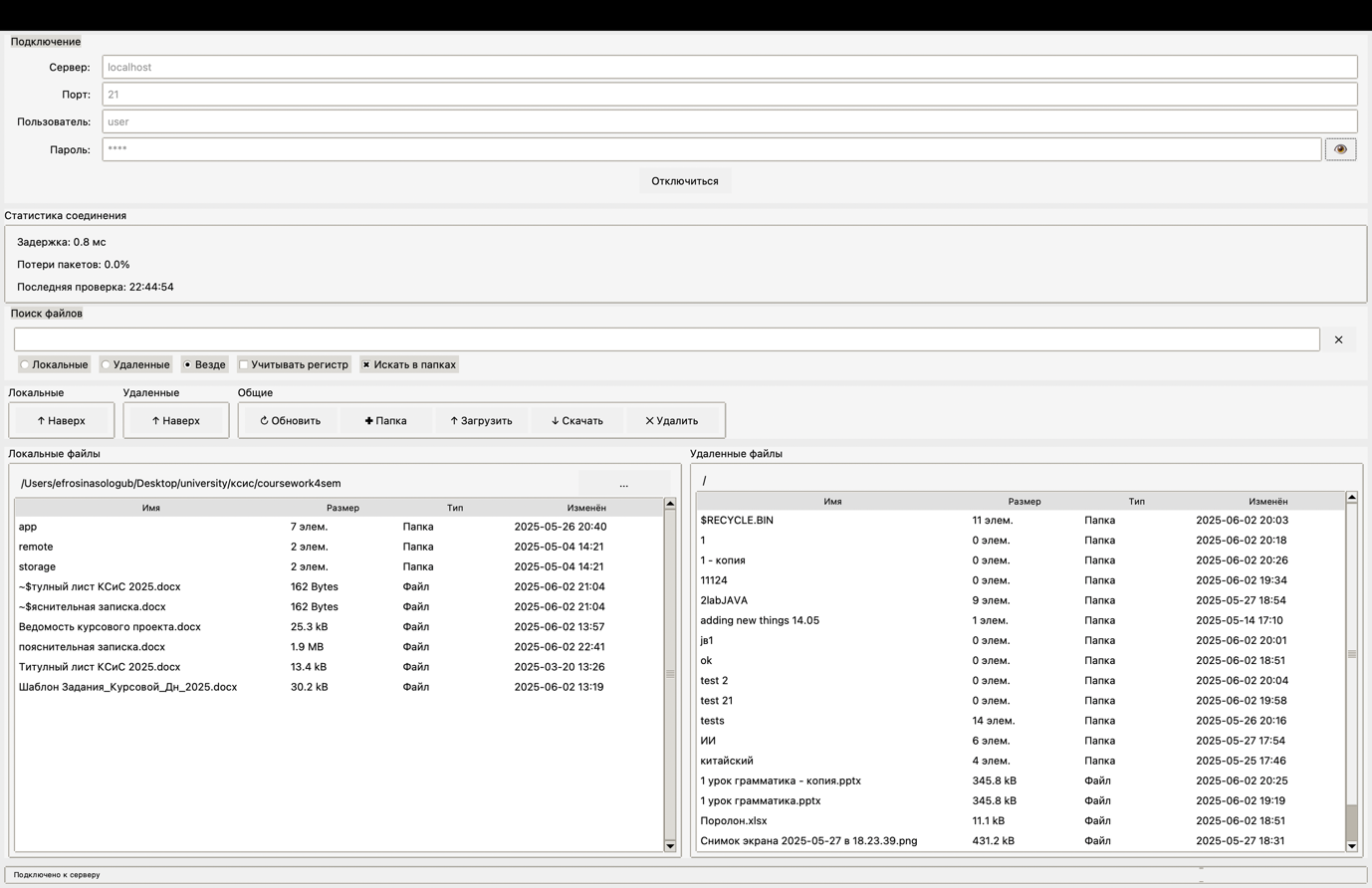


Рисунок 5.1 – Главное окно приложения

**5.1.2** История подключений

Интерфейс истории подключений предоставляет пользователю список всех предыдущих соединений с FTP-серверами.

Таблица истории подключений содержит следующие колонки:

* сервер – доменное имя или IP-адрес FTP-сервера;
* порт – номер порта, использовавшегося для подключения;
* пользователь – имя учетной записи для аутентификации;
* дата – временная метка последнего подключения, формат: ГГГГ-ММ-ДД ЧЧ:ММ.

Интерфейс истории подключений представлен на рисунке ниже.

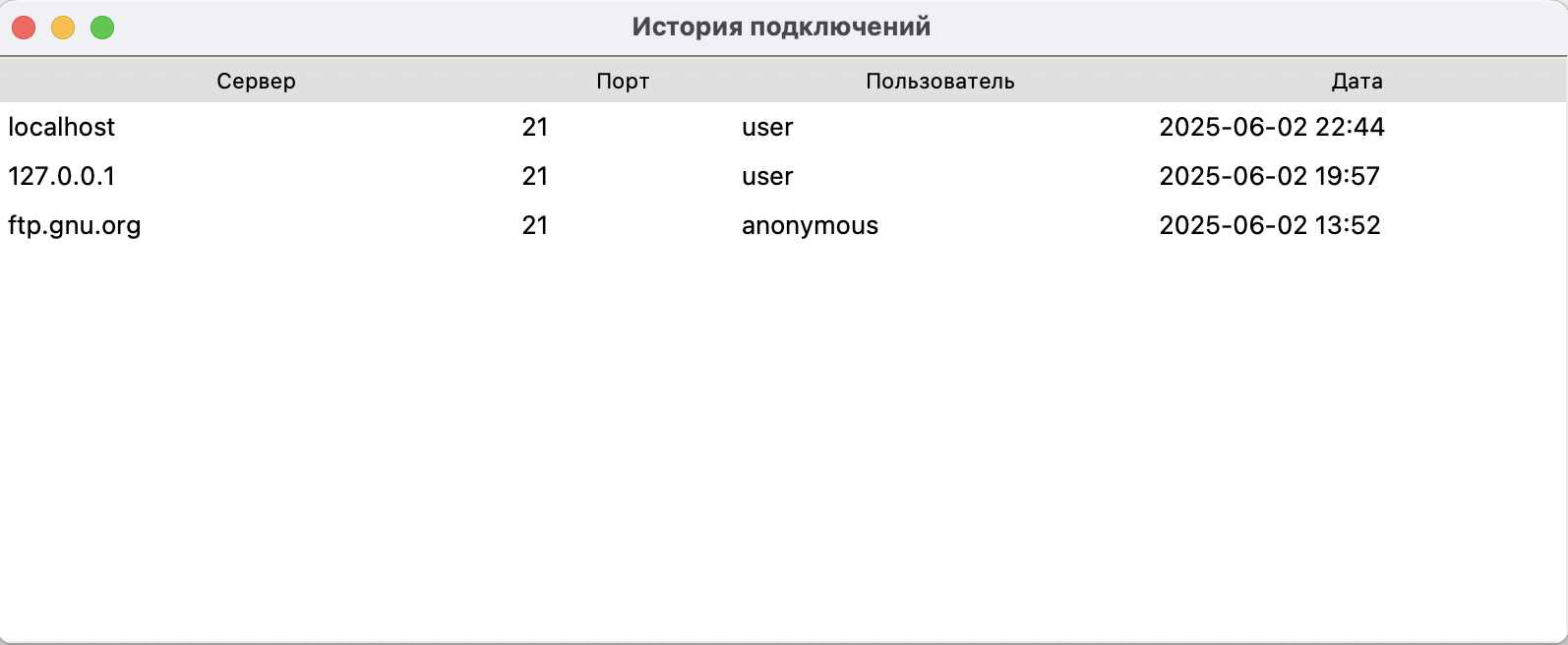


Рисунок 5.2 – Окно истории подключений

**5.1.3** Закладки

Интерфейс закладок предоставляет пользователю список сохраненных серверов и параметры для быстрого подключения к ним. Таблица закладок содержит следующие колонки:

* название – персонализированное имя сервера, введенное пользователем;
* сервер – доменное имя или IP-адрес FTP-сервера;
* порт – номер порта, использовавшегося для подключения;
* пользователь – имя учетной записи для аутентификации.

Также в зашифрованном виде храниться пароль для каждого пользователя, но для безопасности данных на экран данная информация не выводится.

Внешний вид закладок представлен на рисунке ниже.

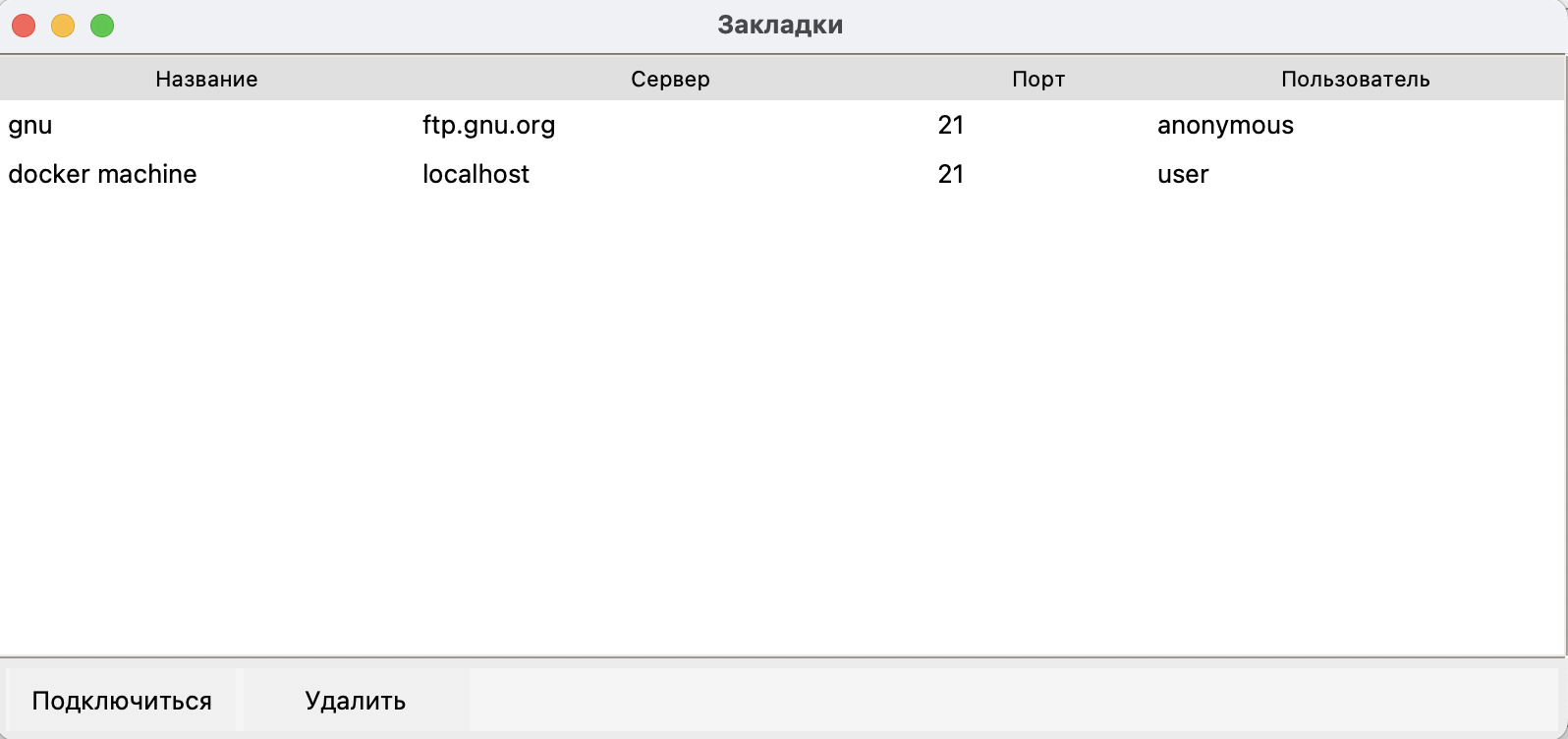


Рисунок 5.3 – Закладки

**5.1.4** Настройки

Окно настроек организовано в виде вкладок:

* общие – базовые параметры программы;
* подтверждения – настройки диалоговых окон;
* производительность – параметры работы системы.

Навигация между вкладками осуществляется кликом по соответствующим заголовкам. В нижней части окна расположены кнопки управления: отмена – закрывает окно без сохранения изменений, сохранить – применяет новые настройки.

Вкладка «Общие» содержит основные параметры программы:

* поле «Локальная директория» отображает текущий путь к рабочей папке по умолчанию;
* кнопка «Обзор» открывает диалог выбора директории;
* чекбокс «Автоматическое переподключение» отвечает за включение и выключение функции автоматического восстановления соединения;
* поле «Количество попыток» определяет максимальное число попыток переподключения;
* чекбокс «Показывать папки первыми» отвечает за сортировку папок в начале списка файлов;
* чекбокс «Показывать скрытые файлы» ответственный за отображение файлов и папок с атрибутом «скрытый».

Вкладка «Подтверждения» – управление диалогами подтверждения:

* чекбокс «Подтверждать удаление» отвечает за отображение предупреждения при удалении файлов;
* чекбокс «Подтверждать перезапись» ответственный за показ предупреждения при перезаписи существующих файлов.

Вкладка «Производительность» отвечает за параметры оптимизации работы:

* размер буфера в байтах определяет объем данных, передаваемых за одну операцию, влияет на скорость передачи файлов;
* Время жизни кэша в секундах указывает продолжительность хранения кэшированных данных в приложении, уменьшает количество запросов к серверу.

Интерфейс настроек представлен на рисунке 5.4.

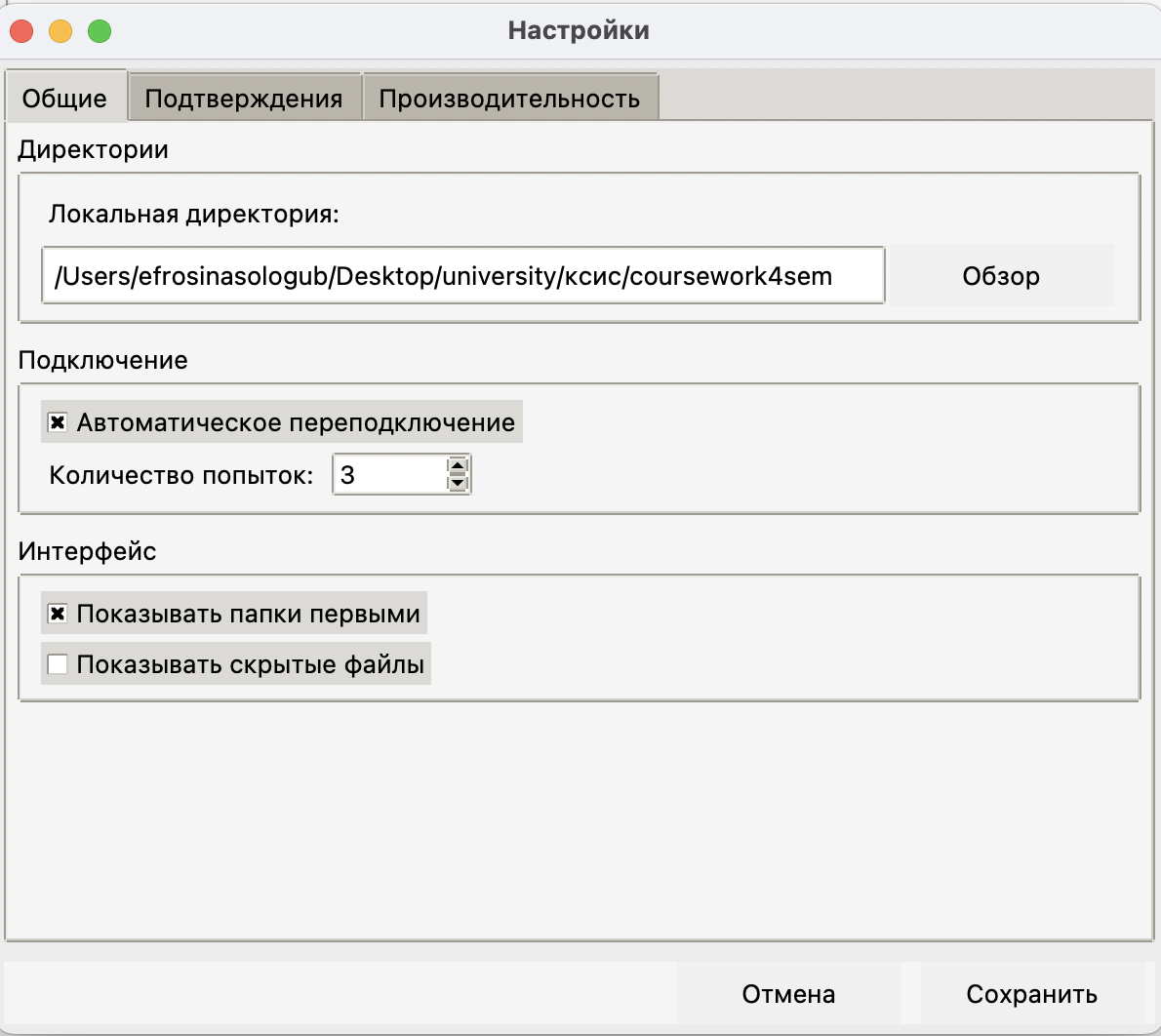


Рисунок 5.4 – Настройки

## 5.2 Управление программным средством

Программное средство представляет собой клиент для работы с FTP-серверами и обеспечивает полнофункциональный интерфейс для выполнения файловых операций. Далее представлено описание основных элементов управления и принципов работы с программой.

**5.2.1** Подключение к серверу

Для установления соединения с FTP-сервером пользователю предоставляется панель подключения, расположенная в верхней части интерфейса. В соответствующие поля необходимо внести следующие данные: адрес сервера, порт подключения, имя пользователя, пароль.

Подключение может быть осуществлено как через основную панель, так и посредством функции «Быстрое подключение», доступной в меню «Подключение» или по сочетанию клавиш Ctrl+Q или command+Q для macOS.

**5.2.2** Навигация по файловой системе

Интерфейс программы разделен на две основные панели:

* левая панель отображает локальную файловую систему;
* правая панель представляет содержимое удаленного FTP-сервера.

Для перемещения по директориям предусмотрены следующие возможности:

* двойной щелчок мыши по папке;
* кнопка "Наверх" для перехода в родительскую директорию;
* строка пути с возможностью прямого ввода или выбора директории.

**5.2.3** Файловые операции

Программное средство поддерживает следующие операции с файлами и директориями:

* Создание новых элементов, а именно создание папок с помощью кнопки на главном окне, через кнопку при вызове контекстного меню путем нажатия правой клавиши мыши, через кнопку «Создать папку» в главном меню приложения в разделе «Операции» или с помощью горячих клавиш Ctrl+N или command+N для macOS.
* Копирование и перемещение файлов: загрузка файлов на сервер с помощью кнопки «Загрузить», через аналогичную кнопку в главном меню приложения в разделе «Операции», через контекстное меню или с помощью горячих клавиш Ctrl+U или command+U для macOS; скачивание файлов с сервера с помощью кнопки «Скачать», через аналогичную кнопку в главном меню приложения в разделе «Операции», через контекстное меню или с помощью горячих клавиш Ctrl+D или command+D для macOS.
* Удаление: кнопка на экране «Удалить», либо при нажатии правой кнопки мыши через контекстное меню, либо кнопкой Delete на клавиатуре.
* Переименование осуществляется через соответствующую кнопку в контекстном меню.

**5.2.4** Дополнительные функции

Поиск:

* поиск файлов в текущей директории с помощью поисковой строки;
* учет регистра при нажатии на поле «Учитывать регистр» под поисковой строкой;
* поиск в подпапках при нажатии на поле «Искать в папках» правее.

Управление подключениями:

* автоматическое сохранение истории подключений
* работа с закладками. Для создания необходимо воспользоваться вкладкой главного меню «Подключение», «Добавить в закладки» или горячими клавишами Ctrl+B или command+B для macOS;

**5.2.5** Горячие клавиши

Для повышения эффективности работы реализованы следующие сочетания клавиш:

* F5,Ctrl+R, command+R – обновление списков файлов;
* Ctrl+Q, command+Q – выход;
* Ctrl+H, command+H – история подключений;
* Ctrl+B, command+B – добавление текущего подключения в закладки;
* Ctrl+N, command+N – создание новой папки;
* Ctrl+U, command+U – загрузка файлов на сервер;
* Ctrl+D, command+D – скачивание файлов с сервера;
* Escape – выход из полноэкранного режима;
* Delete – удаление выбранного файла;
* Enter – подтверждение действия.

