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

from PyQt6 import QtWidgets, uic
from PyQt6.QtWidgets import QWidget, QDialog, QApplication, QLabel, QVBoxLayout, QPushButton, QCheckBox, QGroupBox, \
QFileDialog, QLineEdit, QGridLayout, QScrollArea, QHBoxLayout, QSpacerItem, QSizePolicy, QMenu
from PyQt6.QtGui import QIcon, QDragEnterEvent, QDropEvent, QMoveEvent, QResizeEvent, QContextMenuEvent
from PyQt6.QtCore import QSize, Qt

import os, time

from modules.config import *
from modules.limits import Limits

from modules import helper, protocol, file_send, dialogs, file_viewer

class MainWindow(QtWidgets.QMainWindow):
    """Main application window handling UI, user interactions, and event management."""
    def __init__(self, app, network):
        super().__init__()
        self.app = app
        self.network = network
        self.protocol = protocol.Protocol(self.network, self)
        self.file_sending = file_send.FileSending(self)

        self.window_geometry = WINDOW_GEOMERTY
        self.save_sizes()
        self.setGeometry(self.window_geometry)

        # Set initial size and position
        self.original_width, self.original_height = self.width(), self.height()
        s_width, s_height = app.primaryScreen().geometry().width(), app.primaryScreen().geometry().height()
        self.resize(s_width * 3 // 4, s_height * 2 // 3)
        self.move(s_width // 8, s_height // 6)

        # Enable fast rendering attributes
        self.setAttribute(Qt.WidgetAttribute.WA_OpaquePaintEvent)
        self.setAttribute(Qt.WidgetAttribute.WA_PaintOnScreen, True)

        # Initialize UI state variables
        self.scroll_progress = 0
        self.current_files_amount = ITEMS_TO_LOAD
        self.last_load = time.time()
        self.scroll_size = SCROLL_SIZE

        self.user = {"email": "guest", "username": "guest", "subscription_level": 0, "cwd": "", "parent_cwd": "",
"cwd_name": "", "admin_level": 0}
        self.json = helper.JsonHandle()

        self.search_filter = None
        self.share, self.deleted = False, False
        self.sort, self.sort_direction = "Name", True
        self.remember = False

        self.files, self.directories = [], []
        self.files_downloading = {}
        self.currently_selected = []
        self.uploading_file_id = ""
        self.used_storage = 0
        self.items_amount = 0

        self.original_sizes = {}
        self.scroll = None

        self.start()

    def start(self):
        """Applies initial styling and sets the application icon."""
        try:
            with open(f"{os.getcwd()}/gui/css/style.css", 'r') as f:
                self.app.setStyleSheet(f.read())
        except:
            print(traceback.format_exc())

        if os.path.isfile(f"{os.getcwd()}/assets/icon.ico"):
            self.setWindowIcon(QIcon(f"{os.getcwd()}/assets/icon.ico"))

    def keyPressEvent(self, event):
        """Handles keypress events for shortcuts and file operations."""
        if event.key() == Qt.Key.Key_Delete and self.currently_selected:
            self.protocol.delete()
        elif event.key() == Qt.Key.Key_R and event.modifiers() & Qt.KeyboardModifier.ControlModifier:
            if self.user["username"] != "guest":
                self.user_page()
        elif event.key() == Qt.Key.Key_A and event.modifiers() & Qt.KeyboardModifier.ControlModifier:
            if self.scroll:
                for button in self.scroll.widget().findChildren(FileButton):
```

```python
            if button.id and button not in self.currently_selected:
                self.select_item(button)
        elif event.key() == Qt.Key.Key_S and event.modifiers() & Qt.KeyboardModifier.ControlModifier:
            if self.user["username"] != "guest":
                self.protocol.search()
        super().keyPressEvent(event)

    def save_sizes(self):
        """Stores the initial sizes and font sizes of all widgets for dynamic resizing."""
        for widget in self.findChildren(QWidget):
            font_size = widget.font().pointSize()
            self.original_sizes[widget] = {
                'geometry': widget.geometry(),
                'font_size': font_size
            }

    def moveEvent(self, event: QMoveEvent):
        """Updates the window geometry when moved."""
        self.window_geometry = self.geometry()

    def resizeEvent(self, event):
        """Dynamically resizes widgets based on the new window size."""
        new_width, new_height = self.width(), self.height()
        width_ratio, height_ratio = new_width / self.original_width, new_height / self.original_height

        for widget in self.findChildren(QWidget):
            if widget in self.original_sizes:
                original_geometry = self.original_sizes[widget]['geometry']
                original_font_size = self.original_sizes[widget]['font_size']

                new_x = int(original_geometry.x() * width_ratio) if width_ratio != 1 else original_geometry.x()
                new_width = int(original_geometry.width() * width_ratio) if width_ratio != 1 else
original_geometry.width()

                new_y = int(original_geometry.y() * height_ratio) if height_ratio != 1 else original_geometry.y()
                new_height = int(original_geometry.height() * height_ratio) if height_ratio != 1 else
original_geometry.height()

                self.window_geometry = self.geometry()
                widget.setGeometry(new_x, new_y, new_width, new_height)
                widget.updateGeometry()

                new_font_size = max(int(original_font_size * (width_ratio + height_ratio) / 2), 8)
                font = widget.font()
                font.setPointSize(new_font_size)
                widget.setFont(font)

                if isinstance(widget, QPushButton):
                    icon = widget.icon()
                    if not icon.isNull():
                        base = 60 if widget.text() == "" else 16
                        new_icon_size = int(base * (width_ratio + height_ratio) / 2)
                        widget.setIconSize(QSize(new_icon_size, new_icon_size))