**5.2.6** Контроль операций

Все выполняемые операции сопровождаются индикацией прогресса, отображением текущего статуса, системой уведомлений об ошибках, запросами подтверждения для критических операций.

**5.2.7** Завершение работы

При завершении работы программа запрашивает подтверждение выхода, корректно завершает все активные операции, сохраняет пользовательские настройки, закрывает соединение с сервером.

# Заключение

В ходе выполнения курсовой работы было разработано программное средство «FTP Client», представляющее собой графический клиент для работы с FTP-серверами. Данное приложение предоставляет функциональные возможности для удобного управления файлами как локально, так и на удаленном сервере, с поддержкой всех основных файловых операций и мониторингом состояния соединения.

Разработка приложения осуществлялась с использованием современных технологий и подходов: клиентская часть реализована на языке Python с применением графической библиотеки Tkinter для создания интуитивно понятного пользовательского интерфейса. В качестве архитектурного подхода было выбрано объектно-ориентированное программирование с разделением логики на отдельные модули.

В результате разработки были реализованы следующие ключевые функции:

* подключение к FTP-серверу с использованием логина и пароля;
* просмотр списка файлов и директорий на удаленном сервере;
* загрузка файлов с сервера на локальный компьютер;
* выгрузка файлов с локального компьютера на сервер;
* удаление и переименование файлов на удаленном сервере;
* обработка ошибок подключения и передачи данных для повышения надежности работы программы;
* механизм быстрого подключения к FTP-серверам с сохранением истории и закладок;
* функционал для мониторинга состояния соединения с отслеживанием задержек и потерь пакетов.

Созданное приложение «FTP Client» позволило не только решить все поставленные задачи, но и углубить знания в области разработки графических приложений, сетевого программирования и работы с файловыми системами. Опыт работы с протоколом FTP и создания надежного пользовательского интерфейса значительно расширил понимание принципов разработки сетевых приложений.

Особое внимание было уделено решению технических проблем, возникших в процессе разработки, таких как обработка потери соединения, асинхронное выполнение длительных операций и корректная обработка ошибок.

Разработанное приложение представляет собой полноценный продукт, готовый к практическому использованию. В перспективе планируется добавление поддержки SFTP и FTPS протоколов, реализация многопоточной загрузки файлов и расширение функциональности управления правами доступа.

# Список использованных источников

1. Лутц М. Изучаем Python [Текст]. – 5-е изд. – СПб.: Символ-Плюс, 2021. – 1648 с. – ISBN 978-5-93286-159-2.
2. Ромальо Л. Python. К вершинам мастерства [Текст]. – М.: ДМК Пресс, 2022. – 768 с. – ISBN 978-5-93700-058-3.
3. Макконнелл С. Совершенный код [Текст]. – 2-е изд. – М.: Русская редакция, 2022. – 896 с. – ISBN 978-5-7502-0064-1.
4. Фримен Э., Робсон Э. Паттерны проектирования [Текст]. – СПб.: Питер, 2021. – 656 с. – ISBN 978-5-4461-0743-2.
5. Шефер М. Python и Tkinter. Разработка GUI [Текст]. – М.: БХВ-Петербург, 2023. – 400 с. – ISBN 978-5-9775-4102-5.
6. Python Software Foundation. Документация Tkinter [Электронный ресурс]. – URL: <https://docs.python.org/3/library/tkinter.html> .
7. PEP 8 — Style Guide for Python Code [Электронный ресурс]. – URL: <https://peps.python.org/pep-0008/> .
8. FileZilla – Официальный сайт FTP-клиента [Электронный ресурс]. – Режим доступа: <https://filezilla-project.org/> .
9. WinSCP – Официальный сайт SFTP и FTP-клиента [Электронный ресурс]. – Режим доступа: <https://winscp.net/> .
10. Кристофер, Н. Разработка сетевых приложений на Python [Текст]. – М.: ДМК Пресс, 2022. – 420 с. – ISBN 978-5-93700-154-2.
11. Вордерман, К. Python в сетевых технологиях [Текст]. – СПб.: Питер, 2021. – 352 с. – ISBN 978-5-4461-1452-2.

# Приложение А. Текст программы

import tkinter as tk  
from tkinter import ttk, messagebox, filedialog, simpledialog  
from ftplib import FTP, FTP\_TLS, error\_perm  
import os  
import socket  
from threading import Thread, Lock  
from concurrent.futures import ThreadPoolExecutor, as\_completed  
from queue import Queue  
from datetime import datetime  
import time  
import humanize  
import json  
import re  
from typing import Dict, List, Tuple  
import base64  
import sys  
import threading  
import shutil   
  
from src.core.ftp\_client import FTPClient  
from src.core.settings import Settings  
from src.gui.widgets import FileListView, ConnectionPanel, SearchPanel, PathPanel, StatusBar  
from src.gui.dialogs import QuickConnectDialog, HistoryDialog, BookmarksDialog, SettingsDialog, AboutDialog  
from src.gui.connection\_stats import ConnectionStatsPanel  
from src.utils.crypto import Crypto  
from src.utils.helpers import filter\_hidden\_files, sort\_items  
from src.gui.styles import setup\_styles  
  
  
def debug\_log(message: str):  
 sys.stderr.write(f"{message}\n")  
 sys.stderr.flush()  
  
  
class Application(tk.Tk):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
  
 self.title("FTP Client")  
 self.protocol("WM\_DELETE\_WINDOW", self.\_on\_closing)  
  
 screen\_width = self.winfo\_screenwidth()  
 screen\_height = self.winfo\_screenheight()  
  
 self.geometry(f"{screen\_width}x{screen\_height}+0+0")  
  
 if sys.platform == 'darwin':  
 self.attributes('-fullscreen', True)  
 elif sys.platform == 'win32':  
 self.state('zoomed')  
 else:   
 self.attributes('-zoomed', True)  
  
 self.settings = Settings()  
 self.ftp\_client = FTPClient()  
 self.crypto = Crypto()  
  
 self.connection\_history\_file = os.path.join(  
 os.path.expanduser("~"), ".ftp\_client\_history.json")  
 self.bookmarks\_file = os.path.join(  
 os.path.expanduser("~"), ".ftp\_client\_bookmarks.json")  
  
 self.connection\_history = self.\_load\_connection\_history()  
 self.bookmarks = self.\_load\_bookmarks()  
  
 setup\_styles()  
  
 self.\_create\_menu()  
 self.\_create\_main\_interface()  
 self.\_setup\_bindings()  
  
 self.update\_queue = Queue()  
 self.is\_updating = False  
 self.start\_update\_handler()  
  
 def \_create\_menu(self):  
 menubar = tk.Menu(self)  
 connection\_menu = tk.Menu(menubar, tearoff=False)  
 menubar.add\_cascade(label="Подключение", menu=connection\_menu)  
   
 if sys.platform == 'darwin':  
 menu\_width = 25   
 cmd\_symbol = "⌘"   
 else:  
 menu\_width = 30   
 cmd\_symbol = "Ctrl+"   
   
 connection\_menu.config(postcommand=lambda: self.\_adjust\_menu\_width(connection\_menu))  
 connection\_menu.add\_command(label="Быстрое подключение".ljust(menu\_width),   
 command=self.\_show\_quick\_connect,  
 accelerator="⌘Q" if sys.platform == 'darwin' else "Ctrl+Q")  
 connection\_menu.add\_command(label="История подключений".ljust(menu\_width),   
 command=self.\_show\_connection\_history,  
 accelerator="⌘H" if sys.platform == 'darwin' else "Ctrl+H")  
 connection\_menu.add\_command(label="Закладки".ljust(menu\_width),   
 command=self.\_show\_bookmarks)  
 connection\_menu.add\_command(label="Добавить в закладки".ljust(menu\_width),   
 command=self.\_add\_bookmark,  
 accelerator="⌘B" if sys.platform == 'darwin' else "Ctrl+B")  
 connection\_menu.add\_separator()  
 connection\_menu.add\_command(label="Отключиться".ljust(menu\_width),   
 command=self.\_disconnect,  
 state="disabled")  
 self.connection\_menu = connection\_menu  
  
 operations\_menu = tk.Menu(menubar, tearoff=False)  
 operations\_menu.config(postcommand=lambda: self.\_adjust\_menu\_width(operations\_menu))  
 menubar.add\_cascade(label="Операции", menu=operations\_menu)  
 operations\_menu.add\_command(label="Создать папку".ljust(menu\_width),   
 command=self.\_create\_folder,  
 accelerator="⌘N" if sys.platform == 'darwin' else "Ctrl+N")  
 operations\_menu.add\_command(label="Загрузить файлы".ljust(menu\_width),   
 command=self.\_upload\_files,  
 accelerator="⌘U" if sys.platform == 'darwin' else "Ctrl+U")  
 operations\_menu.add\_command(label="Скачать файлы".ljust(menu\_width),   
 command=self.\_download\_files,  
 accelerator="⌘D" if sys.platform == 'darwin' else "Ctrl+D")  
 operations\_menu.add\_separator()  
 operations\_menu.add\_command(label="Обновить списки".ljust(menu\_width),   
 command=self.\_refresh\_lists,  
 accelerator="F5 / ⌘R" if sys.platform == 'darwin' else "F5 / Ctrl+R")  
  
 settings\_menu = tk.Menu(menubar, tearoff=False)  
 settings\_menu.config(postcommand=lambda: self.\_adjust\_menu\_width(settings\_menu))  
 menubar.add\_cascade(label="Настройки", menu=settings\_menu)  
 settings\_menu.add\_command(label="Параметры".ljust(menu\_width),   
 command=self.\_show\_settings,  
 accelerator="⌘," if sys.platform == 'darwin' else "Ctrl+,")  
  
 help\_menu = tk.Menu(menubar, tearoff=False)  
 help\_menu.config(postcommand=lambda: self.\_adjust\_menu\_width(help\_menu))  
 menubar.add\_cascade(label="Справка", menu=help\_menu)  
 help\_menu.add\_command(label="О программе".ljust(menu\_width),   
 command=self.\_show\_about)  
  
 self['menu'] = menubar  
  
 def \_adjust\_menu\_width(self, menu):  
 try:  
 labels = []  
 for i in range(menu.index('end') + 1):  
 try:  
 if menu.type(i) == 'command':  
 labels.append(menu.entrycget(i, 'label'))  
 except:  
 pass  
  
 max\_width = max(len(label) for label in labels if label)  
  
 for i in range(menu.index('end') + 1):  
 try:  
 if menu.type(i) == 'command':  
 current\_label = menu.entrycget(i, 'label').rstrip()  
 menu.entryconfig(i, label=current\_label.ljust(max\_width + 5))  
 except:  
 pass  
 except:  
 pass   
  
 def \_create\_toolbar(self):  
 toolbar = ttk.Frame(self)  
 toolbar.pack(fill=tk.X, padx=5, pady=5)  
  
 local\_frame = ttk.LabelFrame(toolbar, text="Локальные")  
 local\_frame.pack(side=tk.LEFT, padx=5)  
  
 remote\_frame = ttk.LabelFrame(toolbar, text="Удаленные")  
 remote\_frame.pack(side=tk.LEFT, padx=5)  
  
 common\_frame = ttk.LabelFrame(toolbar, text="Общие")  
 common\_frame.pack(side=tk.LEFT, padx=5)  
  
 ttk.Button(local\_frame, text="↑ Наверх",   
 command=self.\_navigate\_up\_local).pack(side=tk.LEFT, padx=2)  
  
 ttk.Button(remote\_frame, text="↑ Наверх",   
 command=self.\_navigate\_up\_remote).pack(side=tk.LEFT, padx=2)  
  
 buttons = [  
 ("↻ Обновить", self.\_refresh\_lists),  
 ("✚ Папка", self.\_create\_folder),  
 ("↑ Загрузить", self.\_upload\_files),  
 ("↓ Скачать", self.\_download\_files),  
 ("✕ Удалить", self.\_delete\_selected)  
 ]  
  
 for text, command in buttons:  
 ttk.Button(common\_frame, text=text, command=command).pack(side=tk.LEFT, padx=2)  
  
 def \_create\_main\_interface(self):  
 self.connection\_panel = ConnectionPanel(self, self.\_connect)  
 self.connection\_panel.pack(fill=tk.X, padx=5, pady=5)  
  
 self.stats\_panel = ConnectionStatsPanel(self)  
 self.stats\_panel.pack(fill=tk.X, padx=5, pady=2)  
  
 self.search\_panel = SearchPanel(self, self.\_on\_search)  
 self.search\_panel.pack(fill=tk.X, padx=5, pady=2)  
  
 self.\_create\_toolbar()  
  
 main\_frame = ttk.Frame(self)  
 main\_frame.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)  
  
 local\_frame = ttk.LabelFrame(main\_frame, text="Локальные файлы")  
 local\_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=5)  
   
 self.local\_path = PathPanel(local\_frame, self.\_change\_local\_directory)  
 self.local\_path.pack(fill=tk.X, padx=5, pady=2)  
   
 self.local\_files = FileListView(local\_frame)  
 self.local\_files.pack(fill=tk.BOTH, expand=True)  
  
 remote\_frame = ttk.LabelFrame(main\_frame, text="Удаленные файлы")  
 remote\_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True, padx=5)  
   
 self.remote\_path = PathPanel(remote\_frame)  
 self.remote\_path.pack(fill=tk.X, padx=5, pady=2)  
   
 self.remote\_files = FileListView(remote\_frame)  
 self.remote\_files.pack(fill=tk.BOTH, expand=True)  
  
 self.status\_bar = StatusBar(self)  
 self.status\_bar.pack(fill=tk.X, padx=5, pady=5)  
  
 self.\_refresh\_local\_list()  
  
 def \_setup\_bindings(self):  
 self.local\_files.bind("<Double-1>", self.\_on\_local\_double\_click)  
 self.remote\_files.bind("<Double-1>", self.\_on\_remote\_double\_click)  
 self.bind\_all("<F5>", lambda e: self.\_refresh\_lists())  
 self.bind\_all("<Escape>", lambda e: self.\_toggle\_fullscreen())  
 def handle\_backspace(event):  
 debug\_log(f"\nDEBUG: Нажата клавиша: BackSpace")  
 debug\_log(f"DEBUG: Событие: {event}")  
 debug\_log(f"DEBUG: Keysym: {event.keysym}")  
 debug\_log(f"DEBUG: Keycode: {event.keycode}")  
 debug\_log(f"DEBUG: Char: {event.char if hasattr(event, 'char') else 'Нет символа'}")  
   
 if event.keycode == 855638143:  
 debug\_log("DEBUG: Обработка удаления файла")  
 if self.local\_files.focus\_get() == self.local\_files:  
 debug\_log("DEBUG: Удаление из локальной панели")  
 self.\_delete\_local()  
 elif self.remote\_files.focus\_get() == self.remote\_files and self.ftp\_client.ftp:  
 debug\_log("DEBUG: Удаление с удаленного сервера")  
 selected = self.remote\_files.selection()  
 if not selected:  
 return "break"  
   
 if not messagebox.askyesno("Подтверждение",   
 "Вы действительно хотите удалить выбранные файлы?"):  
 return "break"  
   
 for item\_id in selected:  
 values = self.remote\_files.item(item\_id)['values']  
 filename = str(values[0])  
 is\_dir = values[2] == "Папка"  
   
 if is\_dir:  
 success, message = self.ftp\_client.delete\_directory\_recursive(filename)  
 else:  
 success, message = self.ftp\_client.delete\_item(filename)  
   
 if not success:  
 if message == "NOT\_EMPTY\_DIR":  
 if messagebox.askyesno("Подтверждение",   
 f"Папка '{filename}' не пуста. Удалить её содержимое?"):  
 success, message = self.ftp\_client.delete\_directory\_recursive(filename)  
 else:  
 messagebox.showerror("Ошибка", f"Не удалось удалить {filename}: {message}")  
   
 self.\_refresh\_remote\_list()  
 return "break"  
 return self.\_navigate\_up()  
   
 self.bind\_all("<BackSpace>", handle\_backspace)  
 self.bind\_all("<Alt-Left>", lambda e: self.\_navigate\_up())  
 self.bind\_all("<Alt-Up>", lambda e: self.\_navigate\_up())  
   
 def debug\_delete(event):  
 debug\_log(f"\nDEBUG: Нажата клавиша Delete")  
 debug\_log(f"DEBUG: Событие: {event}")  
 debug\_log(f"DEBUG: Keysym: {event.keysym}")  
 debug\_log(f"DEBUG: Keycode: {event.keycode}")  
 debug\_log(f"DEBUG: Char: {event.char if hasattr(event, 'char') else 'Нет символа'}")  
 return "break"  
   
 self.bind\_all("<Delete>", debug\_delete)  
   
 if sys.platform == 'darwin':  
 self.local\_files.bind("<Button-2>", self.\_show\_local\_menu)  
 self.remote\_files.bind("<Button-2>", self.\_show\_remote\_menu)  
 else:  
 self.local\_files.bind("<Button-3>", self.\_show\_local\_menu)  
 self.remote\_files.bind("<Button-3>", self.\_show\_remote\_menu)  
  
 if sys.platform == 'darwin':  
 self.bind\_all("<Meta-q>", lambda e: self.\_show\_quick\_connect())  
 self.bind\_all("<Meta-r>", lambda e: self.\_refresh\_lists())  
 self.bind\_all("<Meta-u>", lambda e: self.\_upload\_files())  
 self.bind\_all("<Meta-d>", lambda e: self.\_download\_files())  
 self.bind\_all("<Meta-n>", lambda e: self.\_create\_folder())  
 self.bind\_all("<Meta-b>", lambda e: self.\_add\_bookmark())  
 self.bind\_all("<Meta-h>", lambda e: self.\_show\_connection\_history())  
 self.bind\_all("<Meta-comma>", lambda e: self.\_show\_settings())  
   
 self.bind\_all("<Command-BackSpace>", lambda e: self.\_navigate\_up())  
 self.bind\_all("<Command-Up>", lambda e: self.\_navigate\_up())  
 self.bind\_all("<Command-Left>", lambda e: self.\_navigate\_up())  
   
 self.bind\_all("<Command-q>", lambda e: self.\_show\_quick\_connect())  
 self.bind\_all("<Command-r>", lambda e: self.\_refresh\_lists())  
 self.bind\_all("<Command-u>", lambda e: self.\_upload\_files())  
 self.bind\_all("<Command-d>", lambda e: self.\_download\_files())  
 self.bind\_all("<Command-n>", lambda e: self.\_create\_folder())  
 self.bind\_all("<Command-b>", lambda e: self.\_add\_bookmark())  
 self.bind\_all("<Command-h>", lambda e: self.\_show\_connection\_history())  
 self.bind\_all("<Command-comma>", lambda e: self.\_show\_settings())  
 else:  
 self.bind\_all("<Control-q>", lambda e: self.\_show\_quick\_connect())  
 self.bind\_all("<Control-r>", lambda e: self.\_refresh\_lists())  
 self.bind\_all("<Control-u>", lambda e: self.\_upload\_files())  
 self.bind\_all("<Control-d>", lambda e: self.\_download\_files())  
 self.bind\_all("<Control-n>", lambda e: self.\_create\_folder())  
 self.bind\_all("<Control-b>", lambda e: self.\_add\_bookmark())  
 self.bind\_all("<Control-h>", lambda e: self.\_show\_connection\_history())  
 self.bind\_all("<Control-comma>", lambda e: self.\_show\_settings())  
   
 self.\_create\_context\_menus()  
  
 def \_create\_context\_menus(self):  
 debug\_log("\nDEBUG: Создание контекстных меню")  
  
 self.local\_menu = tk.Menu(self, tearoff=0)  
 self.local\_menu.add\_command(label="Открыть", command=self.\_open\_local\_file)  
 self.local\_menu.add\_separator()  
 self.local\_menu.add\_command(label="Копировать", command=lambda: self.\_copy\_files('local'))  
 self.local\_menu.add\_command(label="Вставить", command=lambda: self.\_paste\_files('local'))  
 self.local\_menu.add\_separator()  
 self.local\_menu.add\_command(label="Переименовать", command=self.\_rename\_local)  
 self.local\_menu.add\_command(label="Удалить", command=self.\_delete\_local)  
 self.local\_menu.add\_separator()  
 self.local\_menu.add\_command(label="Загрузить на сервер", command=self.\_upload\_files)  
  