        # Adjust scroll area size and file buttons
        try:
            if self.scroll:
                for button in self.scroll.widget().findChildren(FileButton):
                    for i in range(button.layout().count()):
                        label = button.layout().itemAt(i).widget()
                        if isinstance(label, QLabel):
                            font = label.font()
                            font.setPointSize(max(int(9 * (width_ratio + height_ratio) / 2), 8))
                            label.setFont(font)
                    button.setMinimumHeight(int(30 * height_ratio))
                self.scroll_size = [int(850 * width_ratio), int(340 * height_ratio)]
                self.scroll.setFixedSize(self.scroll_size[0], self.scroll_size[1])
        except:
            pass


    def main_page(self):
        """Loads the main page UI."""
        try:
            temp = self.window_geometry
            ui_path = f"{os.getcwd()}/gui/ui/main.ui"
            helper.update_ui_size(ui_path, self.window_geometry.width(), self.window_geometry.height())
            uic.loadUi(ui_path, self)

            self.save_sizes()

            self.signup_button.clicked.connect(self.signup_page)
            self.signup_button.setIcon(QIcon(ASSETS_PATH + "\\new_account.svg"))

            self.login_button.clicked.connect(self.login_page)
            self.login_button.setIcon(QIcon(ASSETS_PATH + "\\login.svg"))
```

```python
            self.exit_button.clicked.connect(self.protocol.exit_program)
            self.exit_button.setIcon(QIcon(ASSETS_PATH + "\\exit.svg"))

            self.setGeometry(temp)
            self.force_update_window()
        except:
            print(traceback.format_exc())

    def signup_page(self):
        """Loads the signup page UI."""
        try:
            temp = self.window_geometry
            ui_path = f"{os.getcwd()}/gui/ui/signup.ui"
            helper.update_ui_size(ui_path, self.window_geometry.width(), self.window_geometry.height())
            uic.loadUi(ui_path, self)

            self.save_sizes()

            self.password.setEchoMode(QLineEdit.EchoMode.Password)
            self.confirm_password.setEchoMode(QLineEdit.EchoMode.Password)

            self.password_toggle.clicked.connect(lambda: self.toggle_password(self.password))
            self.confirm_password_toggle.clicked.connect(lambda: self.toggle_password(self.confirm_password))

            self.signup_button.clicked.connect(lambda: self.protocol.signup(
                self.email.text(), self.username.text(), self.password.text(), self.confirm_password.text()))
            self.signup_button.setShortcut("Return")
            self.signup_button.setIcon(QIcon(ASSETS_PATH + "\\new_account.svg"))

            self.login_button.clicked.connect(self.login_page)
            self.login_button.setStyleSheet("background-color:transparent;color:royalblue;text-decoration:
underline;border:none;")

            self.back_button.clicked.connect(self.main_page)
            self.back_button.setIcon(QIcon(ASSETS_PATH + "\\back.svg"))

            self.setGeometry(temp)
            self.force_update_window()
        except:
            print(traceback.format_exc())

    def login_page(self):
        """Loads the login page UI."""
        try:
            temp = self.window_geometry
            ui_path = f"{os.getcwd()}/gui/ui/login.ui"
            helper.update_ui_size(ui_path, self.window_geometry.width(), self.window_geometry.height())
            uic.loadUi(ui_path, self)

            self.save_sizes()

            self.password.setEchoMode(QLineEdit.EchoMode.Password)
            self.password_toggle.clicked.connect(lambda: self.toggle_password(self.password))

            self.forgot_password_button.clicked.connect(self.forgot_password)
            self.forgot_password_button.setStyleSheet("background-color:transparent;color:royalblue;text-decoration:
underline;border:none;")

            self.signup_button.clicked.connect(self.signup_page)
            self.signup_button.setStyleSheet("background-color:transparent;color:royalblue;text-decoration:
underline;border:none;")

            self.login_button.clicked.connect(lambda: self.protocol.login(
                self.credi.text(), self.password.text(), self.remember_check.isChecked()))
            self.login_button.setShortcut("Return")
            self.login_button.setIcon(QIcon(ASSETS_PATH + "\\login.svg"))

            self.back_button.clicked.connect(self.main_page)
            self.back_button.setIcon(QIcon(ASSETS_PATH + "\\back.svg"))

            self.setGeometry(temp)
            self.force_update_window()
        except:
            print(traceback.format_exc())

    def forgot_password(self):
        """Loads the password recovery page UI."""
        try:
            temp = self.window_geometry
            ui_path = f"{os.getcwd()}/gui/ui/forgot_password.ui"
            helper.update_ui_size(ui_path, self.window_geometry.width(), self.window_geometry.height())
            uic.loadUi(ui_path, self)

            self.save_sizes()

            self.send_code_button.clicked.connect(lambda: self.protocol.reset_password(self.email.text()))
            self.send_code_button.setShortcut("Return")
```

```python
            self.send_code_button.setIcon(QIcon(ASSETS_PATH + "\\send.svg"))

            self.back_button.clicked.connect(self.login_page)
            self.back_button.setIcon(QIcon(ASSETS_PATH + "\\back.svg"))

            self.setGeometry(temp)
            self.force_update_window()
        except:
            print(traceback.format_exc())

    def verification_page(self, email):
        """Loads the account verification page UI."""
        try:
            temp = self.window_geometry
            ui_path = f"{os.getcwd()}/gui/ui/verification.ui"
            helper.update_ui_size(ui_path, self.window_geometry.width(), self.window_geometry.height())
            uic.loadUi(ui_path, self)

            self.save_sizes()

            self.verify_button.clicked.connect(lambda: self.protocol.verify(email, self.code.text()))
            self.verify_button.setShortcut("Return")
            self.verify_button.setIcon(QIcon(ASSETS_PATH + "\\verify.svg"))

            self.send_again_button.clicked.connect(lambda: self.protocol.send_verification(email))
            self.send_again_button.setIcon(QIcon(ASSETS_PATH + "\\again.svg"))

            self.back_button.clicked.connect(self.main_page)
            self.back_button.setIcon(QIcon(ASSETS_PATH + "\\back.svg"))

            self.setGeometry(temp)
            self.force_update_window()
        except:
            print(traceback.format_exc())

    def send_verification_page(self):
        """Loads the send verification email page UI."""
        try:
            temp = self.window_geometry
            ui_path = f"{os.getcwd()}/gui/ui/send_verification.ui"
            helper.update_ui_size(ui_path, self.window_geometry.width(), self.window_geometry.height())
            uic.loadUi(ui_path, self)

            self.save_sizes()

            self.send_code_button.clicked.connect(lambda: self.protocol.send_verification(self.email.text()))
            self.send_code_button.setShortcut("Return")
            self.send_code_button.setIcon(QIcon(ASSETS_PATH + "\\send.svg"))

            self.back_button.clicked.connect(self.main_page)
            self.back_button.setIcon(QIcon(ASSETS_PATH + "\\back.svg"))

            self.setGeometry(temp)
            self.force_update_window()
        except:
            print(traceback.format_exc())

    def user_page(self):
        """Loads and updates the user page UI."""
        self.update_user_page()
        self.run_user_page()

    def update_user_page(self):
        """Fetches updated file and directory listings for the user page."""
        self.files, self.directories = None, None
        self.protocol.get_used_storage()

        if self.user["cwd"] == "" and self.deleted:
            self.protocol.get_deleted_files(self.search_filter)
            self.protocol.get_deleted_directories(self.search_filter)
        elif self.user["cwd"] == "" and self.share:
            self.protocol.get_cwd_shared_files(self.search_filter)
            self.protocol.get_cwd_shared_directories(self.search_filter)
        else:
            self.protocol.get_cwd_files(self.search_filter)
            self.protocol.get_cwd_directories(self.search_filter)

    def run_user_page(self):
        """Loads and sets up the user page UI."""
        try:
            temp = self.window_geometry

            # Load user management and file navigation UI
            ui_path = f"{os.getcwd()}/gui/ui/account_managment.ui"
            helper.update_ui_size(ui_path, self.window_geometry.width(), self.window_geometry.height())
            uic.loadUi(ui_path, self)
```

```python
        ui_path = f"{os.getcwd()}/gui/ui/user.ui"
        helper.update_ui_size(ui_path, self.window_geometry.width(), self.window_geometry.height())
        uic.loadUi(ui_path, self)

        # Enable or disable file dropping based on user mode
        self.setAcceptDrops(not (self.share or self.deleted))
        if self.share:
            self.sort_widget.addItem(" Owner")

        self.set_cwd()

        # Hide progress indicators if no active uploads
        if not self.file_sending.active_threads:
            self.file_upload_progress.hide()
            self.stop_button.hide()

        self.currently_selected = []
        self.main_text.setText(f"Welcome {self.user['username']}")

        # Set storage limit
        self.storage_remaining.setMaximum(Limits(self.user["subscription_level"]).max_storage)
        self.set_used_storage()

        self.sort_widget.currentIndexChanged.connect(lambda: self.change_sort(self.sort.currentText()[1:]))

        # Configure UI buttons
        self.search_button.setIcon(QIcon(ASSETS_PATH + "\\search.svg"))
        self.search_button.setText(f" Search Filter: {self.search_filter}")
        self.search_button.clicked.connect(self.protocol.search)
        self.search_button.setStyleSheet("background-color:transparent;border:none;")

        self.refresh.setIcon(QIcon(ASSETS_PATH + "\\refresh.svg"))
        self.refresh.setText(" ")
        self.refresh.clicked.connect(self.user_page)
        self.refresh.setStyleSheet("background-color:transparent;border:none;")

        self.shared_button.clicked.connect(self.protocol.change_share)
        self.shared_button.setIcon(QIcon(ASSETS_PATH + "\\share.svg"))

        self.recently_deleted_button.clicked.connect(self.protocol.change_deleted)
        self.recently_deleted_button.setIcon(QIcon(ASSETS_PATH + "\\delete.svg"))

        self.user_button.clicked.connect(lambda: self.manage_account())
        self.logout_button.clicked.connect(self.protocol.logout)
        self.logout_button.setIcon(QIcon(ASSETS_PATH + "\\logout.svg"))
        self.upload_button.setIcon(QIcon(ASSETS_PATH + "\\upload.svg"))