 self.remote\_menu = tk.Menu(self, tearoff=0)  
 self.remote\_menu.add\_command(label="Скачать", command=self.\_download\_files)  
 self.remote\_menu.add\_separator()  
 self.remote\_menu.add\_command(label="Копировать", command=lambda: self.\_copy\_files('remote'))  
 self.remote\_menu.add\_command(label="Вставить", command=lambda: self.\_paste\_files('remote'))  
 self.remote\_menu.add\_separator()  
 self.remote\_menu.add\_command(label="Переименовать", command=self.\_rename\_remote)  
 self.remote\_menu.add\_command(label="Удалить", command=self.\_delete\_remote)  
  
 def \_show\_local\_menu(self, event):  
 debug\_log("\nDEBUG: Вызов локального контекстного меню")  
   
 item = self.local\_files.identify\_row(event.y)  
 if item:  
 debug\_log(f"DEBUG: Выбран элемент {item}")  
 if not (event.state & 0x0004):   
 for selected\_item in self.local\_files.selection():  
 self.local\_files.selection\_remove(selected\_item)  
 self.local\_files.selection\_add(item)  
 self.local\_menu.post(event.x\_root, event.y\_root)  
 debug\_log(f"DEBUG: Меню показано в координатах {event.x\_root}, {event.y\_root}")  
 return "break"  
  
 def \_show\_remote\_menu(self, event):  
 debug\_log("\nDEBUG: Вызов удаленного контекстного меню")  
   
 if not self.ftp\_client.ftp:  
 debug\_log("DEBUG: FTP клиент не подключен")  
 return  
   
 item = self.remote\_files.identify\_row(event.y)  
 if item:  
 debug\_log(f"DEBUG: Выбран элемент {item}")  
 if not (event.state & 0x0004):   
 for selected\_item in self.remote\_files.selection():  
 self.remote\_files.selection\_remove(selected\_item)  
 self.remote\_files.selection\_add(item)  
 self.remote\_menu.post(event.x\_root, event.y\_root)  
 debug\_log(f"DEBUG: Меню показано в координатах {event.x\_root}, {event.y\_root}")  
 return "break"  
  
 def \_copy\_files(self, source):  
 self.clipboard\_source = source  
 self.clipboard\_files = []  
   
 if source == 'local':  
 selected = self.local\_files.selection()  
 for item\_id in selected:  
 values = self.local\_files.item(item\_id)['values']  
 self.clipboard\_files.append({  
 'name': str(values[0]),  
 'type': values[2]  
 })  
 else:  
 selected = self.remote\_files.selection()  
 for item\_id in selected:  
 values = self.remote\_files.item(item\_id)['values']  
 self.clipboard\_files.append({  
 'name': str(values[0]),  
 'type': values[2]  
 })  
   
 self.status\_bar.set\_status(f"Скопировано элементов: {len(self.clipboard\_files)}")  
  
 def \_paste\_files(self, target):  
 if not hasattr(self, 'clipboard\_files') or not self.clipboard\_files:  
 return  
   
 if self.clipboard\_source == target:  
 if target == 'local':  
 try:  
 total = len(self.clipboard\_files)  
   
 for i, file\_info in enumerate(self.clipboard\_files, 1):  
 try:  
 filename = str(file\_info['name'])  
 src\_path = os.path.join(self.settings.get('default\_local\_dir'), filename)  
 base, ext = os.path.splitext(filename)  
 new\_name = f"{base} - копия{ext}"  
 dst\_path = os.path.join(self.settings.get('default\_local\_dir'), new\_name)  
 if os.path.exists(dst\_path):  
 if not messagebox.askyesno("Подтверждение",   
 f"Файл {new\_name} уже существует. Перезаписать?"):  
 continue  
 if os.path.isdir(dst\_path):  
 shutil.rmtree(dst\_path)  
 else:  
 os.remove(dst\_path)  
  
 if file\_info['type'] == "Папка":  
 shutil.copytree(src\_path, dst\_path)  
 else:  
 shutil.copy2(src\_path, dst\_path)  
   
 progress = (i / total) \* 100  
 self.status\_bar.set\_progress(progress)  
 self.status\_bar.set\_status(f"Копирование {i}/{total}: {filename}")  
   
 except Exception as e:  
 messagebox.showerror("Ошибка", f"Не удалось скопировать {filename}: {str(e)}")  
   
 self.\_refresh\_local\_list()  
 self.status\_bar.set\_status("Копирование завершено")  
 self.status\_bar.set\_progress(100)  
   
 except Exception as e:  
 self.status\_bar.set\_status(f"Ошибка копирования: {str(e)}", error=True)  
 messagebox.showerror("Ошибка", f"Ошибка копирования: {str(e)}")  
   
 else:   
 if not self.ftp\_client.ftp:  
 return  
   
 def copy\_thread():  
 try:  
 total = len(self.clipboard\_files)  
 for i, file\_info in enumerate(self.clipboard\_files, 1):  
 filename = str(file\_info['name'])  
 base, ext = os.path.splitext(filename)  
 new\_name = f"{base} - копия{ext}"  
   
 progress = (i - 1) / total \* 100  
 self.schedule\_update(lambda p=progress, f=filename: [  
 self.status\_bar.set\_progress(p),  
 self.status\_bar.set\_status(f"Копирование {i}/{total}: {f}")  
 ])  
 try:  
 if file\_info['type'] == "Папка":  
 self.ftp\_client.ftp.cwd(new\_name)  
 self.ftp\_client.ftp.cwd('..')  
 else:  
 self.ftp\_client.ftp.size(new\_name)  
   
 if self.settings.get('confirm\_overwrite', True):  
 confirm\_event = threading.Event()  
 self.schedule\_update(lambda: [  
 setattr(confirm\_event, 'result',  
 messagebox.askyesno("Подтверждение",  
 f"Файл {new\_name} уже существует. Перезаписать?"))  
 ])  
 confirm\_event.wait()  
 if not confirm\_event.result:  
 continue  
 except:  
 pass  
   
 success, message = self.ftp\_client.copy\_file(filename, new\_name)  
 if not success:  
 self.schedule\_update(lambda: [  
 messagebox.showerror("Ошибка", f"Ошибка копирования {filename}: {message}"),  
 self.status\_bar.set\_status(f"Ошибка копирования: {message}", error=True)  
 ])  
 return  
  
 progress = (i / total) \* 100  
 self.schedule\_update(lambda p=progress: [  
 self.status\_bar.set\_progress(p)  
 ])  
  
 self.schedule\_update(lambda: [  
 self.\_refresh\_remote\_list(),  
 self.status\_bar.set\_status("Копирование завершено"),  
 self.status\_bar.set\_progress(100)  
 ])  
  
 except Exception as e:  
 self.schedule\_update(lambda: [  
 messagebox.showerror("Ошибка", f"Ошибка копирования: {str(e)}"),  
 self.status\_bar.set\_status(f"Ошибка копирования: {str(e)}", error=True)  
 ])  
  
 Thread(target=copy\_thread, daemon=True).start()  
   
 else:  
 if self.clipboard\_source == 'local':  
 self.\_upload\_files()  
 else:  
 self.\_download\_files()  
   
 self.status\_bar.set\_status("Вставка завершена")  
  
 def \_open\_local\_file(self):  
 selected = self.local\_files.selection()  
 if not selected:  
 return  
   
 values = self.local\_files.item(selected[0])['values']  
 filename = str(values[0])  
 path = os.path.join(self.settings.get('default\_local\_dir'), filename)  
   
 try:  
 import subprocess  
 if sys.platform == 'darwin':  
 subprocess.run(['open', path])  
 elif sys.platform == 'win32':  
 os.startfile(path)  
 else:  
 subprocess.run(['xdg-open', path])  
 except Exception as e:  
 self.status\_bar.set\_status(f"Ошибка открытия файла: {str(e)}", error=True)  
  
 def \_rename\_local(self):  
 selected = self.local\_files.selection()  
 if not selected:  
 return  
   
 values = self.local\_files.item(selected[0])['values']  
 old\_name = str(values[0])  
 dialog = tk.Toplevel(self)  
 dialog.title("Переименовать")  
 dialog.geometry("300x120")  
 dialog.resizable(False, False)  
 dialog.transient(self)  
 dialog.grab\_set()  
 dialog.geometry("+%d+%d" % (  
 self.winfo\_rootx() + self.winfo\_width()//2 - 150,  
 self.winfo\_rooty() + self.winfo\_height()//2 - 60  
 ))  
 frame = ttk.Frame(dialog, padding="10")  
 frame.pack(fill=tk.BOTH, expand=True)  
   
 ttk.Label(frame, text="Введите новое имя:").pack(pady=(0, 5))  
   
 entry = ttk.Entry(frame, width=40)  
 entry.insert(0, old\_name)  
 entry.pack(pady=(0, 10))  
 entry.select\_range(0, len(old\_name))  
 entry.focus()  
   
 btn\_frame = ttk.Frame(frame)  
 btn\_frame.pack(fill=tk.X)  
   
 def rename():  
 new\_name = entry.get().strip()  
 if new\_name and new\_name != old\_name:  
 dialog.destroy()  
 self.\_perform\_local\_rename(old\_name, new\_name)  
 elif not new\_name:  
 messagebox.showwarning("Ошибка", "Имя файла не может быть пустым")  
   
 ttk.Button(btn\_frame, text="OK", command=rename, width=10).pack(side=tk.LEFT, padx=5)  
 ttk.Button(btn\_frame, text="Отмена", command=dialog.destroy, width=10).pack(side=tk.LEFT)  
  
 dialog.bind('<Return>', lambda e: rename())  
 dialog.bind('<Escape>', lambda e: dialog.destroy())  
  
 def \_perform\_local\_rename(self, old\_name, new\_name):  
 try:  
 current\_dir = self.settings.get('default\_local\_dir')  
 old\_path = os.path.join(current\_dir, old\_name)  
 new\_path = os.path.join(current\_dir, new\_name)  
   
 debug\_log(f"\nDEBUG: Переименование файла")  
 debug\_log(f"DEBUG: Текущая директория: {current\_dir}")  
 debug\_log(f"DEBUG: Старый путь: {old\_path}")  
 debug\_log(f"DEBUG: Новый путь: {new\_path}")  
 if not os.path.exists(old\_path):  
 error\_msg = f"Файл '{old\_name}' не найден в директории '{current\_dir}'"  
 debug\_log(f"DEBUG: {error\_msg}")  
 self.status\_bar.set\_status(error\_msg, error=True)  
 messagebox.showerror("Ошибка", error\_msg)  
 return  
  
 if os.path.exists(new\_path):  
 if not messagebox.askyesno("Подтверждение",   
 f"Файл {new\_name} уже существует. Перезаписать?"):  
 return  
 try:  
 if os.path.isdir(new\_path):  
 import shutil  
 shutil.rmtree(new\_path)  
 else:  
 os.remove(new\_path)  
 except Exception as e:  
 error\_msg = f"Не удалось удалить существующий файл: {str(e)}"  
 debug\_log(f"DEBUG: {error\_msg}")  
 self.status\_bar.set\_status(error\_msg, error=True)  
 messagebox.showerror("Ошибка", error\_msg)  
 return  
   
 os.rename(old\_path, new\_path)  
 self.\_refresh\_local\_list()  
 success\_msg = f"Файл переименован: {old\_name} -> {new\_name}"  
 debug\_log(f"DEBUG: {success\_msg}")  
 self.status\_bar.set\_status(success\_msg)  
 except Exception as e:  
 error\_msg = str(e)  
 debug\_log(f"DEBUG: Ошибка переименования: {error\_msg}")  
 if "Permission denied" in error\_msg:  
 error\_msg = "Отказано в доступе. Проверьте права на файл и директорию."  
 elif "No such file or directory" in error\_msg:  
 error\_msg = f"Файл или директория не найдены. Проверьте, что файл '{old\_name}' существует."  
 elif "Invalid cross-device link" in error\_msg:  
 error\_msg = "Невозможно переместить файл между разными дисками. Попробуйте скопировать и удалить."  
   
 self.status\_bar.set\_status(f"Ошибка переименования: {error\_msg}", error=True)  
 messagebox.showerror("Ошибка", f"Не удалось переименовать файл: {error\_msg}")  
  
 def \_create\_local\_dir(self):  
 dialog = tk.Toplevel(self)  
 dialog.title("Создать папку")  
 dialog.geometry("300x120")  
 dialog.resizable(False, False)  
 dialog.transient(self)  
 dialog.grab\_set()  
 dialog.geometry("+%d+%d" % (  
 self.winfo\_rootx() + self.winfo\_width()//2 - 150,  
 self.winfo\_rooty() + self.winfo\_height()//2 - 60  
 ))  
  
 frame = ttk.Frame(dialog, padding="10")  
 frame.pack(fill=tk.BOTH, expand=True)  
   
 ttk.Label(frame, text="Введите имя папки:").pack(pady=(0, 5))  
   
 entry = ttk.Entry(frame, width=40)  
 entry.pack(pady=(0, 10))  
 entry.focus()  
   
 btn\_frame = ttk.Frame(frame)  
 btn\_frame.pack(fill=tk.X)  
   
 def create():  
 dirname = entry.get().strip()  
 if not dirname:  
 messagebox.showwarning("Ошибка", "Имя папки не может быть пустым")  
 return  
   
 dialog.destroy()  
 try:  
 path = os.path.join(self.settings.get('default\_local\_dir'), dirname)  
 os.makedirs(path)  
 self.\_refresh\_local\_list()  
 self.status\_bar.set\_status(f"Создана папка: {dirname}")  
 except Exception as e:  
 error\_msg = str(e)  
 if "Permission denied" in error\_msg:  
 error\_msg = "Отказано в доступе. Проверьте права на директорию."  
 elif "File exists" in error\_msg:  
 error\_msg = f"Папка '{dirname}' уже существует."  
 self.status\_bar.set\_status(f"Ошибка создания папки: {error\_msg}", error=True)  
 messagebox.showerror("Ошибка", f"Не удалось создать папку: {error\_msg}")  
   
 ttk.Button(btn\_frame, text="OK", command=create, width=10).pack(side=tk.LEFT, padx=5)  
 ttk.Button(btn\_frame, text="Отмена", command=dialog.destroy, width=10).pack(side=tk.LEFT)  
   
 dialog.bind('<Return>', lambda e: create())  
 dialog.bind('<Escape>', lambda e: dialog.destroy())  
  
 def \_rename\_remote(self):  
 if not self.ftp\_client.ftp:  
 messagebox.showwarning("Ошибка", "Сначала подключитесь к серверу")  
 return  
   
 selected = self.remote\_files.selection()  
 if not selected:  
 return   
 values = self.remote\_files.item(selected[0])['values']  
 old\_name = str(values[0])  
 dialog = tk.Toplevel(self)  
 dialog.title("Переименовать")  
 dialog.geometry("300x120")  
 dialog.resizable(False, False)  
 dialog.transient(self)  
 dialog.grab\_set()  
 dialog.geometry("+%d+%d" % (  
 self.winfo\_rootx() + self.winfo\_width()//2 - 150,  
 self.winfo\_rooty() + self.winfo\_height()//2 - 60  
 ))  
   
 frame = ttk.Frame(dialog, padding="10")  
 frame.pack(fill=tk.BOTH, expand=True)  
   
 ttk.Label(frame, text="Введите новое имя:").pack(pady=(0, 5))  
   
 entry = ttk.Entry(frame, width=40)  
 entry.insert(0, old\_name)  
 entry.pack(pady=(0, 10))  
 entry.select\_range(0, len(old\_name))  
 entry.focus()  
   
 btn\_frame = ttk.Frame(frame)  
 btn\_frame.pack(fill=tk.X)  
   
 def rename():  
 new\_name = entry.get().strip()  
 if new\_name and new\_name != old\_name:  
 dialog.destroy()  
 try:  
 debug\_log(f"\nDEBUG: Переименование файла на сервере")  
 debug\_log(f"DEBUG: Текущая директория: {self.ftp\_client.get\_current\_directory()}")  
 debug\_log(f"DEBUG: Старое имя: {old\_name}")  
 debug\_log(f"DEBUG: Новое имя: {new\_name}")  
   
 try:  
 self.ftp\_client.ftp.size(new\_name)  
 if not messagebox.askyesno("Подтверждение",   
 f"Файл {new\_name} уже существует. Перезаписать?"):  
 return  
 except:  
 pass  
   
 self.ftp\_client.ftp.rename(old\_name, new\_name)  
 self.\_refresh\_remote\_list()  
 self.status\_bar.set\_status(f"Файл переименован: {old\_name} -> {new\_name}")  
 except Exception as e:  
 error\_msg = str(e)  
 debug\_log(f"DEBUG: Ошибка переименования: {error\_msg}")  
 self.status\_bar.set\_status(f"Ошибка переименования: {error\_msg}", error=True)  
 messagebox.showerror("Ошибка", f"Не удалось переименовать файл: {error\_msg}")  
 elif not new\_name:  
 messagebox.showwarning("Ошибка", "Имя файла не может быть пустым")  
   
 ttk.Button(btn\_frame, text="OK", command=rename, width=10).pack(side=tk.LEFT, padx=5)  
 ttk.Button(btn\_frame, text="Отмена", command=dialog.destroy, width=10).pack(side=tk.LEFT)  
 dialog.bind('<Return>', lambda e: rename())  
 dialog.bind('<Escape>', lambda e: dialog.destroy())  
  
 def \_create\_remote\_dir(self):  
 if not self.ftp\_client.ftp:  
 messagebox.showwarning("Ошибка", "Сначала подключитесь к серверу")  
 return  
   
 dirname = simpledialog.askstring("Создать папку", "Введите имя папки:")  
 if not dirname:  
 return  
   
 try:  
 self.ftp\_client.ftp.mkd(dirname)  
 self.\_refresh\_remote\_list()  
 self.status\_bar.set\_status(f"Создана папка: {dirname}")  
 except Exception as e:  
 self.status\_bar.set\_status(f"Ошибка создания папки: {str(e)}", error=True)  
 messagebox.showerror("Ошибка", f"Не удалось создать папку: {str(e)}")  
  
 def start\_update\_handler(self):  
 def update\_handler():  
 if not self.is\_updating and not self.update\_queue.empty():  
 self.is\_updating = True  
 try:  
 while not self.update\_queue.empty():  
 update\_func = self.update\_queue.get\_nowait()  
 if update\_func:  
 update\_func()  
 finally:  
 self.is\_updating = False  
 self.after(100, update\_handler)  
   
 self.after(100, update\_handler)  
  
 def schedule\_update(self, update\_func):  
 self.update\_queue.put(update\_func)  
  
 def \_load\_connection\_history(self) -> List[Dict]:  
 try:  
 if os.path.exists(self.connection\_history\_file):  
 with open(self.connection\_history\_file, 'r') as f:  
 return json.load(f)  
 except Exception as e:  
 print(f"Ошибка загрузки истории: {e}")  
 return []  
  
 def \_save\_connection\_history(self):  
 try:  
 with open(self.connection\_history\_file, 'w') as f:  
 json.dump(self.connection\_history, f)  
 except Exception as e:  
 print(f"Ошибка сохранения истории: {e}")  
  
 def \_load\_bookmarks(self) -> List[Dict]:  
 try:  
 if os.path.exists(self.bookmarks\_file):  
 with open(self.bookmarks\_file, 'r') as f:  
 return json.load(f)  
 except Exception as e:  
 print(f"Ошибка загрузки закладок: {e}")  
 return []  
  
 def \_save\_bookmarks(self):  
 try:  
 with open(self.bookmarks\_file, 'w') as f:  
 json.dump(self.bookmarks, f)  
 except Exception as e:  
 print(f"Ошибка сохранения закладок: {e}")  
  
 def \_add\_to\_history(self, host: str, port: int, user: str):  
 connection = {  
 'host': host,  
 'port': port,  
 'user': user,  
 'timestamp': datetime.now().isoformat()  
 }  
  
 self.connection\_history = [  
 c for c in self.connection\_history   
 if not (c['host'] == host and c['port'] == port and c['user'] == user)  
 ]  
  
 self.connection\_history.insert(0, connection)  
  
 self.connection\_history = self.connection\_history[:10]  
 self.\_save\_connection\_history()  
  
 def \_connect(self, host, port, user, password):  
 if host is None and port is None and user is None and password is None:  
 self.\_disconnect()  
 return False  
 if not host or not user or not password:  
 messagebox.showerror("Ошибка", "Все поля должны быть заполнены")  
 return False  
 try:  
 port = int(port)  
 if port < 1 or port > 65535:  
 messagebox.showerror("Ошибка", "Порт должен быть числом от 1 до 65535")  
 return False  
 except (ValueError, TypeError):  
 messagebox.showerror("Ошибка", "Порт должен быть числом")  
 return False  
  
 try:  
 success, message = self.ftp\_client.connect(host, port, user, password)  
 if success:  
 self.status\_bar.set\_status("Подключено к серверу")  
 self.connection\_panel.set\_connected\_state(True)  
 for i in range(self.connection\_menu.index('end') + 1):  
 try:  
 if "Отключиться" in self.connection\_menu.entrycget(i, 'label').strip():  
 self.connection\_menu.entryconfig(i, state="normal")  
 break  
 except:  
 continue  
 self.\_add\_to\_history(host, port, user)  
 self.stats\_panel.start\_monitoring(host, port)  
 self.ftp\_client.start\_connection\_monitor(self.\_on\_connection\_lost)  
 self.\_refresh\_remote\_list()  
 return True  
 else:  
 error\_msg = message  
 if "530" in message:  
 error\_msg = "Неверное имя пользователя или пароль"  
 elif "connection refused" in message.lower():  
 error\_msg = "Подключение отклонено. Проверьте адрес и порт."  
 elif "[Errno 8]" in message:  
 error\_msg = "Не удалось найти сервер. Проверьте правильность введенного адреса."  
   