        # Adjust upload button behavior based on view mode
        if self.deleted:
            try:
                self.upload_button.setIcon(QIcon(USER_ICON))
            except:
                pass
            self.upload_button.setText(" Your files")
            self.upload_button.clicked.connect(self.protocol.change_deleted)
            self.recently_deleted_button.hide()
            self.shared_button.hide()

        elif self.share:
            try:
                self.upload_button.setIcon(QIcon(USER_ICON))
            except:
                pass
            self.upload_button.setText(" Your files")
            self.upload_button.clicked.connect(self.protocol.change_share)
            self.shared_button.hide()
            self.recently_deleted_button.hide()

        else:
            self.upload_button.clicked.connect(lambda: self.file_dialog())

        self.user_button.setIconSize(QSize(self.user_button.size().width(), self.user_button.size().height()))
        self.user_button.setStyleSheet("padding:0px;border-radius:5px;border:none;background-color:transparent")

        try:
            self.user_button.setIcon(QIcon(USER_ICON))
        except:
            pass

        self.stop_button.clicked.connect(self.stop_upload)
        self.stop_button.setIcon(QIcon(ASSETS_PATH + "\\stop.svg"))

        self.setGeometry(temp)
        self.force_update_window()

    except:
```

```python
                print(traceback.format_exc())

    def draw_cwd(self):
        """Creates the file and directory listing in the user interface."""
        try:
            central_widget = self.centralWidget()
            outer_layout = QVBoxLayout()
            outer_layout.addStretch(1)

            # Create a scrollable area for files and directories
            scroll = QScrollArea()
            self.scroll = scroll
            scroll.setWidgetResizable(True)
            scroll.setVerticalScrollBarPolicy(Qt.ScrollBarPolicy.ScrollBarAlwaysOn)

            scroll_container_widget = QWidget()
            scroll_layout = QGridLayout()
            scroll_layout.setSpacing(5)

            # Add column headers
            if self.deleted:
                button = FileButton(self, ["File Name", "Deleted In", "Size"])
            elif self.share:
                button = FileButton(self, ["File Name", "Last Change", "Size", "Owner"])
            else:
                button = FileButton(self, ["File Name", "Last Change", "Size"])

            button.setStyleSheet("background-color:#001122;border-radius: 3px;border:1px solid darkgrey;")
            scroll_layout.addWidget(button)

            # Populate file entries
            for file in self.files:
                file = file.split("~")
                file_name, date, size, file_id = file[0], file[1][:-7], helper.format_file_size(int(file[2])), file[3]
                perms = file[5:]

                if self.share:
                    button = FileButton(self, f" {file_name} | {date} | {size} | {file[4]}".split("|"), file_id,
shared_by=file[4], perms=perms, size=int(file[2]), name=file_name)
                else:
                    button = FileButton(self, f" {file_name} | {date} | {size}".split("|"), file_id, size=int(file[2]),
name=file_name)

                button.clicked.connect(lambda checked, btn=button: self.select_item(btn))
                scroll_layout.addWidget(button)

            # Populate directory entries
            for directory in self.directories:
                directory = directory.split("~")
                dir_name, dir_id, last_change, size = directory[0], directory[1], directory[2][:-7],
helper.format_file_size(int(directory[3]))
                perms = directory[5:]

                if self.share:
                    button = FileButton(self, f" {dir_name} | {last_change} | {size} | {directory[4]}".split("|"), dir_id,
is_folder=True, shared_by=directory[2], perms=perms, size=int(directory[3]), name=dir_name)
                else:
                    button = FileButton(self, f" {dir_name} | {last_change} | {size}".split("|"), dir_id, is_folder=True,
size=int(directory[3]), name=dir_name)

                button.clicked.connect(lambda checked, btn=button: self.select_item(btn))
                scroll_layout.addWidget(button)

            # Handle empty directory
            if not self.directories and not self.files:
                button = FileButton(self, ["No files or folders in this directory"])
                button.setStyleSheet("background-color:red;border-radius: 3px;border:1px solid darkgrey;")
                scroll_layout.addWidget(button)

            # Add "Back" button if not at root
            if self.user["cwd"]:
                button = FileButton(self, ["Back"])
                button.clicked.connect(lambda: self.protocol.move_dir(self.user["parent_cwd"]))
                scroll_layout.addWidget(button)

            # Finalize scroll area
            scroll_container_widget.setLayout(scroll_layout)
            scroll.setWidget(scroll_container_widget)
            scroll.setFixedSize(850, 340)

            # Add scroll area to the layout
            spacer = QSpacerItem(20, 20, QSizePolicy.Policy.Minimum, QSizePolicy.Policy.Expanding)
            outer_layout.addItem(spacer)

            center_layout = QHBoxLayout()
            center_layout.addStretch(1)
            center_layout.addWidget(scroll)
```

```python
                center_layout.addStretch(1)

                outer_layout.addLayout(center_layout)
                outer_layout.addStretch(1)
                central_widget.setLayout(outer_layout)

        except:
            print(traceback.format_exc())


    def scroll_changed(self, value):
        """Handles scroll event to dynamically load more files if near the bottom."""
        self.scroll_progress = value
        total_scroll_height = self.scroll.verticalScrollBar().maximum()
        if total_scroll_height == 0: return
        if self.scroll_progress / total_scroll_height > 0.95 and len(self.directories) + len(self.files) <
int(self.items_amount):
            self.current_files_amount += ITEMS_TO_LOAD
            self.user_page()

    def manage_account(self):
        """Loads the account management page."""
        try:
            temp = self.window_geometry
            ui_path = f"{os.getcwd()}/gui/ui/account_managment.ui"
            helper.update_ui_size(ui_path, self.window_geometry.width(), self.window_geometry.height())
            uic.loadUi(ui_path, self)
            self.save_sizes()

            self.forgot_password_button.clicked.connect(lambda: self.protocol.reset_password(self.user["email"]))
            self.forgot_password_button.setIcon(QIcon(ASSETS_PATH + "\\key.svg"))

            self.delete_account_button.clicked.connect(lambda: self.protocol.delete_user(self.user["email"]))
            self.delete_account_button.setIcon(QIcon(ASSETS_PATH + "\\delete.svg"))

            self.upload_icon_button.clicked.connect(lambda: self.protocol.upload_icon())
            self.upload_icon_button.setIcon(QIcon(ASSETS_PATH + "\\profile.svg"))

            self.subscriptions_button.clicked.connect(self.subscriptions_page)
            self.subscriptions_button.setIcon(QIcon(ASSETS_PATH + "\\upgrade.svg"))

            self.change_username_button.clicked.connect(self.protocol.change_username)
            self.change_username_button.setIcon(QIcon(ASSETS_PATH + "\\change_user.svg"))

            if self.user["admin_level"] > 0:
                self.admin_button.setIcon(QIcon(ASSETS_PATH + "\\admin.svg"))
                self.admin_button.clicked.connect(self.admin_page)
                self.admin_button.setStyleSheet("background-color:transparent;border:none;")
            else:
                self.admin_button.hide()

            self.back_button.clicked.connect(self.user_page)
            self.back_button.setIcon(QIcon(ASSETS_PATH + "\\back.svg"))

            self.setGeometry(temp)
            self.force_update_window()
        except:
            print(traceback.format_exc())

    def admin_page(self):
        """Loads the admin page."""
        try:
            if self.user["admin_level"] <= 0:
                self.user_page()
                return

            temp = self.window_geometry
            ui_path = f"{os.getcwd()}/gui/ui/admin.ui"
            helper.update_ui_size(ui_path, self.window_geometry.width(), self.window_geometry.height())
            uic.loadUi(ui_path, self)
            self.save_sizes()

            self.protocol.admin_data()

            self.back_button.clicked.connect(self.manage_account)
            self.back_button.setIcon(QIcon(ASSETS_PATH + "\\back.svg"))

            self.setGeometry(temp)
            self.force_update_window()
        except:
            print(traceback.format_exc())

    def subscriptions_page(self):
        """Loads the subscription management page."""
        try:
            temp = self.window_geometry
            ui_path = f"{os.getcwd()}/gui/ui/subscription.ui"
```

```python
            helper.update_ui_size(ui_path, self.window_geometry.width(), self.window_geometry.height())
            uic.loadUi(ui_path, self)
            self.save_sizes()

            self.protocol.get_used_storage()

            self.back_button.clicked.connect(self.manage_account)
            self.back_button.setIcon(QIcon(ASSETS_PATH + "\\back.svg"))

            self.free_button.clicked.connect(lambda: self.protocol.subscribe(0))
            self.basic_button.clicked.connect(lambda: self.protocol.subscribe(1))
            self.premium_button.clicked.connect(lambda: self.protocol.subscribe(2))
            self.professional_button.clicked.connect(lambda: self.protocol.subscribe(3))

            # Disable and highlight the currently selected subscription level
            sub_buttons = {
                "0": self.free_button,
                "1": self.basic_button,
                "2": self.premium_button,
                "3": self.professional_button
            }
            if self.user["subscription_level"] in sub_buttons:
                sub_buttons[self.user["subscription_level"]].setDisabled(True)
                sub_buttons[self.user["subscription_level"]].setText("Selected")
                sub_buttons[self.user["subscription_level"]].setStyleSheet("background-color:dimgrey")

            self.storage_remaining.setMaximum(Limits(self.user["subscription_level"]).max_storage)
            self.set_used_storage()

            self.setGeometry(temp)
            self.force_update_window()
        except:
            print(traceback.format_exc())

    def recovery(self, email):
        """Loads the password recovery page."""
        try:
            temp = self.window_geometry
            ui_path = f"{os.getcwd()}/gui/ui/recovery.ui"
            helper.update_ui_size(ui_path, self.window_geometry.width(), self.window_geometry.height())
            uic.loadUi(ui_path, self)
            self.save_sizes()

            self.password.setEchoMode(QLineEdit.EchoMode.Password)
            self.confirm_password.setEchoMode(QLineEdit.EchoMode.Password)

            self.password_toggle.clicked.connect(lambda: self.toggle_password(self.password))
            self.confirm_password_toggle.clicked.connect(lambda: self.toggle_password(self.confirm_password))

            self.reset_button.clicked.connect(lambda: self.protocol.password_recovery(email, self.code.text(),
self.password.text(), self.confirm_password.text()))
            self.reset_button.setShortcut("Return")
            self.reset_button.setIcon(QIcon(ASSETS_PATH + "\\reset.svg"))

            self.send_again_button.clicked.connect(lambda: self.protocol.reset_password(email))
            self.send_again_button.setIcon(QIcon(ASSETS_PATH + "\\again.svg"))

            self.back_button.clicked.connect(self.manage_account)
            self.back_button.setIcon(QIcon(ASSETS_PATH + "\\back.svg"))

            self.setGeometry(temp)
            self.force_update_window()
        except:
            print(traceback.format_exc())

    def not_connected_page(self, connect=True):
        """Loads the not connected page and attempts reconnection."""
        try:
            temp = self.window_geometry
            ui_path = f"{os.getcwd()}/gui/ui/not_connected.ui"
            helper.update_ui_size(ui_path, self.window_geometry.width(), self.window_geometry.height())
            uic.loadUi(ui_path, self)
            self.save_sizes()