 self.status\_bar.set\_status(error\_msg, error=True)  
 messagebox.showerror("Ошибка подключения", error\_msg)  
 return False  
   
 except Exception as e:  
 error\_msg = str(e)  
 if "timeout" in error\_msg.lower():  
 error\_msg = "Превышено время ожидания подключения"  
 elif "connection refused" in error\_msg.lower():  
 error\_msg = "Подключение отклонено. Проверьте адрес и порт."  
 elif "[Errno 8]" in error\_msg:  
 error\_msg = "Не удалось найти сервер. Проверьте правильность введенного адреса."  
   
 self.status\_bar.set\_status(error\_msg, error=True)  
 messagebox.showerror("Ошибка подключения", error\_msg)  
 return False  
  
 def \_disconnect(self):  
 debug\_log("\nDEBUG: Начало отключения от сервера")  
   
 try:  
 debug\_log("DEBUG: Останавливаем мониторинг")  
 self.stats\_panel.stop\_monitoring()  
   
 debug\_log("DEBUG: Вызываем disconnect у FTP клиента")  
 self.ftp\_client.disconnect()  
   
 debug\_log("DEBUG: Обновляем состояние панели подключения")  
 self.connection\_panel.set\_connected\_state(False)  
   
 debug\_log("DEBUG: Деактивируем пункт меню Отключиться")  
 for i in range(self.connection\_menu.index('end') + 1):  
 try:  
 if "Отключиться" in self.connection\_menu.entrycget(i, 'label').strip():  
 self.connection\_menu.entryconfig(i, state="disabled")  
 break  
 except:  
 continue  
   
 debug\_log("DEBUG: Очищаем список удаленных файлов")  
 self.remote\_files.delete(\*self.remote\_files.get\_children())  
   
 debug\_log("DEBUG: Очищаем путь")  
 self.remote\_path.set\_path("")  
   
 debug\_log("DEBUG: Обновляем статус")  
 self.status\_bar.set\_status("Отключено от сервера")  
   
 debug\_log("DEBUG: Отключение завершено успешно")  
 except Exception as e:  
 debug\_log(f"DEBUG: Ошибка при отключении: {str(e)}")  
 self.status\_bar.set\_status(f"Ошибка при отключении: {str(e)}", error=True)  
  
 def \_on\_connection\_lost(self):  
 def update():  
 self.status\_bar.set\_status("Соединение потеряно", error=True)  
 self.connection\_panel.set\_connected\_state(False)  
 self.connection\_menu.entryconfig("Отключиться", state="disabled")  
 self.stats\_panel.stop\_monitoring()  
 self.remote\_files.clear()  
 messagebox.showerror("Ошибка", "Соединение с сервером потеряно")  
 self.schedule\_update(update)  
  
 def \_refresh\_lists(self):  
 self.\_refresh\_local\_list()  
 self.\_refresh\_remote\_list()  
  
 def \_refresh\_local\_list(self):  
 try:  
 items = []  
 current\_dir = self.settings.get('default\_local\_dir')  
 for item in os.listdir(current\_dir):  
 try:  
 path = os.path.join(current\_dir, item)  
 stat = os.stat(path)  
 is\_dir = os.path.isdir(path)  
  
 if is\_dir:  
 try:  
 size = f"{len(os.listdir(path))} элем."  
 except:  
 size = "Нет доступа"  
 else:  
 size = humanize.naturalsize(stat.st\_size)  
  
 modified = datetime.fromtimestamp(stat.st\_mtime).strftime(  
 self.settings.get('date\_format', "%Y-%m-%d %H:%M")  
 )  
  
 items.append({  
 'name': item,  
 'size': size,  
 'type': "Папка" if is\_dir else "Файл",  
 'modified': modified  
 })  
 except Exception as e:  
 items.append({  
 'name': item,  
 'size': "Ошибка",  
 'type': "Неизвестно",  
 'modified': ""  
 })  
  
 items = filter\_hidden\_files(items, self.settings.get('show\_hidden\_files'))  
 items = sort\_items(items, self.settings.get('sort\_folders\_first'))  
  
 items = [(item['name'], item['size'], item['type'], item['modified']) for item in items]  
   
 self.local\_files.set\_items(items)  
 self.local\_path.set\_path(current\_dir)  
 except Exception as e:  
 self.status\_bar.set\_status(f"Ошибка чтения локальной директории: {e}", error=True)  
  
 def \_refresh\_remote\_list(self):  
 if not self.ftp\_client.ftp:  
 return  
   
 try:  
 items = self.ftp\_client.list\_files()  
 dict\_items = [  
 {  
 'name': item[0],  
 'size': item[1],  
 'type': item[2],  
 'modified': item[3]  
 }  
 for item in items  
 ]  
 dict\_items = filter\_hidden\_files(dict\_items, self.settings.get('show\_hidden\_files'))  
 dict\_items = sort\_items(dict\_items, self.settings.get('sort\_folders\_first'))  
 items = [  
 (item['name'], item['size'], item['type'], item['modified'])  
 for item in dict\_items  
 ]  
 self.remote\_files.set\_items(items)  
 self.remote\_path.set\_path(self.ftp\_client.get\_current\_directory())  
 except Exception as e:  
 self.status\_bar.set\_status(f"Ошибка чтения удаленной директории: {e}", error=True)  
  
 def \_on\_search(self, text: str, scope: str, case\_sensitive: bool, search\_in\_folders: bool):  
 if not text:  
 self.\_refresh\_lists()  
 return  
  
 def matches\_search(name: str) -> bool:  
 if not case\_sensitive:  
 return text.lower() in name.lower()  
 return text in name  
  
 if scope in ["local", "both"]:  
 try:  
 items = []  
 current\_dir = self.settings.get('default\_local\_dir')  
 for item in os.listdir(current\_dir):  
 try:  
 path = os.path.join(current\_dir, item)  
 is\_dir = os.path.isdir(path)  
  
 if matches\_search(item) or (is\_dir and search\_in\_folders):  
 stat = os.stat(path)  
  
 if is\_dir:  
 try:  
 size = f"{len(os.listdir(path))} элем."  
 except:  
 size = "Нет доступа"  
 else:  
 size = humanize.naturalsize(stat.st\_size)  
 modified = datetime.fromtimestamp(stat.st\_mtime).strftime(  
 self.settings.get('date\_format', "%Y-%m-%d %H:%M")  
 )  
   
 items.append({  
 'name': item,  
 'size': size,  
 'type': "Папка" if is\_dir else "Файл",  
 'modified': modified  
 })  
 except Exception:  
 continue  
  
 items = filter\_hidden\_files(items, self.settings.get('show\_hidden\_files'))  
 items = sort\_items(items, self.settings.get('sort\_folders\_first'))  
  
 items = [(item['name'], item['size'], item['type'], item['modified'])   
 for item in items]  
   
 self.local\_files.set\_items(items)  
 except Exception as e:  
 self.status\_bar.set\_status(f"Ошибка поиска в локальных файлах: {e}", error=True)  
  
 if scope in ["remote", "both"] and self.ftp\_client.ftp:  
 try:  
 items = self.ftp\_client.list\_files()  
 filtered\_items = []  
   
 for item in items:  
 name, size, type\_, modified = item  
 is\_dir = type\_ == "Папка"  
   
 if matches\_search(name) or (is\_dir and search\_in\_folders):  
 filtered\_items.append({  
 'name': name,  
 'size': size,  
 'type': type\_,  
 'modified': modified  
 })  
  
 filtered\_items = filter\_hidden\_files(filtered\_items, self.settings.get('show\_hidden\_files'))  
 filtered\_items = sort\_items(filtered\_items, self.settings.get('sort\_folders\_first'))  
  
 filtered\_items = [(item['name'], item['size'], item['type'], item['modified'])   
 for item in filtered\_items]  
   
 self.remote\_files.set\_items(filtered\_items)  
 except Exception as e:  
 self.status\_bar.set\_status(f"Ошибка поиска в удаленных файлах: {e}", error=True)  
  
 total\_found = len(self.local\_files.get\_children()) + len(self.remote\_files.get\_children())  
 self.status\_bar.set\_status(f"Найдено элементов: {total\_found}")  
  
 def \_show\_quick\_connect(self):  
 QuickConnectDialog(self, self.\_connect)  
  
 def \_show\_connection\_history(self):  
 HistoryDialog(self, self.connection\_history, self.\_connect\_from\_history)  
  
 def \_connect\_from\_history(self, values):  
 host, port, user, \_ = values  
 self.connection\_panel.entries["host"].delete(0, tk.END)  
 self.connection\_panel.entries["host"].insert(0, host)  
   
 self.connection\_panel.entries["port"].delete(0, tk.END)  
 self.connection\_panel.entries["port"].insert(0, str(port))  
   
 self.connection\_panel.entries["user"].delete(0, tk.END)  
 self.connection\_panel.entries["user"].insert(0, user)  
   
 self.\_connect(host, port, user, "")  
  
 def \_show\_bookmarks(self):  
 BookmarksDialog(self, self.bookmarks,   
 self.\_connect\_from\_bookmark,  
 self.\_delete\_bookmark)  
  
 def \_connect\_from\_bookmark(self, values):  
 name, host, port, user = values  
 bookmark = next((b for b in self.bookmarks if b['name'] == name), None)  
 if not bookmark:  
 messagebox.showerror("Ошибка", "Закладка не найдена")  
 return  
   
 self.connection\_panel.entries["host"].delete(0, tk.END)  
 self.connection\_panel.entries["host"].insert(0, host)  
   
 self.connection\_panel.entries["port"].delete(0, tk.END)  
 self.connection\_panel.entries["port"].insert(0, str(port))  
   
 self.connection\_panel.entries["user"].delete(0, tk.END)  
 self.connection\_panel.entries["user"].insert(0, user)  
  
 password = self.crypto.decrypt(bookmark.get('password', ''))  
 self.connection\_panel.password\_entry.delete(0, tk.END)  
 self.connection\_panel.password\_entry.insert(0, password)  
   
 self.\_connect(host, port, user, password)  
  
 def \_add\_bookmark(self):  
 if not self.ftp\_client.ftp:  
 messagebox.showwarning("Ошибка", "Сначала подключитесь к серверу")  
 return  
   
 name = simpledialog.askstring("Закладка", "Введите название закладки:")  
 if name:  
 password = self.connection\_panel.password\_entry.get()  
 encrypted\_password = self.crypto.encrypt(password)  
   
 bookmark = {  
 'name': name,  
 'host': self.connection\_panel.entries["host"].get(),  
 'port': int(self.connection\_panel.entries["port"].get()),  
 'user': self.connection\_panel.entries["user"].get(),  
 'password': encrypted\_password  
 }  
 self.bookmarks.append(bookmark)  
 self.\_save\_bookmarks()  
 messagebox.showinfo("Успех", "Закладка добавлена")  
  
 def \_delete\_bookmark(self, name: str) -> bool:  
 if messagebox.askyesno("Подтверждение", f"Удалить закладку '{name}'?"):  
 self.bookmarks = [b for b in self.bookmarks if b['name'] != name]  
 self.\_save\_bookmarks()  
 return True  
 return False  
  
 def \_show\_settings(self):  
 settings\_window = tk.Toplevel(self)  
 settings\_window.title("Настройки")  
 settings\_window.geometry("600x500")  
 settings\_window.transient(self)  
 settings\_window.grab\_set()  
  
 notebook = ttk.Notebook(settings\_window)  
 notebook.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)  
  
 general\_frame = ttk.Frame(notebook)  
 notebook.add(general\_frame, text="Общие")  
  
 dir\_frame = ttk.LabelFrame(general\_frame, text="Директории")  
 dir\_frame.pack(fill=tk.X, padx=5, pady=5)  
  
 ttk.Label(dir\_frame, text="Локальная директория:").pack(anchor="w", padx=5, pady=2)  
 local\_dir\_frame = ttk.Frame(dir\_frame)  
 local\_dir\_frame.pack(fill=tk.X, padx=5, pady=2)  
   
 local\_dir\_entry = ttk.Entry(local\_dir\_frame)  
 local\_dir\_entry.insert(0, self.settings.get('default\_local\_dir'))  
 local\_dir\_entry.pack(side=tk.LEFT, fill=tk.X, expand=True)  
   
 def choose\_directory():  
 directory = filedialog.askdirectory(initialdir=local\_dir\_entry.get())  
 if directory:  
 local\_dir\_entry.delete(0, tk.END)  
 local\_dir\_entry.insert(0, directory)  
   
 ttk.Button(local\_dir\_frame, text="Обзор",   
 command=choose\_directory).pack(side=tk.LEFT, padx=2)  
  
 connection\_frame = ttk.LabelFrame(general\_frame, text="Подключение")  
 connection\_frame.pack(fill=tk.X, padx=5, pady=5)  
  
 auto\_reconnect\_var = tk.BooleanVar(value=self.settings.get('auto\_reconnect', True))  
 ttk.Checkbutton(connection\_frame, text="Автоматическое переподключение",  
 variable=auto\_reconnect\_var).pack(anchor="w", padx=5, pady=2)  
  
 reconnect\_frame = ttk.Frame(connection\_frame)  
 reconnect\_frame.pack(fill=tk.X, padx=5, pady=2)  
 ttk.Label(reconnect\_frame, text="Количество попыток:").pack(side=tk.LEFT)  
 reconnect\_attempts = ttk.Spinbox(reconnect\_frame, from\_=1, to=10, width=5)  
 reconnect\_attempts.set(self.settings.get('reconnect\_attempts', 3))  
 reconnect\_attempts.pack(side=tk.LEFT, padx=5)  
  
 interface\_frame = ttk.LabelFrame(general\_frame, text="Интерфейс")  
 interface\_frame.pack(fill=tk.X, padx=5, pady=5)  
  
 sort\_folders\_var = tk.BooleanVar(value=self.settings.get('sort\_folders\_first', True))  
 ttk.Checkbutton(interface\_frame, text="Показывать папки первыми",  
 variable=sort\_folders\_var).pack(anchor="w", padx=5, pady=2)  
  
 show\_hidden\_var = tk.BooleanVar(value=self.settings.get('show\_hidden\_files', False))  
 ttk.Checkbutton(interface\_frame, text="Показывать скрытые файлы",  
 variable=show\_hidden\_var).pack(anchor="w", padx=5, pady=2)  
  
 confirm\_frame = ttk.Frame(notebook)  
 notebook.add(confirm\_frame, text="Подтверждения")  
  
 confirm\_delete\_var = tk.BooleanVar(value=self.settings.get('confirm\_delete', True))  
 ttk.Checkbutton(confirm\_frame, text="Подтверждать удаление",  
 variable=confirm\_delete\_var).pack(anchor="w", padx=5, pady=2)  
  
 confirm\_overwrite\_var = tk.BooleanVar(value=self.settings.get('confirm\_overwrite', True))  
 ttk.Checkbutton(confirm\_frame, text="Подтверждать перезапись",  
 variable=confirm\_overwrite\_var).pack(anchor="w", padx=5, pady=2)  
  
 performance\_frame = ttk.Frame(notebook)  
 notebook.add(performance\_frame, text="Производительность")  
  
 ttk.Label(performance\_frame, text="Размер буфера (байт):").pack(anchor="w", padx=5, pady=2)  
 buffer\_size = ttk.Entry(performance\_frame)  
 buffer\_size.insert(0, str(self.settings.get('buffer\_size', 8192)))  
 buffer\_size.pack(anchor="w", padx=5, pady=2)  
  
 ttk.Label(performance\_frame, text="Время жизни кэша (сек):").pack(anchor="w", padx=5, pady=2)  
 cache\_ttl = ttk.Entry(performance\_frame)  
 cache\_ttl.insert(0, str(self.settings.get('cache\_ttl', 30)))  
 cache\_ttl.pack(anchor="w", padx=5, pady=2)  
  
 btn\_frame = ttk.Frame(settings\_window)  
 btn\_frame.pack(fill=tk.X, padx=5, pady=5)  
  
 def save\_settings():  
 try:  
 new\_settings = {  
 'default\_local\_dir': local\_dir\_entry.get(),  
 'buffer\_size': int(buffer\_size.get()),  
 'auto\_reconnect': auto\_reconnect\_var.get(),  
 'reconnect\_attempts': int(reconnect\_attempts.get()),  
 'cache\_ttl': int(cache\_ttl.get()),  
 'show\_hidden\_files': show\_hidden\_var.get(),  
 'confirm\_delete': confirm\_delete\_var.get(),  
 'confirm\_overwrite': confirm\_overwrite\_var.get(),  
 'sort\_folders\_first': sort\_folders\_var.get(),  
 'date\_format': self.settings.get('date\_format', "%Y-%m-%d %H:%M")  
 }  
 self.\_save\_settings(new\_settings)  
 settings\_window.destroy()  
 except ValueError as e:  
 messagebox.showerror("Ошибка", "Проверьте правильность введенных числовых значений")  
  
 ttk.Button(btn\_frame, text="Сохранить",   
 command=save\_settings).pack(side=tk.RIGHT, padx=5)  
  
 ttk.Button(btn\_frame, text="Отмена",  
 command=settings\_window.destroy).pack(side=tk.RIGHT, padx=5)  
  
 def \_save\_settings(self, new\_settings: Dict):  
 self.settings.update(new\_settings)  
 self.settings.save\_settings()  
 self.\_refresh\_lists()  
  
 def \_show\_about(self):  
 AboutDialog(self)  
  
 def \_create\_folder(self):  
 dialog = tk.Toplevel(self)  
 dialog.title("Создать папку")  
 dialog.geometry("400x330")  
 dialog.transient(self)  
 dialog.grab\_set()  
  
 dialog.geometry("+%d+%d" % (  
 self.winfo\_rootx() + self.winfo\_width()//2 - 200,  
 self.winfo\_rooty() + self.winfo\_height()//2 - 125  
 ))  
  
 main\_frame = ttk.Frame(dialog, padding="20")  
 main\_frame.pack(fill=tk.BOTH, expand=True)  
  
 input\_frame = ttk.LabelFrame(main\_frame, text="Введите имя папки", padding="10")  
 input\_frame.pack(fill=tk.X, padx=5, pady=(0, 15))  
  
 name\_entry = ttk.Entry(input\_frame, width=40)  
 name\_entry.pack(fill=tk.X, padx=5, pady=5)  
 name\_entry.focus()  
  
 radio\_frame = ttk.LabelFrame(main\_frame, text="Выберите место создания", padding="10")  
 radio\_frame.pack(fill=tk.X, padx=5, pady=(0, 15))  
  
 location = tk.StringVar(value="local")  
  
 ttk.Radiobutton(radio\_frame, text="Локально",   
 variable=location, value="local").pack(anchor=tk.W, padx=5, pady=2)  
 remote\_radio = ttk.Radiobutton(radio\_frame, text="На сервере",   
 variable=location, value="remote")  
 remote\_radio.pack(anchor=tk.W, padx=5, pady=2)  
   
 both\_radio = ttk.Radiobutton(radio\_frame, text="В обоих местах",   
 variable=location, value="both")  
 both\_radio.pack(anchor=tk.W, padx=5, pady=2)  
  
 if not self.ftp\_client.ftp:  
 remote\_radio.configure(state="disabled")  
 both\_radio.configure(state="disabled")  
  
 def create():  
 dirname = name\_entry.get().strip()  
 if not dirname:  
 messagebox.showwarning("Ошибка", "Введите имя папки")  
 return  
  
 loc = location.get()  
 if loc in ("local", "both"):  
 try:  
 path = os.path.join(self.settings.get('default\_local\_dir'), dirname)  
 os.makedirs(path, exist\_ok=True)  
 self.\_refresh\_local\_list()  
 self.status\_bar.set\_status(f"Создана локальная папка: {dirname}")  
 except Exception as e:  
 self.status\_bar.set\_status(f"Ошибка создания локальной папки: {e}", error=True)  
  
 if loc in ("remote", "both") and self.ftp\_client.ftp:  
 try:  
 self.ftp\_client.ftp.mkd(dirname)  
 self.\_refresh\_remote\_list()  
 self.status\_bar.set\_status(f"Создана удаленная папка: {dirname}")  
 except Exception as e:  
 self.status\_bar.set\_status(f"Ошибка создания удаленной папки: {e}", error=True)  
  
 dialog.destroy()  
  
 btn\_frame = ttk.Frame(main\_frame)  
 btn\_frame.pack(fill=tk.X, pady=(0, 5))  
  
 center\_frame = ttk.Frame(btn\_frame)  
 center\_frame.pack(anchor=tk.CENTER)  
   
 ttk.Button(center\_frame, text="Создать",   
 command=create, width=15).pack(side=tk.LEFT, padx=5)  
 ttk.Button(center\_frame, text="Отмена",  
 command=dialog.destroy, width=15).pack(side=tk.LEFT, padx=5)  
  
 def \_upload\_files(self):  
 if not self.ftp\_client.ftp:  
 messagebox.showwarning("Ошибка", "Сначала подключитесь к серверу")  
 return  
  
 selected = self.local\_files.selection()  
 if not selected:  
 messagebox.showwarning("Ошибка", "Выберите файлы для загрузки")  
 return  
  
 try:  
 total = len(selected)  
 for i, item\_id in enumerate(selected, 1):  
 values = self.local\_files.item(item\_id)['values']  
 filename = str(values[0])  
 is\_dir = values[2] == "Папка"  
 local\_path = os.path.join(self.settings.get('default\_local\_dir'), filename)  
  
 if not is\_dir:  
 try:  
 self.ftp\_client.ftp.size(filename)  
 if self.settings.get('confirm\_overwrite', True):  
 if not messagebox.askyesno("Подтверждение",   
 f"Файл {filename} уже существует. Перезаписать?"):  
 continue  
 except:  
 pass  
  
 if is\_dir:  
 initial\_remote\_dir = self.ftp\_client.ftp.pwd()  
   
 try:  
 try:  
 self.ftp\_client.ftp.mkd(filename)  
 except:  
 pass   
 self.ftp\_client.ftp.cwd(filename)  
  
 for root, dirs, files in os.walk(local\_path):  
 rel\_path = os.path.relpath(root, local\_path)  
   
 if rel\_path != '.':  
 remote\_path\_parts = [str(part) for part in rel\_path.split(os.sep)]  
 for part in remote\_path\_parts:  
 try:  
 self.ftp\_client.ftp.mkd(part)  
 except:  
 pass  
 self.ftp\_client.ftp.cwd(part)  
  
 for file in files:  
 local\_file = os.path.join(root, file)  
 try:  
 self.ftp\_client.ftp.size(str(file))  
 if self.settings.get('confirm\_overwrite', True):  
 if not messagebox.askyesno("Подтверждение",   
 f"Файл {file} уже существует. Перезаписать?"):  
 continue  
 except:  
 pass  
  
 with open(local\_file, 'rb') as f:  
 self.ftp\_client.ftp.storbinary(f'STOR {str(file)}', f)  
   
 progress = (i / total) \* 100  
 self.status\_bar.set\_progress(progress)  
 self.status\_bar.set\_status(f"Загружен файл: {file}")  
  
 if rel\_path != '.':  
 for \_ in remote\_path\_parts:  
 self.ftp\_client.ftp.cwd('..')  
  
 self.ftp\_client.ftp.cwd(initial\_remote\_dir)  
   
 except Exception as e:  
 self.ftp\_client.ftp.cwd(initial\_remote\_dir)  
 raise e  
   
 else:  
 with open(local\_path, 'rb') as f:  
 self.ftp\_client.ftp.storbinary(f'STOR {str(filename)}', f)  
 progress = (i / total) \* 100  
 self.status\_bar.set\_progress(progress)  
 self.status\_bar.set\_status(f"Загружен файл: {filename}")  
  
 self.\_refresh\_remote\_list()  
 self.status\_bar.set\_status("Загрузка завершена")  
 self.status\_bar.set\_progress(100)  
  
 except Exception as e:  
 self.status\_bar.set\_status(f"Ошибка загрузки: {str(e)}", error=True)  
 messagebox.showerror("Ошибка", f"Ошибка загрузки: {str(e)}")  
  
 def \_download\_files(self):  
 if not self.ftp\_client.ftp:  
 messagebox.showwarning("Ошибка", "Сначала подключитесь к серверу")  
 return  
  
 selected = self.remote\_files.selection()  
 if not selected:  
 messagebox.showwarning("Ошибка", "Выберите файлы для скачивания")  
 return  
  
 try:  
 total = len(selected)  
 for i, item\_id in enumerate(selected, 1):  
 values = self.remote\_files.item(item\_id)['values']  
 filename = str(values[0])  
 is\_dir = values[2] == "Папка"  
 local\_path = os.path.join(self.settings.get('default\_local\_dir'), filename)  
   
 if os.path.exists(local\_path):  
 if self.settings.get('confirm\_overwrite', True):  
 if not messagebox.askyesno("Подтверждение",   
 f"Файл {filename} уже существует. Перезаписать?"):  
 continue  
  
 if is\_dir:  
 os.makedirs(local\_path, exist\_ok=True)  
 current\_remote = self.ftp\_client.ftp.pwd()  
 try:  
 self.ftp\_client.ftp.cwd(filename)  
 for item in self.ftp\_client.list\_files():  
 name = str(item[0])  
 item\_type = item[2]  
 item\_path = os.path.join(local\_path, name)  
   
 if item\_type == "Папка":  
 os.makedirs(item\_path, exist\_ok=True)  
 sub\_remote = self.ftp\_client.ftp.pwd()  
 try:  
 self.ftp\_client.ftp.cwd(name)  
 for sub\_item in self.ftp\_client.list\_files():  
 sub\_name = str(sub\_item[0])  
 if sub\_item[2] == "Файл":  
 sub\_path = os.path.join(item\_path, sub\_name)  
 with open(sub\_path, 'wb') as f:  
 self.ftp\_client.ftp.retrbinary(f'RETR {sub\_name}', f.write)  
 finally:  
 self.ftp\_client.ftp.cwd(sub\_remote)  
 else:  
 with open(item\_path, 'wb') as f:  
 self.ftp\_client.ftp.retrbinary(f'RETR {name}', f.write)  
 finally:  
 self.ftp\_client.ftp.cwd(current\_remote)  
 else:  
 with open(local\_path, 'wb') as f:  
 self.ftp\_client.ftp.retrbinary(f'RETR {str(filename)}', f.write)  
  
 progress = (i / total) \* 100  
 self.status\_bar.set\_progress(progress)  
 self.status\_bar.set\_status(f"Скачано {i}/{total}: {filename}")  
  
 self.\_refresh\_local\_list()  
 self.status\_bar.set\_status("Скачивание завершено")  
 self.status\_bar.set\_progress(100)  
  
 except Exception as e:  
 self.status\_bar.set\_status(f"Ошибка скачивания: {str(e)}", error=True)  
 messagebox.showerror("Ошибка", f"Ошибка скачивания: {str(e)}")  
  
 def \_delete\_selected(self):  
 if self.local\_files.focus():  
 self.\_delete\_local()  
 elif self.remote\_files.focus() and self.ftp\_client.ftp:  
 self.\_delete\_remote()  
 return "break"   
  
 def \_on\_local\_double\_click(self, event):  
 item = self.local\_files.identify('item', event.x, event.y)  
 if not item:  
 return  
  
 values = self.local\_files.item(item)['values']  
 if not values:  
 return  
  
 filename = str(values[0])  
 is\_dir = values[2] == "Папка"  
  
 if is\_dir:  
 try:  
 new\_path = os.path.join(self.settings.get('default\_local\_dir'), filename)  
 if os.path.exists(new\_path) and os.path.isdir(new\_path):  
 self.settings.set('default\_local\_dir', new\_path)  
 self.\_refresh\_local\_list()  
 self.status\_bar.set\_status(f"Текущая локальная директория: {new\_path}")  
 except Exception as e:  
 self.status\_bar.set\_status(f"Ошибка перехода в папку: {str(e)}", error=True)  
 else:  
 try:  
 import subprocess  
 import sys  
 path = os.path.join(self.settings.get('default\_local\_dir'), filename)  
 if sys.platform == 'darwin':  
 subprocess.run(['open', path])  
 elif sys.platform == 'win32':  
 os.startfile(path)  
 else:  
 subprocess.run(['xdg-open', path])  
 except Exception as e:  
 self.status\_bar.set\_status(f"Ошибка открытия файла: {str(e)}", error=True)  
  
 def \_on\_remote\_double\_click(self, event):  
 if not self.ftp\_client.ftp:  
 return  
  
 item = self.remote\_files.identify('item', event.x, event.y)  
 if not item:  
 return  
  
 values = self.remote\_files.item(item)['values']  
 if not values:  
 return  
  
 filename = str(values[0])  
 is\_dir = values[2] == "Папка"  
  
 if is\_dir:  
 try:  
 self.ftp\_client.ftp.cwd(filename)  
 self.\_refresh\_remote\_list()  
 except Exception as e:  
 self.status\_bar.set\_status(f"Ошибка перехода в папку: {e}", error=True)  
  
 def \_toggle\_fullscreen(self, event=None):  
 if sys.platform == 'darwin':  
 is\_fullscreen = self.attributes('-fullscreen')  
 self.attributes('-fullscreen', not is\_fullscreen)  
 elif sys.platform == 'win32':  
 is\_zoomed = self.state() == 'zoomed'  
 self.state('normal' if is\_zoomed else 'zoomed')  
 else:  
 is\_zoomed = self.attributes('-zoomed')  
 self.attributes('-zoomed', not is\_zoomed)  
  
 def \_change\_local\_directory(self, path: str):  
 self.settings.set('default\_local\_dir', path)  
 self.settings.save\_settings()  
 self.\_refresh\_local\_list()  
  
 def \_navigate\_up\_local(self):  
 current\_dir = self.settings.get('default\_local\_dir')  
 parent\_dir = os.path.dirname(current\_dir)  
 if os.path.exists(parent\_dir) and parent\_dir != current\_dir:  
 self.settings.set('default\_local\_dir', parent\_dir)  
 self.\_refresh\_local\_list()  
 self.status\_bar.set\_status(f"Текущая локальная директория: {parent\_dir}")  
  
 def \_navigate\_up\_remote(self):  
 if self.ftp\_client.ftp:  
 try:  
 self.ftp\_client.ftp.cwd('..')  
 current\_dir = self.ftp\_client.ftp.pwd()  
 self.\_refresh\_remote\_list()  
 self.status\_bar.set\_status(f"Текущая удаленная директория: {current\_dir}")  
 except Exception as e:  
 self.status\_bar.set\_status(f"Ошибка перехода: {str(e)}", error=True)  
  
 def \_navigate\_up(self):  
 if self.local\_files.focus():  
 self.\_navigate\_up\_local()  
 elif self.remote\_files.focus() and self.ftp\_client.ftp:  
 self.\_navigate\_up\_remote()  
  
 def \_on\_drag\_start(self, event):  
 tree = event.widget  
 item = tree.identify\_row(event.y)  
 if item:  
 tree.selection\_set(item)  
 self.\_drag\_data = {  
 'source': tree,  
 'item': item,  
 'start\_x': event.x,  
 'start\_y': event.y  
 }  
  
 def \_on\_drag\_motion(self, event):  
 if hasattr(self, '\_drag\_data'):  
 pass  
  
 def \_on\_drag\_end(self, event):  
 if not hasattr(self, '\_drag\_data'):  
 return  
 source = self.\_drag\_data['source']  
 target = event.widget  
 if source != target:  
 if source == self.local\_files and target == self.remote\_files:  
 self.\_upload\_selected()  
 elif source == self.remote\_files and target == self.local\_files:  
 self.\_download\_selected()  
  
 del self.\_drag\_data  
  
 def \_upload\_selected(self):  
 if not self.ftp\_client.ftp:  
 messagebox.showwarning("Ошибка", "Сначала подключитесь к серверу")  
 return  
  
 selected = self.local\_files.selection()  
 if not selected:  
 return  
  
 def upload\_thread():  
 try:  
 total = len(selected)  
 for i, item\_id in enumerate(selected, 1):  
 values = self.local\_files.item(item\_id)['values']  
 filename = str(values[0])  
 is\_dir = values[2] == "Папка"  
 local\_path = os.path.join(self.settings.get('default\_local\_dir'), filename)  
 try:  
 if is\_dir:  
 self.ftp\_client.ftp.cwd(filename)  
 self.ftp\_client.ftp.cwd('..')  
 else:  
 self.ftp\_client.ftp.size(filename)  
   
 if self.settings.get('confirm\_overwrite', True):  
 confirm\_event = threading.Event()  
 self.schedule\_update(lambda: [  
 setattr(confirm\_event, 'result',  
 messagebox.askyesno("Подтверждение",  
 f"Файл {filename} уже существует. Перезаписать?"))  
 ])  
 confirm\_event.wait()  
 if not confirm\_event.result:  
 continue  
 except:  
 pass  
  
 progress = (i - 1) / total \* 100  
 self.schedule\_update(lambda p=progress, f=filename: [  
 self.status\_bar.set\_progress(p),  
 self.status\_bar.set\_status(f"Загрузка {i}/{total}: {f}")  
 ])  
  
 if is\_dir:  
 success, message = self.ftp\_client.upload\_folder(local\_path, filename)  
 else:  
 success, message = self.ftp\_client.upload\_file(local\_path, filename)  
  
 if not success:  
 self.schedule\_update(lambda: [  
 messagebox.showerror("Ошибка", f"Ошибка загрузки {filename}: {message}"),  
 self.status\_bar.set\_status(f"Ошибка загрузки: {message}", error=True)  
 ])  
 return  
  
 self.schedule\_update(lambda: [  
 self.\_refresh\_remote\_list(),  
 self.status\_bar.set\_status("Загрузка завершена"),  
 self.status\_bar.set\_progress(100)  
 ])  
  
 except Exception as e:  
 self.schedule\_update(lambda: [  
 messagebox.showerror("Ошибка", f"Ошибка загрузки: {str(e)}"),  
 self.status\_bar.set\_status(f"Ошибка загрузки: {str(e)}", error=True)  
 ])  
  
 Thread(target=upload\_thread, daemon=True).start()  
  
 def \_download\_selected(self):  
 if not self.ftp\_client.ftp:  
 messagebox.showwarning("Ошибка", "Сначала подключитесь к серверу")  
 return  
   
 selected = self.remote\_files.selection()  
 if not selected:  
 return  
   
 def download\_thread():  
 try:  
 total = len(selected)  
 for i, item\_id in enumerate(selected, 1):  
 values = self.remote\_files.item(item\_id)['values']  
 filename = str(values[0])  
 is\_dir = values[2] == "Папка"  
 local\_path = os.path.join(self.settings.get('default\_local\_dir'), filename)  
 if os.path.exists(local\_path):  
 if self.settings.get('confirm\_overwrite', True):  
 confirm\_event = threading.Event()  
 self.schedule\_update(lambda: [  
 setattr(confirm\_event, 'result',  
 messagebox.askyesno("Подтверждение",  
 f"Файл {filename} уже существует. Перезаписать?"))  
 ])  
 confirm\_event.wait()  
 if not confirm\_event.result:  
 continue  
  
 progress = (i - 1) / total \* 100  
 self.schedule\_update(lambda p=progress, f=filename: [  
 self.status\_bar.set\_progress(p),  
 self.status\_bar.set\_status(f"Скачивание {i}/{total}: {f}")  
 ])  
  
 if is\_dir:  
 os.makedirs(local\_path, exist\_ok=True)  
 current\_remote = self.ftp\_client.ftp.pwd()  
 try:  
 self.ftp\_client.ftp.cwd(filename)  
 for item in self.ftp\_client.list\_files():  
 name = str(item[0])  
 item\_type = item[2]  
 if item\_type == "Файл":  
 local\_file = os.path.join(local\_path, name)  
 with open(local\_file, 'wb') as f:  
 self.ftp\_client.ftp.retrbinary(f'RETR {name}', f.write)  
 finally:  
 self.ftp\_client.ftp.cwd(current\_remote)  
 else:  
 success, message = self.ftp\_client.download\_file(filename, local\_path)  
 if not success:  
 self.schedule\_update(lambda: [  
 messagebox.showerror("Ошибка", f"Ошибка скачивания {filename}: {message}"),  
 self.status\_bar.set\_status(f"Ошибка скачивания: {message}", error=True)  
 ])  
 return  
  
 self.schedule\_update(lambda: [  
 self.\_refresh\_local\_list(),  
 self.status\_bar.set\_status("Скачивание завершено"),  
 self.status\_bar.set\_progress(100)  
 ])  
  
 except Exception as e:  
 self.schedule\_update(lambda: [  
 messagebox.showerror("Ошибка", f"Ошибка скачивания: {str(e)}"),  
 self.status\_bar.set\_status(f"Ошибка скачивания: {str(e)}", error=True)  
 ])  
  
 Thread(target=download\_thread, daemon=True).start()  
  
 def \_delete\_local(self):  
 selected = self.local\_files.selection()  
 if not selected:  
 return  
   
 if not messagebox.askyesno("Подтверждение",   
 "Вы действительно хотите удалить выбранные файлы?"):  
 return  
   
 try:  
 for item\_id in selected:  
 values = self.local\_files.item(item\_id)['values']  
 filename = str(values[0])  
 path = os.path.join(self.settings.get('default\_local\_dir'), filename)  
   
 if os.path.exists(path):  
 if os.path.isdir(path):  
 shutil.rmtree(path)  
 else:  
 os.remove(path)  
   
 self.\_refresh\_local\_list()  
 self.status\_bar.set\_status("Удаление завершено")  
 except Exception as e:  
 self.status\_bar.set\_status(f"Ошибка удаления: {str(e)}", error=True)  
 messagebox.showerror("Ошибка", f"Не удалось удалить файл(ы): {str(e)}")  
  
 def \_delete\_remote(self):  
 if not self.ftp\_client.ftp:  
 return  
   
 selected = self.remote\_files.selection()  
 if not selected:  
 return  
   
 confirm = messagebox.askyesno(  
 "Подтверждение",  
 "Вы действительно хотите удалить выбранные файлы с сервера?"  
 )  
 if not confirm:  
 return  
   
 try:  
 for item\_id in selected:  
 values = self.remote\_files.item(item\_id)['values']  
 filename = str(values[0])  
 is\_dir = values[2] == "Папка"  
   
 if is\_dir:  
 success, message = self.ftp\_client.delete\_directory\_recursive(filename)  
 else:  
 success, message = self.ftp\_client.delete\_item(filename)  
   
 if not success:  
 if message == "NOT\_EMPTY\_DIR":  
 if messagebox.askyesno("Подтверждение",   
 f"Папка '{filename}' не пуста. Удалить её содержимое?"):  
 success, message = self.ftp\_client.delete\_directory\_recursive(filename)  
 if not success:   
 messagebox.showerror("Ошибка", f"Не удалось удалить {filename}: {message}")  
   
 self.\_refresh\_remote\_list()  
 self.status\_bar.set\_status("Удаление завершено")  
 except Exception as e:  
 self.status\_bar.set\_status(f"Ошибка удаления: {str(e)}", error=True)  
 messagebox.showerror("Ошибка", f"Не удалось удалить файл(ы): {str(e)}")  
  
 def \_on\_closing(self):  
 if not messagebox.askyesno("Подтверждение", "Вы действительно хотите выйти из программы?"):  
 return  
   
 debug\_log("\nDEBUG: Начало закрытия приложения")  
   
 try:  
 debug\_log("DEBUG: Останавливаем мониторинг статистики")  
 self.stats\_panel.stop\_monitoring()  
   
 debug\_log("DEBUG: Отключаемся от FTP сервера")  
 self.ftp\_client.disconnect()  
   
 debug\_log("DEBUG: Сохраняем настройки")  
 self.settings.save\_settings()  
   
 debug\_log("DEBUG: Закрытие приложения завершено")  
 except Exception as e:  
 debug\_log(f"DEBUG: Ошибка при закрытии приложения: {str(e)}")  
 finally:  
 self.quit()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app = Application()  
 app.mainloop()

from ftplib import FTP, FTP\_TLS, error\_perm  
import os  
from threading import Lock, Thread  
from typing import Optional, Tuple, List, Dict, Any, Callable  
from datetime import datetime, timezone  
import humanize  
import time  
from queue import Queue  
import socket  
import sys  
from src.core.settings import Settings  
  