            self.ip.setText(self.protocol.ip)
            self.port.setText(str(self.protocol.port))

            self.connect_button.clicked.connect(lambda: self.protocol.connect_server(self.ip.text(), self.port.text(),
loop=True))
            self.connect_button.setShortcut("Return")
            self.connect_button.setIcon(QIcon(ASSETS_PATH + "\\connect.svg"))

            self.exit_button.clicked.connect(helper.force_exit)
            self.exit_button.setIcon(QIcon(ASSETS_PATH + "\\exit.svg"))

            self.setGeometry(temp)
            self.force_update_window()
```

```python
        if connect:
            self.protocol.connect_server(loop=True)

    except:
        print(traceback.format_exc())

def select_item(self, btn):
    """Handles selection of files and folders."""
    item_id = btn.id
    item_name = btn.name

    if btn in self.currently_selected and len(self.currently_selected) == 1:
        if btn.is_folder:
            self.protocol.move_dir(item_id)
            self.reset_selected()
        else:
            self.protocol.view_file(item_id, item_name, btn.file_size)
    elif helper.control_pressed() and btn not in self.currently_selected:
        self.currently_selected.append(btn)
    elif helper.control_pressed() and btn in self.currently_selected:
        self.currently_selected.remove(btn)
    else:
        self.reset_selected()
        self.currently_selected = [btn]

    # Update UI styling for selection
    for label in btn.lables:
        label.setObjectName("selected" if btn in self.currently_selected else ("folder-label" if btn.is_folder else
"file-label"))

    # Refresh UI stylesheet
    current_stylesheet = self.app.styleSheet()
    self.app.setStyleSheet("")
    self.app.setStyleSheet(current_stylesheet)

    self.force_update_window()



def finish_sending(self):
    """Clears the file queue and hides upload-related UI elements."""
    self.file_sending.file_queue = []
    try:
        self.stop_button.setEnabled(False)
        self.stop_button.hide()
    except:
        pass
    try:
        self.file_upload_progress.hide()
    except:
        pass

def update_progress(self, value):
    """Updates the file upload progress bar."""
    try:
        self.file_upload_progress.show()
    except:
        pass
    try:
        self.stop_button.setEnabled(True)
        self.stop_button.show()
    except:
        pass
    try:
        self.file_upload_progress.setValue(value)
    except:
        pass

def reset_progress(self, value):
    """Resets the file upload progress bar."""
    try:
        self.file_upload_progress.show()
    except:
        pass
    try:
        self.stop_button.setEnabled(True)
        self.stop_button.show()
    except:
        pass
    try:
        self.file_upload_progress.setMaximum(value)
    except:
        pass

def reset_selected(self):
    """Deselects all selected items."""
```

```python
        for btn in self.currently_selected:
            for label in btn.lables:
                try:
                    label.setObjectName("folder-label" if btn.is_folder else "file-label")
                except RuntimeError:
                    if label in self.currently_selected:
                        self.currently_selected.remove(label)
        self.currently_selected = []

    def confirm_account_deletion(self, email):
        """Prompts the user to confirm account deletion."""
        confirm_email = dialogs.new_name_dialog("Delete Account", "Enter account email:")
        if email == confirm_email:
            return True
        else:
            self.set_error_message("Entered email does not match account email")

    def activate_file_view(self, file_id):
        """Opens the file viewer and checks for modifications."""
        save_path = self.files_downloading[file_id].save_location
        file_hash = helper.compute_file_md5(save_path)
        file_viewer.FileViewer(save_path, "File Viewer")

        if file_hash != helper.compute_file_md5(save_path):
            save = dialogs.show_confirmation_dialog("Do you want to save changes?")
            if save:
                self.file_sending.file_queue.append(save_path)
                self.file_sending.send_files("UPFL", file_id)
            else:
                os.remove(save_path)
        else:
            os.remove(save_path)

    def toggle_password(self, text):
        """Toggles password visibility in password fields."""
        text.setEchoMode(QLineEdit.EchoMode.Password if text.echoMode() == QLineEdit.EchoMode.Normal else
QLineEdit.EchoMode.Normal)

    def file_dialog(self):
        """Opens a file dialog to select files for uploading."""
        try:
            file_paths, _ = QFileDialog.getOpenFileNames(self, "Open File", "", "All Files (*);;Text Files (*.txt)")
            if file_paths:
                self.file_sending.file_queue.extend(file_paths)
                self.file_sending.send_files()
        except:
            print(traceback.format_exc())

    def dragEnterEvent(self, event: QDragEnterEvent):
        """Handles drag enter event to allow file dropping."""
        if event.mimeData().hasUrls():
            event.acceptProposedAction()

    def dropEvent(self, event: QDropEvent):
        """Handles file drop event and queues files for upload."""
        if event.mimeData().hasUrls():
            file_paths = [url.toLocalFile() for url in event.mimeData().urls()]
            self.file_sending.file_queue.extend(file_paths)
            self.file_sending.send_files()

    def change_sort(self, new_sort):
        """Changes the sorting method and reloads the file list."""
        if self.sort == new_sort:
            self.sort_direction = not self.sort_direction
        self.sort = new_sort
        self.user_page()

    def set_error_message(self, msg):
        """Displays an error message in the UI."""
        try:
            if hasattr(self, "message"):
                self.message.setStyleSheet("color: red;")
                self.message.setText(msg)
        except:
            pass

    def set_message(self, msg):
        """Displays a success message in the UI."""
        try:
            if hasattr(self, "message"):
                self.message.setStyleSheet("color: lightgreen;")
                self.message.setText(msg)
        except:
            pass

    def set_cwd(self):
        """Updates the displayed current working directory path."""
```