  
def debug\_log(message: str):  
 *"""Логирование отладочной информации"""* print(message, file=sys.stderr, flush=True)  
  
  
class FTPClient:  
 def \_\_init\_\_(self):  
 self.ftp = None  
 self.ftp\_lock = Lock()  
 self.monitor\_thread = None  
 self.stop\_monitor = False  
 self.settings = Settings()  
 self.current\_remote\_dir = "/"  
 self.connection\_params = None  
 self.monitor\_running = False  
 self.remote\_cache = {}  
  
 def connect(self, host: str, port: int, user: str, password: str) -> Tuple[bool, str]:  
 debug\_log("\nDEBUG: FTPClient: Начало подключения")  
 try:  
 with self.ftp\_lock:  
 # Если есть активное подключение, сначала отключаемся  
 if self.ftp:  
 debug\_log("DEBUG: FTPClient: Закрываем предыдущее подключение")  
 try:  
 self.ftp.quit()  
 except:  
 pass  
 self.ftp = None  
  
 debug\_log("DEBUG: FTPClient: Создаем новое подключение")  
 self.ftp = FTP()  
 self.ftp.connect(host, port)  
 self.ftp.login(user, password)  
 self.ftp.encoding = self.settings.get('encoding', 'utf-8')  
  
 self.connection\_params = {  
 'host': host,  
 'port': port,  
 'user': user,  
 'password': password  
 }  
  
 debug\_log("DEBUG: FTPClient: Подключение успешно установлено")  
 return True, "Успешное подключение"  
 except Exception as e:  
 debug\_log(f"DEBUG: FTPClient: Ошибка подключения: {str(e)}")  
 # Очищаем объект FTP при ошибке  
 self.ftp = None  
 self.connection\_params = None  
 return False, str(e)  
  
 def reconnect(self) -> Tuple[bool, str]:  
 if not self.connection\_params:  
 return False, "Нет сохраненных параметров подключения"  
  
 return self.connect(\*\*self.connection\_params)  
  
 def disconnect(self) -> None:  
 debug\_log("\nDEBUG: FTPClient: Начало отключения")  
 self.monitor\_running = False  
  
 with self.ftp\_lock:  
 if self.ftp:  
 try:  
 debug\_log("DEBUG: FTPClient: Отправляем команду QUIT")  
 self.ftp.quit()  
 except Exception as e:  
 debug\_log(f"DEBUG: FTPClient: Ошибка при отправке QUIT: {str(e)}")  
 finally:  
 debug\_log("DEBUG: FTPClient: Очищаем объект FTP")  
 self.ftp = None  
 self.connection\_params = None  
 debug\_log("DEBUG: FTPClient: Отключение завершено")  
  
 def \_get\_file\_list(self) -> List[Tuple[str, bool]]:  
 file\_list = []  
 self.ftp.retrlines('LIST', file\_list.append)  
 items = []  
 for line in file\_list:  
 parts = line.split(maxsplit=8)  
 if len(parts) < 9:  
 continue  
 perm = parts[0]  
 name = parts[8].strip()  
 if name in ('.', '..'):  
 continue  
 is\_dir = perm.startswith('d')  
 items.append((name, is\_dir))  
 return items  
  
 def list\_files(self) -> List[Tuple[str, str, str, str]]:  
 if not self.ftp:  
 return []  
  
 items = []  
 try:  
 with self.ftp\_lock:  
 file\_list = []  
 self.ftp.retrlines('LIST', file\_list.append)  
  
 for line in file\_list:  
 try:  
 parts = line.split(maxsplit=8)  
 if len(parts) < 9:  
 continue  
  
 name = parts[8].strip()  
 if name in ('.', '..'):  
 continue  
  
 perm = parts[0]  
 is\_dir = perm.startswith('d')  
  
 if is\_dir:  
 try:  
 current\_dir = self.ftp.pwd()  
 self.ftp.cwd(name)  
 dir\_items = []  
 self.ftp.retrlines('NLST', dir\_items.append)  
 size = f"{len(dir\_items)} элем."  
 self.ftp.cwd(current\_dir)  
 except:  
 size = "Нет доступа"  
 else:  
 try:  
 size = humanize.naturalsize(int(parts[4]))  
 except:  
 size = parts[4]  
  
 time\_str = ' '.join(parts[5:8])  
 modified = self.\_parse\_ftp\_time(time\_str)  
  
 items.append((name, size, "Папка" if is\_dir else "Файл", modified))  
 except:  
 continue  
 except:  
 pass  
  
 return items  
  
 def download\_file(self, remote\_file: str, local\_path: str,  
 progress\_callback=None) -> Tuple[bool, str]:  
 if not self.ftp:  
 return False, "Нет подключения"  
  
 try:  
 file\_size = self.ftp.size(remote\_file)  
 buffer\_size = self.settings.get('buffer\_size', 8192)  
  
 with self.ftp\_lock:  
 with open(local\_path, 'wb') as f:  
 bytes\_received = 0  
  
 def callback(block):  
 nonlocal bytes\_received  
 bytes\_received += len(block)  
 f.write(block)  
 if progress\_callback:  
 progress\_callback(bytes\_received, file\_size)  
  
 self.ftp.retrbinary(f'RETR {remote\_file}', callback, buffer\_size)  
  
 downloaded\_size = os.path.getsize(local\_path)  
 if downloaded\_size != file\_size:  
 os.remove(local\_path)  
 return False, "Ошибка скачивания: размер файла не совпадает"  
  
 return True, "Файл успешно скачан"  
  
 except Exception as e:  
 if os.path.exists(local\_path):  
 os.remove(local\_path)  
 return False, str(e)  
  
 def upload\_file(self, local\_path: str, remote\_file: str,  
 progress\_callback=None) -> Tuple[bool, str]:  
 if not self.ftp:  
 return False, "Нет подключения"  
  
 try:  
 file\_size = os.path.getsize(local\_path)  
 buffer\_size = self.settings.get('buffer\_size', 8192)  
  
 with self.ftp\_lock:  
 with open(local\_path, 'rb') as f:  
 bytes\_sent = 0  
  
 def callback(block):  
 nonlocal bytes\_sent  
 bytes\_sent += len(block)  
 if progress\_callback:  
 progress\_callback(bytes\_sent, file\_size)  
 return block  
  
 self.ftp.storbinary(f'STOR {remote\_file}', f, buffer\_size, callback)  
  
 uploaded\_size = self.ftp.size(remote\_file)  
 if uploaded\_size != file\_size:  
 self.ftp.delete(remote\_file)  
 return False, "Ошибка загрузки: размер файла не совпадает"  
  
 return True, "Файл успешно загружен"  
  
 except Exception as e:  
 return False, str(e)  
  
 def upload\_folder(self, local\_path: str, remote\_folder: str,  
 progress\_callback=None) -> Tuple[bool, str]:  
 try:  
 with self.ftp\_lock:  
 try:  
 self.ftp.mkd(remote\_folder)  
 except:  
 pass  
  
 current\_remote = self.ftp.pwd()  
 self.ftp.cwd(remote\_folder)  
  
 for item in os.listdir(local\_path):  
 local\_item\_path = os.path.join(local\_path, item)  
  
 if os.path.isfile(local\_item\_path):  
 success, message = self.upload\_file(local\_item\_path, item, progress\_callback)  
 if not success:  
 return False, f"Ошибка загрузки {item}: {message}"  
 elif os.path.isdir(local\_item\_path):  
 success, message = self.upload\_folder(local\_item\_path, item, progress\_callback)  
 if not success:  
 return False, message  
  
 self.ftp.cwd(current\_remote)  
 return True, "Папка успешно загружена"  
  
 except Exception as e:  
 return False, str(e)  
  
 def create\_directory(self, dirname: str) -> Tuple[bool, str]:  
 if not self.ftp:  
 return False, "Нет подключения"  
  
 try:  
 dirname = dirname.strip()  
 if not dirname:  
 return False, "Пустое имя директории"  
  
 with self.ftp\_lock:  
 items = self.\_get\_file\_list()  
 for name, is\_dir in items:  
 if name == dirname and is\_dir:  
 return False, f"Папка '{dirname}' уже существует"  
  
 try:  
 self.ftp.mkd(dirname)  
 return True, f"Папка '{dirname}' создана"  
 except error\_perm as e:  
 if '550' in str(e):  
 return False, f"Папка '{dirname}' не может быть создана: {str(e)}"  
 else:  
 return False, f"Ошибка создания папки: {str(e)}"  
 except Exception as e:  
 return False, f"Ошибка создания папки: {str(e)}"  
 except Exception as e:  
 return False, str(e)  
  
 def delete\_item(self, name: str) -> Tuple[bool, str]:  
 *"""Удаление файла или пустой директории"""* if not self.ftp:  
 return False, "Нет подключения"  
  
 try:  
 name = name.strip()  
 if not name:  
 return False, "Пустое имя"  
  
 debug\_log(f"\nDEBUG: Начинаем удаление элемента {name}")  
  
 with self.ftp\_lock:  
 current\_dir = self.ftp.pwd()  
 debug\_log(f"DEBUG: Текущая директория: {current\_dir}")  
   
 # Определяем тип элемента  
 try:  
 debug\_log(f"DEBUG: Проверяем, является ли {name} директорией")  
 self.ftp.cwd(name)  
 is\_dir = True  
 self.ftp.cwd(current\_dir)  
 debug\_log("DEBUG: Элемент является директорией")  
 except:  
 is\_dir = False  
 debug\_log("DEBUG: Элемент является файлом")  
  
 if is\_dir:  
 try:  
 debug\_log(f"DEBUG: Проверяем содержимое директории {name}")  
 self.ftp.cwd(name)  
 dir\_items = []  
 self.ftp.retrlines('NLST', dir\_items.append)  
 self.ftp.cwd(current\_dir)  
  
 dir\_items = [item for item in dir\_items if item not in ('.', '..')]  
 debug\_log(f"DEBUG: Найдено элементов в директории: {len(dir\_items)}")  
   
 if dir\_items:  
 debug\_log("DEBUG: Директория не пуста")  
 return False, "NOT\_EMPTY\_DIR"  
  
 debug\_log(f"DEBUG: Пытаемся удалить пустую директорию {name}")  
 self.ftp.rmd(name)  
 debug\_log(f"DEBUG: Директория {name} успешно удалена")  
 return True, f"Папка '{name}' удалена"  
 except Exception as e:  
 error\_msg = str(e)  
 debug\_log(f"DEBUG: Ошибка при удалении директории: {error\_msg}")  
 if "550" in error\_msg:  
 debug\_log("DEBUG: Ошибка 550 - возможно, нет прав на удаление")  
 return False, f"Ошибка удаления папки: {error\_msg}"  
 else:  
 try:  
 debug\_log(f"DEBUG: Пытаемся удалить файл {name}")  
 self.ftp.delete(name)  
 debug\_log(f"DEBUG: Файл {name} успешно удален")  
 return True, f"Файл '{name}' удален"  
 except Exception as e:  
 error\_msg = str(e)  
 debug\_log(f"DEBUG: Ошибка при удалении файла: {error\_msg}")  
 if "550" in error\_msg:  
 debug\_log("DEBUG: Ошибка 550 - возможно, нет прав на удаление")  
 return False, f"Ошибка удаления файла: {error\_msg}"  
 except Exception as e:  
 debug\_log(f"DEBUG: Критическая ошибка: {str(e)}")  
 return False, str(e)  
  
 def delete\_directory\_recursive(self, dirname: str) -> Tuple[bool, str]:  
 *"""Рекурсивное удаление директории и всего её содержимого"""* if not self.ftp:  
 return False, "Нет подключения"  
  
 debug\_log(f"\nDEBUG: Начинаем удаление директории {dirname}")  
  
 try:  
 with self.ftp\_lock:  
 # Сохраняем текущую директорию  
 current\_dir = self.ftp.pwd()  
 debug\_log(f"DEBUG: Текущая директория: {current\_dir}")  
   
 # Переходим в удаляемую директорию  
 try:  
 debug\_log(f"DEBUG: Пытаемся перейти в директорию {dirname}")  
 self.ftp.cwd(dirname)  
 current\_path = self.ftp.pwd()  
 debug\_log(f"DEBUG: Успешно перешли. Текущий путь: {current\_path}")  
 except Exception as e:  
 debug\_log(f"DEBUG: Ошибка при переходе в директорию: {str(e)}")  
 return False, f"Не удалось перейти в директорию {dirname}: {str(e)}"  
  
 # Получаем список файлов  
 try:  
 files = []  
 def list\_callback(line):  
 debug\_log(f"DEBUG: Получена строка листинга: {line}")  
 files.append(line)  
   
 debug\_log("DEBUG: Запрашиваем список файлов")  
 self.ftp.retrlines('LIST', list\_callback)  
 debug\_log(f"DEBUG: Получен список файлов: {len(files)} элементов")  
 except Exception as e:  
 debug\_log(f"DEBUG: Ошибка при получении списка файлов: {str(e)}")  
 self.ftp.cwd(current\_dir)  
 return False, f"Не удалось получить список файлов: {str(e)}"  
  
 # Обрабатываем каждый элемент  
 for file\_info in files:  
 debug\_log(f"DEBUG: Обработка элемента: {file\_info}")  
   
 # Пропускаем . и ..  
 if file\_info.endswith('.') or file\_info.endswith('..'):  
 debug\_log("DEBUG: Пропускаем специальную директорию")  
 continue  
  
 # Разбираем информацию о файле  
 parts = file\_info.split(maxsplit=8)  
 if len(parts) < 9:  
 debug\_log("DEBUG: Некорректный формат строки листинга")  
 continue  
  
 name = parts[8]  
 is\_dir = file\_info.startswith('d')  
 debug\_log(f"DEBUG: Обнаружен {'каталог' if is\_dir else 'файл'}: {name}")  
  
 try:  
 if is\_dir:  
 debug\_log(f"DEBUG: Удаляем содержимое поддиректории: {name}")  
 # Сначала переходим в поддиректорию  
 self.ftp.cwd(name)  
   
 # Удаляем все файлы в поддиректории  
 subfiles = []  
 self.ftp.retrlines('LIST', subfiles.append)  
 for subfile\_info in subfiles:  
 if subfile\_info.endswith('.') or subfile\_info.endswith('..'):  
 continue  
 subparts = subfile\_info.split(maxsplit=8)  
 if len(subparts) >= 9:  
 subname = subparts[8]  
 if subfile\_info.startswith('d'):  
 debug\_log(f"DEBUG: Рекурсивно удаляем поддиректорию: {subname}")  
 self.delete\_directory\_recursive(subname)  
 else:  
 debug\_log(f"DEBUG: Удаляем файл в поддиректории: {subname}")  
 self.ftp.delete(subname)  
   
 # Возвращаемся в родительскую директорию  
 self.ftp.cwd('..')  
   
 # Удаляем саму поддиректорию  
 debug\_log(f"DEBUG: Удаляем пустую поддиректорию: {name}")  
 self.ftp.rmd(name)  
 debug\_log(f"DEBUG: Поддиректория {name} успешно удалена")  
 else:  
 debug\_log(f"DEBUG: Удаляем файл: {name}")  
 self.ftp.delete(name)  
 debug\_log(f"DEBUG: Файл {name} успешно удален")  
 except Exception as e:  
 debug\_log(f"DEBUG: Ошибка при обработке элемента {name}: {str(e)}")  
 self.ftp.cwd(current\_dir)  
 return False, f"Ошибка при удалении {name}: {str(e)}"  
  
 # Возвращаемся в исходную директорию  
 debug\_log(f"DEBUG: Возвращаемся в исходную директорию: {current\_dir}")  
 self.ftp.cwd(current\_dir)  
  
 # Удаляем саму директорию  
 try:  
 debug\_log(f"DEBUG: Удаляем исходную директорию: {dirname}")  
 self.ftp.rmd(dirname)  
 debug\_log(f"DEBUG: Директория {dirname} успешно удалена")  
 return True, f"Папка '{dirname}' и её содержимое удалены"  
 except Exception as e:  
 error\_msg = str(e)  
 debug\_log(f"DEBUG: Ошибка при удалении директории: {error\_msg}")  
 if "550" in error\_msg:  
 debug\_log("DEBUG: Ошибка 550 - возможно, директория не пуста или нет прав")  
 return False, f"Не удалось удалить директорию {dirname}: {error\_msg}"  
   
 except Exception as e:  
 debug\_log(f"DEBUG: Критическая ошибка: {str(e)}")  
 return False, str(e)  
  
 def rename\_item(self, old\_name: str, new\_name: str) -> Tuple[bool, str]:  
 if not self.ftp:  
 return False, "Нет подключения"  
  
 try:  
 with self.ftp\_lock:  
 self.ftp.rename(str(old\_name), str(new\_name))  
 return True, "Успешно переименовано"  
 except Exception as e:  
 return False, str(e)  
  
 def change\_directory(self, path: str) -> Tuple[bool, str]:  
 if not self.ftp:  
 return False, "Нет подключения"  
  
 try:  
 with self.ftp\_lock:  
 self.ftp.cwd(path)  
 self.current\_remote\_dir = self.ftp.pwd()  
 return True, "Директория изменена"  
 except Exception as e:  
 return False, str(e)  
  
 def get\_current\_directory(self) -> str:  
 if not self.ftp:  
 return "/"  
 try:  
 with self.ftp\_lock:  
 return self.ftp.pwd()  
 except:  
 return "/"  
  
 def \_parse\_ftp\_time(self, time\_str: str) -> str:  
 *"""Парсинг времени из FTP-листинга с учетом UTC"""* try:  
 current\_year = datetime.now().year  
 months = {  
 'Jan': 1, 'Feb': 2, 'Mar': 3, 'Apr': 4, 'May': 5, 'Jun': 6,  
 'Jul': 7, 'Aug': 8, 'Sep': 9, 'Oct': 10, 'Nov': 11, 'Dec': 12  
 }  
   
 parts = time\_str.split()  
 if len(parts) != 3:  
 return time\_str  
   
 month = months.get(parts[0], 1)  
 day = int(parts[1])  
   
 if ':' in parts[2]: # Формат времени ЧЧ:ММ  
 hour, minute = map(int, parts[2].split(':'))  
 year = current\_year  
  
 dt = datetime(year, month, day, hour, minute, tzinfo=timezone.utc)  
  
 if dt > datetime.now(timezone.utc):  
 dt = datetime(year - 1, month, day, hour, minute, tzinfo=timezone.utc)  
  
 local\_dt = dt.astimezone()  
 return local\_dt.strftime("%Y-%m-%d %H:%M")  
 else:  
 year = int(parts[2])  
 dt = datetime(year, month, day, 0, 0, tzinfo=timezone.utc)  
 local\_dt = dt.astimezone()  
 return local\_dt.strftime("%Y-%m-%d %H:%M")  
 except:  
 return time\_str  
  
 def \_get\_optimal\_buffer\_size(self, file\_size: int) -> int:  
 if file\_size < 1024 \* 1024: # < 1MB  
 return 8192  
 elif file\_size < 10 \* 1024 \* 1024: # < 10MB  
 return 32768  
 else: # >= 10MB  
 return 65536  
  
 def start\_connection\_monitor(self, on\_connection\_lost: Callable):  
 def monitor():  
 while not self.stop\_monitor:  
 try:  
 with self.ftp\_lock:  
 if self.ftp:  
 self.ftp.voidcmd("NOOP")  
 except:  
 if self.settings.get('auto\_reconnect', True):  
 attempts = self.settings.get('reconnect\_attempts', 3)  
 for \_ in range(attempts):  
 success, \_ = self.reconnect()  
 if success:  
 break  
 else:  
 on\_connection\_lost()  
 else:  
 on\_connection\_lost()  
 time.sleep(30)  
  
 self.stop\_monitor = False  
 self.monitor\_thread = Thread(target=monitor, daemon=True)  
 self.monitor\_thread.start()  
  
 def stop\_connection\_monitor(self):  
 self.stop\_monitor = True  
 if self.monitor\_thread:  
 self.monitor\_thread.join(timeout=1)  
 self.monitor\_thread = None  
  
 def copy\_file(self, source: str, destination: str) -> Tuple[bool, str]:  
 *"""Копирование файла на FTP сервере"""* if not self.ftp:  
 return False, "Нет подключения"  
  
 debug\_log(f"\nDEBUG: Начинаем копирование файла {source} -> {destination}")  
   
 try:  
 with self.ftp\_lock:  
 # Проверяем, является ли источник директорией  
 try:  
 current\_dir = self.ftp.pwd()  
 self.ftp.cwd(source)  
 self.ftp.cwd(current\_dir)  
 # Если мы здесь, значит source - это директория  
 debug\_log("DEBUG: Источник является директорией")  
 return self.copy\_directory(source, destination)  
 except:  
 # Если не удалось перейти, значит это файл  
 debug\_log("DEBUG: Источник является файлом")  
 pass  
  