```python
        if hasattr(self, "cwd"):
            self.cwd.setStyleSheet("color: yellow;")
            if self.share:
                self.cwd.setText(f"Shared > {" > ".join(self.user['cwd_name'].split('\\'))}"[:-3])
            elif self.deleted:
                self.cwd.setText(f"Deleted > {" > ".join(self.user['cwd_name'].split('\\'))}"[:-3])
            else:
                self.cwd.setText(f"{self.user['username']} > {" > ".join(self.user['cwd_name'].split('\\'))}"[:-3])

    def set_used_storage(self):
        """Updates the displayed storage usage."""
        self.storage_remaining.setValue(int(self.used_storage))
        self.storage_label.setText(f"Storage used ({helper.format_file_size(self.used_storage * 1_000_000)} /
{Limits(self.user['subscription_level']).max_storage // 1000} GB):")

    def stop_upload(self):
        """Stops the current file upload."""
        self.stop_button.setEnabled(False)
        if self.file_sending.active_threads:
            self.file_sending.active_threads[0].running = False
        self.protocol.send_data(b"STOP|" + self.uploading_file_id.encode())

    def force_update_window(self):
        """Forces a UI update to apply changes."""
        size = self.size()
        resize_event = QResizeEvent(size, size)
        self.resizeEvent(resize_event)

    def update_current_files(self):
        """Refreshes the file list UI based on the current sorting method."""
        self.sort_widget.currentIndexChanged.disconnect()

        sort_map = {"Name": 0, "Date": 1, "Type": 2, "Size": 3, "Owner": 4}
        if self.sort in sort_map:
            self.sort_widget.setCurrentIndex(sort_map[self.sort])

        self.save_sizes()
        self.draw_cwd()
        self.sort_widget.currentIndexChanged.connect(lambda: self.change_sort(self.sort_widget.currentText()[1:]))
        self.scroll.verticalScrollBar().setMaximum(self.scroll_progress)
        self.scroll.verticalScrollBar().setValue(self.scroll_progress)
        self.scroll.verticalScrollBar().valueChanged.connect(self.scroll_changed)


    def share_file(self, file_id, user_cred, file_name, read="False", write="False", delete="False", rename="False",
download="False", share="False"):
        """Displays a dialog to set file sharing permissions."""
        temp_app = QApplication.instance()
        if temp_app is None:
            temp_app = QApplication([])

        dialog = QDialog()
        dialog.setWindowTitle("File Share Options")
        dialog.setStyleSheet("font-size:15px;")
        dialog.resize(600, 400)

        # Group checkboxes for permission settings
        permissions_group = QGroupBox(f"File sharing permissions for {file_name} with {user_cred}")
        permissions_layout = QGridLayout()

        read_cb = QCheckBox("Read")
        read_cb.setChecked(read == "True")
        permissions_layout.addWidget(read_cb, 0, 0)

        write_cb = QCheckBox("Write")
        write_cb.setChecked(write == "True")
        permissions_layout.addWidget(write_cb, 0, 1)

        delete_cb = QCheckBox("Delete")
        delete_cb.setChecked(delete == "True")
        permissions_layout.addWidget(delete_cb, 1, 0)

        rename_cb = QCheckBox("Rename")
        rename_cb.setChecked(rename == "True")
        permissions_layout.addWidget(rename_cb, 1, 1)

        download_cb = QCheckBox("Download")
        download_cb.setChecked(download == "True")
        permissions_layout.addWidget(download_cb, 2, 0)

        share_cb = QCheckBox("Share")
        share_cb.setChecked(share == "True")
        permissions_layout.addWidget(share_cb, 2, 1)

        permissions_group.setLayout(permissions_layout)

        # Submit button
```

```python
        submit_btn = QPushButton("Submit")
        submit_btn.setShortcut("Return")
        submit_btn.clicked.connect(lambda: self.protocol.send_share_premissions(
            dialog, file_id, user_cred,
            read_cb.isChecked(), write_cb.isChecked(), delete_cb.isChecked(),
            rename_cb.isChecked(), download_cb.isChecked(), share_cb.isChecked()
        ))

        # Layout setup
        button_layout = QHBoxLayout()
        button_layout.addSpacerItem(QSpacerItem(40, 20, QSizePolicy.Policy.Expanding, QSizePolicy.Policy.Minimum))
        button_layout.addWidget(submit_btn)

        main_layout = QVBoxLayout()
        main_layout.addWidget(permissions_group)
        main_layout.addLayout(button_layout)

        dialog.setLayout(main_layout)
        dialog.exec()

    def check_all_perms(self, perm):
        """Checks if all selected items have the specified permission."""
        return all(button.perms[perm] == "True" for button in self.currently_selected)

    def check_all_id(self):
        """Ensures all selected items have valid IDs."""
        return all(button.id is not None for button in self.currently_selected)

    def remove_selected(self, button):
        """Removes a button from the selected list."""
        if button in self.currently_selected:
            self.currently_selected.remove(button)


class FileButton(QPushButton):
    """Represents a file or folder button in the UI."""
    def __init__(self, window, text, id=None, parent=None, is_folder=False, shared_by=None, perms=None, size=0, name=""):
        super().__init__("|".join(text), parent)
        self.id = id
        self.is_folder = is_folder
        self.shared_by = shared_by
        self.perms = perms or ["True", "True", "True", "True", "True", "True"]
        self.file_size = size
        self.name = name
        self.window = window
        self.lables = []

        self.setMinimumHeight(30)
        self.setSizePolicy(QSizePolicy.Policy.Expanding, QSizePolicy.Policy.Expanding)
        button_layout = QHBoxLayout()
        button_layout.setContentsMargins(0, 0, 0, 0)
        button_layout.setSpacing(0)

        # Create labels for file button
        for i, label_text in enumerate(text):
            label = QLabel(label_text)
            if i == 0:
                if self.is_folder:
                    label.setText(f' <img src="{ASSETS_PATH + "\\folder.svg"}" width="20" height="20">'
                                  f'<label> {helper.truncate_label(label, label_text)}</label>')
                elif self.id:
                    icon_path = ASSETS_PATH + "\\file_types\\" + helper.format_file_type(label_text.split("~")
[0].split(".")[-1][:-1]) + ".svg"
                    if not os.path.isfile(icon_path):
                        icon_path = ASSETS_PATH + "\\file.svg"
                    label.setText(f' <img src="{icon_path}" width="16" height="20">'
                                  f' {helper.truncate_label(label, label_text)}')

            if self.id is None:
                label.setAlignment(Qt.AlignmentFlag.AlignCenter)
                if label_text not in ["Back", "No files or folders in this directory"]:
                    sort_key = ["Name", "Date", "Size", "Owner"][i] if i < 4 else None
                    if sort_key:
                        label.mousePressEvent = lambda event, key=sort_key: self.window.change_sort(key)
                        if sort_key == self.window.sort:
                            label.setText(f'<img src="{ASSETS_PATH}\\{"asc.svg" if self.window.sort_direction else
"dsc.svg"}" width="20" height="20">'
                                          f'<label>  {label_text}</label>')

            # Set label styling
            label.setObjectName("folder-label" if self.is_folder else "file-label" if self.id else "back-label" if
label_text == "Back" else "")
            button_layout.addWidget(label, stretch=1)
            self.lables.append(label)

        # Adjust layout spacing
        button_layout.setStretch(0, 2)
```

```python
        for i in range(1, len(text)):
            button_layout.setStretch(i, 1)

        self.setLayout(button_layout)


    def contextMenuEvent(self, event: QContextMenuEvent):
        """Creates a context menu for file and folder actions."""
        menu = QMenu(self)

        # Add download option if all selected items are valid and have download permission
        if self.window.check_all_id() and self.window.check_all_perms(4) and not self.window.deleted and
self.window.currently_selected:
            action = menu.addAction(" Download")
            action.triggered.connect(self.window.protocol.download)
            action.setIcon(QIcon(ASSETS_PATH + "\\download.svg"))

        if self.window.check_all_id() and self.window.currently_selected:
            # Add delete option if all selected items have delete permission
            if self.window.check_all_perms(2):
                if (self.window.deleted and self.window.user["cwd"] == "") or not self.window.deleted:
                    action = menu.addAction(" Delete")
                    action.triggered.connect(self.window.protocol.delete)
                    action.setIcon(QIcon(ASSETS_PATH + "\\delete.svg"))

            # Add rename option if a single item is selected and rename is allowed
            if self.window.check_all_perms(3) and not self.window.deleted and len(self.window.currently_selected) == 1:
                action = menu.addAction(" Rename")
                action.triggered.connect(self.rename)
                action.setIcon(QIcon(ASSETS_PATH + "\\change_user.svg"))

            # Add share option if sharing is allowed
            if self.window.check_all_perms(5) and not self.window.deleted:
                action = menu.addAction(" Share")
                action.triggered.connect(self.window.protocol.share_action)
                action.setIcon(QIcon(ASSETS_PATH + "\\share.svg"))

            # Add remove from share option if the user is in shared files view
            if self.window.share and self.window.user["cwd"] == "" and not self.window.deleted:
                action = menu.addAction(" Remove")
                action.triggered.connect(self.window.protocol.remove)
                action.setIcon(QIcon(ASSETS_PATH + "\\remove.svg"))

        # Add new folder option if the user is not in shared or deleted mode
        if not self.window.share and not self.window.deleted:
            action = menu.addAction(" New Folder")
            action.triggered.connect(self.window.protocol.new_folder)
            action.setIcon(QIcon(ASSETS_PATH + "\\new_account.svg"))

        # Add recover option if user is in the deleted files view
        if self.window.deleted and self.window.user["cwd"] == "" and self.window.currently_selected:
            action = menu.addAction(" Recover")
            action.triggered.connect(self.window.protocol.recover)
            action.setIcon(QIcon(ASSETS_PATH + "\\new_account.svg"))

        # Add search option
        action = menu.addAction(" Search")
        action.triggered.connect(self.window.protocol.search)
        action.setIcon(QIcon(ASSETS_PATH + "\\search.svg"))

        # Display the menu at the cursor position
        menu.exec(event.globalPos())

    def rename(self):
        """Prompts the user to enter a new name and sends a rename request."""
        name = self.text().split(" | ")[0][1:]  # Extracts the current file name
        new_name = dialogs.new_name_dialog("Rename", "Enter new file name:", name)
        if new_name:
            self.window.protocol.send_data(b"RENA|" + self.id.encode() + b"|" + name.encode() + b"|" + new_name.encode())
```