 # Создаем временный файл для копирования  
 import tempfile  
 with tempfile.NamedTemporaryFile(delete=False) as temp\_file:  
 temp\_path = temp\_file.name  
 debug\_log(f"DEBUG: Создан временный файл: {temp\_path}")  
   
 try:  
 # Скачиваем исходный файл  
 debug\_log(f"DEBUG: Скачиваем файл {source}")  
 self.ftp.retrbinary(f'RETR {source}', temp\_file.write)  
 temp\_file.close() # Закрываем файл после записи  
   
 # Загружаем файл с новым именем  
 debug\_log(f"DEBUG: Загружаем файл как {destination}")  
 with open(temp\_path, 'rb') as f:  
 self.ftp.storbinary(f'STOR {destination}', f)  
   
 debug\_log("DEBUG: Копирование успешно завершено")  
 return True, "Файл успешно скопирован"  
 finally:  
 try:  
 os.unlink(temp\_path)  
 debug\_log("DEBUG: Временный файл удален")  
 except:  
 pass  
  
 except Exception as e:  
 error\_msg = str(e)  
 debug\_log(f"DEBUG: Ошибка копирования: {error\_msg}")  
 return False, f"Ошибка копирования файла: {error\_msg}"  
  
 def copy\_directory(self, source: str, destination: str) -> Tuple[bool, str]:  
 *"""Копирование директории на FTP сервере"""* if not self.ftp:  
 return False, "Нет подключения"  
  
 debug\_log(f"\nDEBUG: Начинаем копирование директории {source} -> {destination}")  
   
 try:  
 with self.ftp\_lock:  
 current\_dir = self.ftp.pwd()  
   
 # Создаем новую директорию  
 try:  
 debug\_log(f"DEBUG: Создаем директорию {destination}")  
 self.ftp.mkd(destination)  
 except:  
 debug\_log("DEBUG: Директория назначения уже существует или ошибка создания")  
 pass  
  
 try:  
 # Переходим в исходную директорию  
 debug\_log(f"DEBUG: Переходим в исходную директорию {source}")  
 self.ftp.cwd(source)  
 source\_dir = self.ftp.pwd()  
   
 # Получаем список файлов в исходной директории  
 items = []  
 def list\_callback(line):  
 items.append(line)  
 self.ftp.retrlines('LIST', list\_callback)  
 debug\_log(f"DEBUG: Получен список файлов: {len(items)} элементов")  
   
 # Возвращаемся в исходную директорию и переходим в целевую  
 self.ftp.cwd(current\_dir)  
 self.ftp.cwd(destination)  
 dest\_dir = self.ftp.pwd()  
   
 # Копируем каждый элемент  
 for item in items:  
 parts = item.split(maxsplit=8)  
 if len(parts) < 9:  
 continue  
   
 name = parts[8]  
 if name in ('.', '..'):  
 continue  
   
 is\_dir = item.startswith('d')  
   
 if is\_dir:  
 debug\_log(f"DEBUG: Копируем поддиректорию {name}")  
 # Создаем поддиректорию  
 try:  
 self.ftp.mkd(name)  
 except:  
 pass  
   
 # Рекурсивно копируем содержимое  
 self.ftp.cwd(source\_dir) # Возвращаемся в исходную директорию  
 success, message = self.copy\_directory(f"{source}/{name}", f"{destination}/{name}")  
 if not success:  
 debug\_log(f"DEBUG: Ошибка копирования поддиректории: {message}")  
 return False, f"Ошибка копирования поддиректории {name}: {message}"  
 self.ftp.cwd(dest\_dir) # Возвращаемся в целевую директорию  
 else:  
 debug\_log(f"DEBUG: Копируем файл {name}")  
 # Создаем временный файл для копирования  
 import tempfile  
 with tempfile.NamedTemporaryFile(delete=False) as temp\_file:  
 temp\_path = temp\_file.name  
 debug\_log(f"DEBUG: Создан временный файл: {temp\_path}")  
   
 try:  
 # Скачиваем файл из исходной директории  
 self.ftp.cwd(source\_dir)  
 self.ftp.retrbinary(f'RETR {name}', temp\_file.write)  
 temp\_file.close()  
   
 # Загружаем файл в целевую директорию  
 self.ftp.cwd(dest\_dir)  
 with open(temp\_path, 'rb') as f:  
 self.ftp.storbinary(f'STOR {name}', f)  
   
 debug\_log(f"DEBUG: Файл {name} успешно скопирован")  
 finally:  
 try:  
 os.unlink(temp\_path)  
 debug\_log("DEBUG: Временный файл удален")  
 except:  
 pass  
   
 debug\_log("DEBUG: Копирование директории успешно завершено")  
 return True, "Директория успешно скопирована"  
   
 finally:  
 # Возвращаемся в исходную директорию  
 try:  
 self.ftp.cwd(current\_dir)  
 except:  
 pass  
  
 except Exception as e:  
 error\_msg = str(e)  
 debug\_log(f"DEBUG: Ошибка копирования директории: {error\_msg}")  
 return False, f"Ошибка копирования директории: {error\_msg}" import os  
import json  
from typing import Dict, Any  
  
  
class Settings:  
 def \_\_init\_\_(self):  
 self.settings\_file = os.path.join(  
 os.path.expanduser("~"), ".ftp\_client\_settings.json")  
 self.default\_settings = {  
 'default\_local\_dir': os.path.expanduser("~"),  
 'show\_hidden\_files': False,  
 'sort\_folders\_first': True,  
 'confirm\_delete': True,  
 'confirm\_overwrite': True,  
 'buffer\_size': 8192,  
 'encoding': 'utf-8',  
 'date\_format': '%Y-%m-%d %H:%M:%S',  
 'theme': 'default',  
 'auto\_reconnect': True,  
 'reconnect\_attempts': 3,  
 'cache\_ttl': 30  
 }  
 self.current\_settings = self.load\_settings()  
  
 def load\_settings(self) -> Dict[str, Any]:  
 try:  
 if os.path.exists(self.settings\_file):  
 with open(self.settings\_file, 'r') as f:  
 saved\_settings = json.load(f)  
 settings = self.default\_settings.copy()  
 settings.update(saved\_settings)  
 return settings  
 except Exception as e:  
 print(f"Ошибка загрузки настроек: {e}")  
 return self.default\_settings.copy()  
  
 def save\_settings(self):  
 try:  
 with open(self.settings\_file, 'w') as f:  
 json.dump(self.current\_settings, f, indent=4)  
 except Exception as e:  
 print(f"Ошибка сохранения настроек: {e}")  
  
 def get(self, key: str, default: Any = None) -> Any:  
 return self.current\_settings.get(key, default)  
  
 def set(self, key: str, value: Any):  
 self.current\_settings[key] = value  
  
 def update(self, settings: Dict[str, Any]):  
 self.current\_settings.update(settings)  
  
 def reset(self):  
 self.current\_settings = self.default\_settings.copy()  
 self.save\_settings()

import socket  
import threading  
import time  
from typing import Dict, Optional  
  
class ConnectionMonitor:  
 def \_\_init\_\_(self, host: str, port: int):  
 self.host = host  
 self.port = port  
 self.running = False  
 self.stats: Dict[str, float] = {  
 'latency': 0.0,  
 'packet\_loss': 0.0,  
 'last\_check': 0.0  
 }  
 self.monitor\_thread: Optional[threading.Thread] = None  
 self.lock = threading.Lock()  
  
 def start\_monitoring(self):  
 self.running = True  
 self.monitor\_thread = threading.Thread(target=self.\_monitor\_loop)  
 self.monitor\_thread.daemon = True  
 self.monitor\_thread.start()  
  
 def stop\_monitoring(self):  
 self.running = False  
 if self.monitor\_thread:  
 self.monitor\_thread.join()  
  
 def \_monitor\_loop(self):  
 while self.running:  
 try:  
 start\_time = time.time()  
 with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as sock:  
 sock.settimeout(2.0)  
 result = sock.connect\_ex((self.host, self.port))  
 end\_time = time.time()  
   
 with self.lock:  
 if result == 0:  
 self.stats['latency'] = (end\_time - start\_time) \* 1000  
 self.stats['packet\_loss'] = 0.0  
 else:  
 self.stats['packet\_loss'] = 100.0  
 self.stats['last\_check'] = time.time()  
   
 except Exception:  
 with self.lock:  
 self.stats['packet\_loss'] = 100.0  
 self.stats['latency'] = 0.0  
 self.stats['last\_check'] = time.time()  
   
 time.sleep(1)  
  
 def get\_stats(self) -> Dict[str, float]:  
 with self.lock:  
 return self.stats.copy()

from cryptography.fernet import Fernet  
import base64  
import os  
from typing import Optional  
  
  
class Crypto:  
 def \_\_init\_\_(self):  
 self.key\_file = os.path.join(  
 os.path.expanduser("~"), ".ftp\_client\_key")  
 self.key = self.\_load\_or\_generate\_key()  
 self.fernet = Fernet(self.key)  
  
 def \_load\_or\_generate\_key(self) -> bytes:  
 try:  
 if os.path.exists(self.key\_file):  
 with open(self.key\_file, 'rb') as f:  
 return f.read()  
 except Exception:  
 pass  
  
 key = Fernet.generate\_key()  
 try:  
 with open(self.key\_file, 'wb') as f:  
 f.write(key)  
 except Exception as e:  
 print(f"Ошибка сохранения ключа: {e}")  
 return key  
  
 def encrypt(self, data: str) -> str:  
 if not data:  
 return ""  
 try:  
 return self.fernet.encrypt(data.encode()).decode()  
 except Exception as e:  
 print(f"Ошибка шифрования: {e}")  
 return ""  
  
 def decrypt(self, encrypted\_data: str) -> str:  
 if not encrypted\_data:  
 return ""  
 try:  
 return self.fernet.decrypt(encrypted\_data.encode()).decode()  
 except Exception as e:  
 print(f"Ошибка дешифрования: {e}")  
 return ""

import os  
from datetime import datetime  
from typing import List, Dict, Any  
import humanize  
  
  
def list\_directory(path: str) -> List[Dict[str, Any]]:  
 items = []  
 try:  
 for name in os.listdir(path):  
 full\_path = os.path.join(path, name)  
 stat = os.stat(full\_path)  
   
 item = {  
 'name': name,  
 'path': full\_path,  
 'size': stat.st\_size,  
 'type': 'folder' if os.path.isdir(full\_path) else 'file',  
 'modified': datetime.fromtimestamp(stat.st\_mtime),  
 'permissions': stat.st\_mode  
 }  
  
 if item['type'] == 'file':  
 item['size\_human'] = humanize.naturalsize(item['size'])  
 else:  
 item['size\_human'] = ''  
   
 items.append(item)  
 except Exception as e:  
 print(f"Ошибка чтения директории {path}: {e}")  
   
 return items  
  
  
def filter\_hidden\_files(items: List[Dict[str, Any]], show\_hidden: bool = False) -> List[Dict[str, Any]]:  
 if show\_hidden:  
 return items  
   
 return [item for item in items if not item['name'].startswith('.')]  
  
  
def sort\_items(items: List[Dict[str, Any]], folders\_first: bool = True) -> List[Dict[str, Any]]:  
 if folders\_first:  
 folders = sorted(  
 [item for item in items if item['type'] == "Папка"],  
 key=lambda x: x['name'].lower()  
 )  
 files = sorted(  
 [item for item in items if item['type'] == "Файл"],  
 key=lambda x: x['name'].lower()  
 )  
 return folders + files  
 else:  
 return sorted(items, key=lambda x: x['name'].lower())  
  
  
def format\_size(size: int) -> str:  
 return humanize.naturalsize(size)  
  
  
def format\_date(date: datetime, format\_str: str = '%Y-%m-%d %H:%M:%S') -> str:  
 return date.strftime(format\_str)  
  
  
def get\_file\_type(filename: str) -> str:  
 ext = os.path.splitext(filename)[1].lower()  
   
 types = {  
 '.txt': 'Текстовый файл',  
 '.doc': 'Microsoft Word',  
 '.docx': 'Microsoft Word',  
 '.pdf': 'PDF документ',  
 '.jpg': 'Изображение JPEG',  
 '.jpeg': 'Изображение JPEG',  
 '.png': 'Изображение PNG',  
 '.gif': 'Изображение GIF',  
 '.mp3': 'Аудио MP3',  
 '.wav': 'Аудио WAV',  
 '.mp4': 'Видео MP4',  
 '.avi': 'Видео AVI',  
 '.zip': 'Архив ZIP',  
 '.rar': 'Архив RAR',  
 '.py': 'Python скрипт',  
 '.html': 'HTML файл',  
 '.css': 'CSS файл',  
 '.js': 'JavaScript файл'  
 }  
   
 return types.get(ext, 'Файл')

import tkinter as tk  
from tkinter import ttk  
import time  
from src.utils.connection\_monitor import ConnectionMonitor  
  
class ConnectionStatsPanel(ttk.LabelFrame):  
 def \_\_init\_\_(self, parent):  
 super().\_\_init\_\_(parent, text="Статистика соединения")  
   
 # Создаем метки для отображения статистики  
 self.latency\_label = ttk.Label(self, text="Задержка: --")  
 self.latency\_label.pack(anchor="w", padx=5, pady=2)  
   
 self.packet\_loss\_label = ttk.Label(self, text="Потери пакетов: --")  
 self.packet\_loss\_label.pack(anchor="w", padx=5, pady=2)  
   
 self.last\_check\_label = ttk.Label(self, text="Последняя проверка: --")  
 self.last\_check\_label.pack(anchor="w", padx=5, pady=2)  
   
 self.monitor = None  
   
 def start\_monitoring(self, host: str, port: int):  
 if self.monitor:  
 self.stop\_monitoring()  
   
 self.monitor = ConnectionMonitor(host, port)  
 self.monitor.start\_monitoring()  
 self.update\_stats()  
   
 def stop\_monitoring(self):  
 if self.monitor:  
 self.monitor.stop\_monitoring()  
 self.monitor = None  
  
 self.latency\_label.config(text="Задержка: --")  
 self.packet\_loss\_label.config(text="Потери пакетов: --")  
 self.last\_check\_label.config(text="Последняя проверка: --")  
   
 def update\_stats(self):  
 *"""Обновление статистики соединения"""* if self.monitor:  
 stats = self.monitor.get\_stats()  
   
 # Форматируем задержку  
 latency = stats['latency']  
 if latency > 0:  
 latency\_text = f"Задержка: {latency:.1f} мс"  
 else:  
 latency\_text = "Задержка: --"  
 self.latency\_label.config(text=latency\_text)  
   
 # Форматируем потери пакетов  
 packet\_loss = stats['packet\_loss']  
 if packet\_loss >= 0:  
 packet\_loss\_text = f"Потери пакетов: {packet\_loss:.1f}%"  
 else:  
 packet\_loss\_text = "Потери пакетов: --"  
 self.packet\_loss\_label.config(text=packet\_loss\_text)  
   
 # Форматируем время последней проверки  
 last\_check = stats['last\_check']  
 if last\_check > 0:  
 last\_check\_time = time.strftime("%H:%M:%S", time.localtime(last\_check))  
 last\_check\_text = f"Последняя проверка: {last\_check\_time}"  
 else:  
 last\_check\_text = "Последняя проверка: --"  
 self.last\_check\_label.config(text=last\_check\_text)  
   
 # Обновляем каждые 500 мс  
 self.after(500, self.update\_stats)

import tkinter as tk  
from tkinter import ttk, messagebox  
from typing import Callable, Dict, List, Tuple  
from datetime import datetime  
import sys  
  
  
class QuickConnectDialog:  
 def \_\_init\_\_(self, parent, callback: Callable):  
 self.dialog = tk.Toplevel(parent)  
 self.dialog.title("Быстрое подключение")  
 self.dialog.geometry("350x200")  
 self.dialog.transient(parent)  
 self.dialog.grab\_set()  
  
 self.dialog.geometry("+%d+%d" % (  
 parent.winfo\_rootx() + parent.winfo\_width()//2 - 150,  
 parent.winfo\_rooty() + parent.winfo\_height()//2 - 100  
 ))  
  
 frame = ttk.Frame(self.dialog, padding="10")  
 frame.pack(fill=tk.BOTH, expand=True)  
  
 entries = {}  
 fields = [  
 ("host", "Сервер:", "localhost"),  
 ("port", "Порт:", "21"),  
 ("user", "Пользователь:", "user"),  
 ("password", "Пароль:", "")  
 ]  
  
 for row, (field, label, default) in enumerate(fields):  
 ttk.Label(frame, text=label).grid(row=row, column=0, sticky="e", pady=2)  
 entry = ttk.Entry(frame)  
 entry.insert(0, default)  
 entry.grid(row=row, column=1, sticky="ew", pady=2)  
 if field == "password":  
 entry.configure(show="\*")  
 entries[field] = entry  
  
 btn\_frame = ttk.Frame(frame)  
 btn\_frame.grid(row=len(fields), column=0, columnspan=2, pady=10)  
  
 ttk.Button(btn\_frame, text="Подключиться",   
 command=lambda: self.\_connect(callback, entries)).pack(side=tk.LEFT, padx=5)  
  
 ttk.Button(btn\_frame, text="Отмена",  
 command=self.dialog.destroy).pack(side=tk.LEFT, padx=5)  
  
 def \_connect(self, callback: Callable, entries: Dict[str, ttk.Entry]):  
 try:  
 host = entries["host"].get()  
 port = int(entries["port"].get())  
 user = entries["user"].get()  
 password = entries["password"].get()  
   
 self.dialog.destroy()  
 callback(host, port, user, password)  
 except ValueError:  
 messagebox.showerror("Ошибка", "Неверный формат порта")  
  
  
class HistoryDialog:  
 def \_\_init\_\_(self, parent, history: List[Dict], callback: Callable):  
 self.dialog = tk.Toplevel(parent)  
 self.dialog.title("История подключений")  
 self.dialog.geometry("800x300")  
 self.dialog.transient(parent)  
 self.dialog.grab\_set()  
  
 tree = ttk.Treeview(self.dialog,   
 columns=("host", "port", "user", "date"),  
 show="headings")  
   
 tree.heading("host", text="Сервер")  
 tree.heading("port", text="Порт")  
 tree.heading("user", text="Пользователь")  
 tree.heading("date", text="Дата")  
  
 tree.column("host", width=250)  
 tree.column("port", width=100)  
 tree.column("user", width=200)  
 tree.column("date", width=200)  
  
 for conn in history:  
 date = datetime.fromisoformat(conn['timestamp']).strftime("%Y-%m-%d %H:%M")  
 tree.insert("", tk.END, values=(  
 conn['host'],  
 conn['port'],  
 conn['user'],  
 date  
 ))  
  
 tree.bind("<Double-1>", lambda e: self.\_connect(callback, tree))  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 def \_connect(self, callback: Callable, tree: ttk.Treeview):  
 selected = tree.selection()  
 if not selected:  
 return  
   
 item = tree.item(selected[0])  
 values = item['values']  
   
 self.dialog.destroy()  
 callback(values)  
  
  
class BookmarksDialog:  
 def \_\_init\_\_(self, parent, bookmarks: List[Dict],   
 connect\_callback: Callable, delete\_callback: Callable):  
 self.dialog = tk.Toplevel(parent)  
 self.dialog.title("Закладки")  
 self.dialog.geometry("800x350")  
 self.dialog.transient(parent)  
 self.dialog.grab\_set()  
  
 tree = ttk.Treeview(self.dialog,   
 columns=("name", "host", "port", "user"),  
 show="headings")  
   
 tree.heading("name", text="Название")  
 tree.heading("host", text="Сервер")  
 tree.heading("port", text="Порт")  
 tree.heading("user", text="Пользователь")  
 tree.column("name", width=200)  
 tree.column("host", width=250)  
 tree.column("port", width=100)  
 tree.column("user", width=200)  
  
 for bookmark in bookmarks:  
 tree.insert("", tk.END, values=(  
 bookmark['name'],  
 bookmark['host'],  
 bookmark['port'],  
 bookmark['user']  
 ))  
  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 btn\_frame = ttk.Frame(self.dialog)  
 btn\_frame.pack(fill=tk.X, padx=5, pady=5)  
   
 ttk.Button(btn\_frame, text="Подключиться",   
 command=lambda: self.\_connect(connect\_callback, tree)).pack(side=tk.LEFT, padx=2)  
 ttk.Button(btn\_frame, text="Удалить",   
 command=lambda: self.\_delete(delete\_callback, tree)).pack(side=tk.LEFT, padx=2)  
  
 def \_connect(self, callback: Callable, tree: ttk.Treeview):  
 selected = tree.selection()  
 if not selected:  
 return  
   
 item = tree.item(selected[0])  
 values = item['values']  
   
 self.dialog.destroy()  
 callback(values)  
  
 def \_delete(self, callback: Callable, tree: ttk.Treeview):  
 selected = tree.selection()  
 if not selected:  
 return  
   
 item = tree.item(selected[0])  
 name = item['values'][0]  
   
 if callback(name):  
 tree.delete(selected)  
  
  
class SettingsDialog:  
 def \_\_init\_\_(self, parent, settings: Dict, callback: Callable):  
 self.dialog = tk.Toplevel(parent)  
 self.dialog.title("Настройки")  
 self.dialog.geometry("600x500")  
 self.dialog.transient(parent)  
 self.dialog.grab\_set()  
  
 notebook = ttk.Notebook(self.dialog)  
 notebook.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)  
  
 general\_frame = ttk.Frame(notebook)  
 notebook.add(general\_frame, text="Общие")  
  
 interface\_frame = ttk.LabelFrame(general\_frame, text="Интерфейс")  
 interface\_frame.pack(fill=tk.X, padx=5, pady=5)  
  
 self.show\_hidden = tk.BooleanVar(value=settings.get('show\_hidden\_files', False))  
 self.sort\_folders = tk.BooleanVar(value=settings.get('sort\_folders\_first', True))  
 self.confirm\_delete = tk.BooleanVar(value=settings.get('confirm\_delete', True))  
 self.confirm\_overwrite = tk.BooleanVar(value=settings.get('confirm\_overwrite', True))  
  
 ttk.Checkbutton(interface\_frame, text="Показывать скрытые файлы",  
 variable=self.show\_hidden).pack(anchor="w", padx=5, pady=2)  
 ttk.Checkbutton(interface\_frame, text="Показывать папки первыми",  
 variable=self.sort\_folders).pack(anchor="w", padx=5, pady=2)  
 ttk.Checkbutton(interface\_frame, text="Подтверждать удаление",  
 variable=self.confirm\_delete).pack(anchor="w", padx=5, pady=2)  
 ttk.Checkbutton(interface\_frame, text="Подтверждать перезапись",  
 variable=self.confirm\_overwrite).pack(anchor="w", padx=5, pady=2)  
  
 btn\_frame = ttk.Frame(self.dialog)  
 btn\_frame.pack(fill=tk.X, padx=5, pady=5)  
  
 ttk.Button(btn\_frame, text="Сохранить",   
 command=lambda: self.\_save(callback)).pack(side=tk.RIGHT, padx=5)  
 ttk.Button(btn\_frame, text="Отмена",  
 command=self.dialog.destroy).pack(side=tk.RIGHT, padx=5)  
  
 def \_save(self, callback: Callable):  
 settings = {  
 'show\_hidden\_files': self.show\_hidden.get(),  
 'sort\_folders\_first': self.sort\_folders.get(),  
 'confirm\_delete': self.confirm\_delete.get(),  
 'confirm\_overwrite': self.confirm\_overwrite.get()  
 }  
 self.dialog.destroy()  
 callback(settings)  
  
  
class AboutDialog:  
 def \_\_init\_\_(self, parent):  
 self.dialog = tk.Toplevel(parent)  
 self.dialog.title("О программе")  
 self.dialog.geometry("600x500")  
 self.dialog.transient(parent)  
 self.dialog.grab\_set()  
  
 # Центрируем окно  
 self.dialog.geometry("+%d+%d" % (  
 parent.winfo\_rootx() + parent.winfo\_width()//2 - 300,  
 parent.winfo\_rooty() + parent.winfo\_height()//2 - 250  
 ))  
  
 main\_frame = ttk.Frame(self.dialog, padding="20")  
 main\_frame.pack(fill=tk.BOTH, expand=True)  
  
 # Заголовок  
 title\_label = ttk.Label(main\_frame,   
 text="FTP Клиент",  
 font=('Helvetica', 16, 'bold'))  
 title\_label.pack(pady=(0, 10))  
  
 # Версия  
 version\_label = ttk.Label(main\_frame,  
 text="Версия 1.0.0")  
 version\_label.pack(pady=(0, 20))  
  
 dev\_label = ttk.Label(main\_frame,  
 text="Ефросинья Сологуб, 2025 год")  
 dev\_label.pack(pady=(0, 20))  
  
 # Создаем notebook для вкладок  
 notebook = ttk.Notebook(main\_frame)  
 notebook.pack(fill=tk.BOTH, expand=True)  
  
 # Вкладка с описанием  
 desc\_frame = ttk.Frame(notebook, padding="10")  
 notebook.add(desc\_frame, text="Описание")  
  
 description = """  
 FTP клиент с графическим интерфейсом, поддерживающий:  
 • Передачу файлов между локальной и удаленной системами  
 • Работу с папками и вложенными структурами  
 • Закладки и историю подключений  
 • Поиск по файлам  
 • Мониторинг состояния соединения  
 """  
 desc\_label = ttk.Label(desc\_frame, text=description, justify=tk.LEFT)  
 desc\_label.pack(anchor=tk.W)  
  
 keys\_frame = ttk.Frame(notebook, padding="10")  
 notebook.add(keys\_frame, text="Горячие клавиши")  
  
 mod = "⌘" if sys.platform == 'darwin' else "Ctrl"  
  
 hotkeys = f"""  
 Основные команды:  
 • {mod}+H - История подключений  
 • {mod}+B - Добавить в закладки  
 • {mod}+N - Создать папку  
 • {mod}+U - Загрузить файлы  
 • {mod}+D - Скачать файлы  
 • {mod}+R или F5 - Обновить списки  
 • {mod}+, - Настройки  
 • {mod}+Q - Выход  
   
 Навигация:  
 • Delete - Удаление выбранного файла  
 • Escape - Выход из полноэкранного режима  
   
 Enter - Подтверждение действия  
 """  
 keys\_label = ttk.Label(keys\_frame, text=hotkeys, justify=tk.LEFT)  
 keys\_label.pack(anchor=tk.W)  
  
 ttk.Button(main\_frame,   
 text="Закрыть",  
 command=self.dialog.destroy).pack(pady=(20, 0))

from tkinter import ttk  
  
  
def setup\_styles():  
 style = ttk.Style()  
  
 style.theme\_use('clam')  
  
 style.configure('TButton',  
 padding=5,  
 relief='flat',  
 background='#f0f0f0',  
 foreground='black')  
   
 style.map('TButton',  
 background=[('active', '#e0e0e0'),  
 ('pressed', '#d0d0d0')],  
 relief=[('pressed', 'sunken')])  
  
 style.configure('TLabel',  
 padding=2,  
 background='#f5f5f5',  
 foreground='black')  
  
 style.configure('TEntry',  
 padding=5,  
 relief='solid',  
 fieldbackground='white')  
  
 style.configure('TFrame',  
 background='#f5f5f5')  
   
 style.configure('TLabelframe',  
 background='#f5f5f5',  
 padding=5)  
   
 style.configure('TLabelframe.Label',  
 background='#f5f5f5',  
 foreground='black',  
 padding=(5, 2))  
  
 style.configure('Treeview',  
 background='white',  
 fieldbackground='white',  
 relief='solid',  
 borderwidth=1,  
 rowheight=25)  
   
 style.configure('Treeview.Heading',  
 padding=2,  
 relief='flat',  
 background='#e0e0e0',  
 foreground='black')  
   
 style.map('Treeview',  
 background=[('selected', '#0078d7')],  
 foreground=[('selected', 'white')])  
  
 style.configure('Vertical.TScrollbar',  
 background='#f0f0f0',  
 relief='flat',  
 arrowcolor='black',  
 borderwidth=0)  
   
 style.configure('Horizontal.TScrollbar',  
 background='#f0f0f0',  
 relief='flat',  
 arrowcolor='black',  
 borderwidth=0)  
  
 style.configure('Toolbar.TFrame',  
 background='#f0f0f0',  
 relief='flat')  
  
 style.configure('Statusbar.TFrame',  
 background='#f0f0f0',  
 relief='sunken',  
 borderwidth=1)  
   
 style.configure('Statusbar.TLabel',  
 background='#f0f0f0',  
 foreground='black',  
 font=('Helvetica', 9),  
 padding=(5, 2))  
   
 style.configure('StatusbarError.TLabel',  
 background='#f0f0f0',  
 foreground='red',  
 font=('Helvetica', 9, 'bold'),  
 padding=(5, 2))  
  
 style.configure("Colored.Horizontal.TProgressbar",  
 troughcolor='#f0f0f0',  
 background='#4CAF50',  
 bordercolor='#f0f0f0',  
 lightcolor='#4CAF50',  
 darkcolor='#4CAF50')  
  
 style.configure('Search.TFrame',  
 background='#f5f5f5',  
 relief='flat',  
 padding=5)  
  
 style.configure('Path.TFrame',  
 background='#f5f5f5',  
 relief='flat',  
 padding=2)  
  
 style.configure('Connection.TFrame',  
 background='#f5f5f5',  
 relief='flat',  
 padding=5)  
  
 style.configure('Dialog.TFrame',  
 background='#f5f5f5',  
 relief='flat',  
 padding=10)

import tkinter as tk  
from tkinter import ttk, filedialog, messagebox  
from typing import Callable, Optional, List, Tuple, Dict, Any  
from datetime import datetime  
  
  
class FileListView(ttk.Treeview):  
 def \_\_init\_\_(self, parent, \*\*kwargs):  
 super().\_\_init\_\_(parent,   
 columns=("name", "size", "type", "modified"),  
 show="headings",  
 selectmode="extended",  
 \*\*kwargs)  
  
 self.heading("name", text="Имя")  
 self.heading("size", text="Размер")  
 self.heading("type", text="Тип")  
 self.heading("modified", text="Изменён")  
   
 self.column("name", width=300)  
 self.column("size", width=100)  
 self.column("type", width=100)  
 self.column("modified", width=150)  
  
 self.scrollbar = ttk.Scrollbar(parent, orient="vertical", command=self.yview)  
 self.configure(yscrollcommand=self.scrollbar.set)  
  
 self.pack(side="left", fill="both", expand=True)  
 self.scrollbar.pack(side="right", fill="y")  
  
 self.current\_sort = None  
 self.reverse\_sort = False  
  
 def set\_items(self, items: List[Any]) -> None:  
 self.delete(\*self.get\_children())  
 for item in items:  
 if isinstance(item, dict):  
 values = (  
 item['name'],  
 item.get('size\_human', ''),  
 item['type'],  
 item['modified'].strftime('%Y-%m-%d %H:%M:%S') if isinstance(item['modified'], datetime) else item['modified']  
 )  
 else:  
 values = item  
 self.insert("", "end", values=values)  
  
 def \_sort\_by\_column(self, column: str) -> None:  
 items = [(self.set(item, column), item) for item in self.get\_children('')]  
  
 if self.current\_sort == column:  
 self.reverse\_sort = not self.reverse\_sort  
 else:  
 self.reverse\_sort = False  
 self.current\_sort = column  
  
 items.sort(reverse=self.reverse\_sort)  
 for index, (\_, item) in enumerate(items):  
 self.move(item, '', index)  
  
  
class StatusBar(ttk.Frame):  
 def \_\_init\_\_(self, parent, \*\*kwargs):  
 super().\_\_init\_\_(parent, style='Statusbar.TFrame', \*\*kwargs)  
  
 status\_frame = ttk.Frame(self, style='Statusbar.TFrame')  
 status\_frame.pack(side="left", fill="x", expand=True, padx=5, pady=2)  
   
 self.status\_label = ttk.Label(status\_frame,  
 text="Готов",  
 style='Statusbar.TLabel')  
 self.status\_label.pack(side="left", fill="x", expand=True)  
  
 progress\_frame = ttk.Frame(self, style='Statusbar.TFrame')  
 progress\_frame.pack(side="right", padx=5, pady=2)  
   
 self.progress = ttk.Progressbar(progress\_frame,  
 orient="horizontal",  
 length=200,  
 mode='determinate',  
 style='Colored.Horizontal.TProgressbar')  
 self.progress.pack(side="right")  
  
 self.percent\_label = ttk.Label(progress\_frame,  
 text="",  
 style='Statusbar.TLabel')  
 self.percent\_label.pack(side="right", padx=(0, 5))  
  
 def set\_status(self, text: str, error: bool = False) -> None:  
 self.status\_label.configure(  
 text=text,  
 style='StatusbarError.TLabel' if error else 'Statusbar.TLabel'  
 )  
 self.status\_label.update\_idletasks()  
  
 def show\_progress(self, show: bool = True) -> None:  
 if show:  
 self.progress.pack(side="right")  
 self.percent\_label.pack(side="right", padx=(0, 5))  
 else:  
 self.progress.pack\_forget()  
 self.percent\_label.pack\_forget()  
  
 def set\_progress(self, value: float) -> None:  
 self.progress['value'] = value  
 self.percent\_label.configure(text=f"{int(value)}%")  
 self.progress.update\_idletasks()  
 self.percent\_label.update\_idletasks()  
  
  
class ConnectionPanel(ttk.LabelFrame):  
 def \_\_init\_\_(self, parent, on\_connect: Callable, \*\*kwargs):  
 super().\_\_init\_\_(parent, text="Подключение", style='Connection.TFrame', \*\*kwargs)  
  
 entries = [  
 ("Сервер:", "host", "localhost"),  
 ("Порт:", "port", "21"),  
 ("Пользователь:", "user", "user")  
 ]  
   
 self.entries = {}  
 for i, (label, name, default) in enumerate(entries):  
 ttk.Label(self, text=label).grid(row=i, column=0, padx=5, pady=2, sticky="e")  
 entry = ttk.Entry(self)  
 entry.insert(0, default)  
 entry.grid(row=i, column=1, padx=5, pady=2, sticky="ew")  
 self.entries[name] = entry  
  
 ttk.Label(self, text="Пароль:").grid(row=3, column=0, padx=5, pady=2, sticky="e")  
 pwd\_frame = ttk.Frame(self)  
 pwd\_frame.grid(row=3, column=1, padx=5, pady=2, sticky="ew")  
   
 self.password\_entry = ttk.Entry(pwd\_frame, show="\*")  
 self.password\_entry.pack(side=tk.LEFT, fill=tk.X, expand=True)  
   
 self.show\_password = tk.BooleanVar(value=False)  
 self.toggle\_pwd\_btn = ttk.Button(pwd\_frame, text="👁", width=3,  
 command=self.\_toggle\_password\_visibility)  
 self.toggle\_pwd\_btn.pack(side=tk.LEFT, padx=(2, 0))  
  
 self.connect\_btn = ttk.Button(self, text="Подключиться",  
 command=self.\_on\_button\_click,  
 style="Primary.TButton")  
 self.connect\_btn.grid(row=4, column=0, columnspan=2, pady=5)  
  
 self.columnconfigure(1, weight=1)  
   
 self.\_on\_connect\_callback = on\_connect  
 self.\_is\_connected = False  
  
 def \_on\_button\_click(self) -> None:  
 if self.\_is\_connected:  
 # Если мы подключены, просто вызываем отключение  
 if self.\_on\_connect\_callback:  
 self.\_on\_connect\_callback(None, None, None, None)  
 return  
 else:  
 # Получаем значения полей  
 host = self.entries["host"].get().strip()  
 port = self.entries["port"].get().strip()  
 user = self.entries["user"].get().strip()  
 password = self.password\_entry.get()  
  
 # Базовая валидация  
 if not host:  
 messagebox.showerror("Ошибка", "Поле 'Сервер' не может быть пустым")  
 self.entries["host"].focus()  
 return  
   
 if not port:  
 messagebox.showerror("Ошибка", "Поле 'Порт' не может быть пустым")  
 self.entries["port"].focus()  
 return  
   
 if not user:  
 messagebox.showerror("Ошибка", "Поле 'Пользователь' не может быть пустым")  
 self.entries["user"].focus()  
 return  
   
 if not password:  
 messagebox.showerror("Ошибка", "Поле 'Пароль' не может быть пустым")  
 self.password\_entry.focus()  
 return  
  
 try:  
 port = int(port)  
 except ValueError:  
 messagebox.showerror("Ошибка", "Порт должен быть числом")  
 self.entries["port"].focus()  
 return  
  
 if self.\_on\_connect\_callback:  
 self.\_on\_connect\_callback(host, port, user, password)  
  
 def \_toggle\_password\_visibility(self) -> None:  
 if self.show\_password.get():  
 self.password\_entry.configure(show="\*")  
 self.show\_password.set(False)  
 else:  
 self.password\_entry.configure(show="")  
 self.show\_password.set(True)  
  
 def set\_connected\_state(self, connected: bool) -> None:  
 self.\_is\_connected = connected  
 if connected:  
 self.connect\_btn.configure(text="Отключиться")  
 for entry in self.entries.values():  
 entry.configure(state="disabled")  
 self.password\_entry.configure(state="disabled")  
 else:  
 self.connect\_btn.configure(text="Подключиться")  
 for entry in self.entries.values():  
 entry.configure(state="normal")  
 self.password\_entry.configure(state="normal")  
  
  
class SearchPanel(ttk.LabelFrame):  
 def \_\_init\_\_(self, parent, on\_search: Callable, \*\*kwargs):  
 super().\_\_init\_\_(parent, text="Поиск файлов", style='Search.TFrame', \*\*kwargs)  
  
 input\_frame = ttk.Frame(self)  
 input\_frame.pack(fill=tk.X, padx=5, pady=2)  
  
 self.search\_var = tk.StringVar()  
 self.search\_var.trace('w', lambda \*args: self.\_on\_search())  
   
 self.search\_entry = ttk.Entry(input\_frame, textvariable=self.search\_var)  
 self.search\_entry.pack(side=tk.LEFT, fill=tk.X, expand=True)  
  
 ttk.Button(input\_frame, text="✕", width=3,  
 command=self.clear\_search).pack(side=tk.LEFT, padx=2)  
  
 options\_frame = ttk.Frame(self)  
 options\_frame.pack(fill=tk.X, padx=5, pady=2)  
  
 self.search\_scope = tk.StringVar(value="both")  
 ttk.Radiobutton(options\_frame, text="Локальные",   
 variable=self.search\_scope,   
 value="local",  
 command=self.\_on\_search).pack(side=tk.LEFT, padx=5)  
 ttk.Radiobutton(options\_frame, text="Удаленные",   
 variable=self.search\_scope,   
 value="remote",  
 command=self.\_on\_search).pack(side=tk.LEFT, padx=5)  
 ttk.Radiobutton(options\_frame, text="Везде",   
 variable=self.search\_scope,   
 value="both",  
 command=self.\_on\_search).pack(side=tk.LEFT, padx=5)  
  
 self.case\_sensitive = tk.BooleanVar(value=False)  
 ttk.Checkbutton(options\_frame, text="Учитывать регистр",  
 variable=self.case\_sensitive,  
 command=self.\_on\_search).pack(side=tk.LEFT, padx=5)  
  
 self.search\_in\_folders = tk.BooleanVar(value=True)  
 ttk.Checkbutton(options\_frame, text="Искать в папках",  
 variable=self.search\_in\_folders,  
 command=self.\_on\_search).pack(side=tk.LEFT, padx=5)  
  
 self.\_on\_search\_callback = on\_search  
  
 def clear\_search(self) -> None:  
 self.search\_var.set("")  
  
 def \_on\_search(self) -> None:  
 if self.\_on\_search\_callback:  
 self.\_on\_search\_callback(  
 self.search\_var.get(),  
 self.search\_scope.get(),  
 self.case\_sensitive.get(),  
 self.search\_in\_folders.get()  
 )  
  
  
class PathPanel(ttk.Frame):  
 def \_\_init\_\_(self, parent, on\_path\_change: Optional[Callable] = None, \*\*kwargs):  
 super().\_\_init\_\_(parent, style='Path.TFrame', \*\*kwargs)  
   
 self.path\_var = tk.StringVar()  
 self.label = ttk.Label(self, textvariable=self.path\_var)  
 self.label.pack(side=tk.LEFT, fill=tk.X, expand=True)  
   
 if on\_path\_change:  
 self.browse\_btn = ttk.Button(self, text="...",  
 command=lambda: self.\_browse\_directory(on\_path\_change))  
 self.browse\_btn.pack(side=tk.RIGHT, padx=2)  
  
 def set\_path(self, path: str) -> None:  
 self.path\_var.set(path)  
  
 def \_browse\_directory(self, callback: Callable) -> None:  
 directory = filedialog.askdirectory(initialdir=self.path\_var.get())  
 if directory:  
 callback(directory)