МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В. И. Ульянова (ЛЕНИНА)
Кафедра информационной безопасности

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Модели безопасности компьютерных систем»

Тема: Модель Белла-Лападулы

Студентки гр. 1361 _____ Галунина Е.С.

_____ Токарева У.В.

Преподаватели _____ Шкляр Е.В.

_____ Шульженко А.Д.

Санкт-Петербург

2024

ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Студентки: Галунина Е.С., Токарева У.В.

Группа 1361

Тема: Информационный поток по памяти

Задача:

Разработать программу, управляющую уровнями секретности папок в файловой системе и контролирующую информационные потоки между ними.

Основные функции программы делятся на две части – работа с уровнями секретности и копирование файлов.

Работа с уровнями секретности:

1) Создание (с заданным именем), изменение (имя и секретность) и удаление уровней секретности. Все изменения в уровнях секретности внутри программы должны влиять на папки, которым этим уровни назначены.

2) Могут существовать уровни секретности, которые не установлены ни для одной папки.

3) Создание (с заданным именем), переименование и удаление папок и подпапок внутри заданной корневой папки.

4) Новые папки должны создаваться с минимальным доступным уровнем секретности.

5) Как следствие, в списке всегда должен быть хотя бы один уровень секретности, который считается минимальным.

6) Выбор уровня секретности для папок и подпапок из нескольких заданных в программе вариантов.

Копирование файлов:

1) Копирование файлов между папками согласно модели Белла — Лападулы. При этом копируются все файлы, лежащие внутри папки.

2) Выбор папок для копирования происходит в интерфейсе программы. При этом должны отображаться только папки, подходящие под выбранный уровень секретности.

3) Предусмотрите работу со вложенностью папок — папки разных уровней секретности могут быть вложенными друг в друга, копирование должно учитывать эту ситуацию.

Корнем файловой системы в этой работе считается заранее выбранная папка на ваше усмотрение — например, "C:\\LAB4".

Обратите внимание: в этой работе вы имеете дело с настоящими папками и файлами в файловой системе, но уровнями секретности и копированием управляет ваша программа. Не нужно придумывать, как реализовать «настоящие» уровни секретности для папок внутри ОС, достаточно хранить и обрабатывать список уровней секретности в привязке к папкам где-то в системе.

Требования к программе: приложение с пользовательским интерфейсом, язык программирования любой, все поля подписаны, предусмотрена обработка ошибок. Программа должна быть скомпилирована в exe-файл, иметь иконку и открываться отдельно от среды разработки.

Дата сдачи отчета: 22.05.2024

Дата защиты отчета: 22.05.2024

Студентки	_____	Галунина Е.С.
	_____	Токарева У.В.
Преподаватели	_____	Шкляр Е.В.
	_____	Шульженко А.Д.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

S — множество субъектов; O — множество объектов;

$R = \{read, write, append), execute\}$ — множество видов доступа и видов прав доступа;

$B = \{b \subseteq S \acute{} O \acute{} R\}$ — множество возможных множеств текущих доступов в системе;

$(L, \leq)$ — решетка уровней конфиденциальности, например: $L = \{U(unclussified), C\ (confidential), S\ (secret), TS\ (top\ secret)\}$, где $U < C < S < TS$;

$M = \{m_{|S| \acute{} |O|}\}$ — множество возможных матриц доступов, где $m_{|S| \acute{} |O|}$ — матрица доступов, $m[s, o] \subseteq R$ — права доступа субъекта s к объекту o;

$(fs, fo, fc) \in F = L\ s\ x\ L\ o\ x\ L\ s$ — тройка функций (fs, fo, fc), задающих: $fs: S \to L$ — уровень доступа субъекта; $fo: O \to L$ — уровень конфиденциальности объекта; $fc: S \to L$ — текущий уровень доступа субъекта, при этом для любого $s \in S$ выполняется неравенство $fc(s) \leq fs(s)$;

$V = B\ x\ M\ x\ F$ — множество состояний системы; Q — множество запросов системе;

D — множество ответов по запросам, например: $D = \{yes, no, error\}$;

$W \subseteq Q\ x\ D\ x\ V\ x\ V$ — множество действий системы, где четверка $(q, d, v*, v) \in W$ означает, что система по запросу q с ответом d перешла из состояния v в состояние $v*$;

$N0 = \{0, 1, 2, \dots\}$ — множество значений времени;

X — множество функций $x: N0 \to Q$, задающих все возможные последовательности запросов к системе;

Y — множество функций $y: N0 \to D$, задающих все возможные последовательности ответов системы по запросам;

Z — множество функций $z: N0 \to V$, задающих все возможные последовательности состояний системы.

Определение. $\sum(Q, D, W, z0) \subseteq X\ x\ Y\ x\ Z$ называется системой, если для каждого $(x, y, z) \in \sum(Q, D, W, z0)$ выполняется условие: для $t \in N0, (xt, yt, zt + 1, zt) \in W$, где $z0$ начальное состояние системы. При этом каждый набор $(x, y, z) \in \sum(Q, D, W, z0)$ называется реализацией системы, а $(xt, yt, zt + 1, zt) \in W$ называется действием системы в момент времени $t \in N0$.

В классической модели Белла-ЛаПадулы рассматриваются следующие запросы, входящие во множество Q:

1) Запросы изменения множества текущих доступов b;
2) Запросы изменения функций f;
3) Запросы изменения прав доступа в матрице m.

Изменение текущих доступов:

- Получить доступ (добавить тройку (субъект, объект, вид доступа) в текущее множество доступов b);
- Отменить доступ (удалить тройку из текущего множества доступов b).

Изменение значений функций уровней конфиденциальности и доступа:

- Изменить уровень конфиденциальности объекта;
- Изменить уровень доступа субъекта. Изменение прав доступа:
- Дать разрешение на доступ (добавить право доступа в соответствующий элемент матрицы доступов m);
- Отменить разрешение на доступ (удалить право доступа из соответствующего элемента матрицы доступов m).

Безопасность системы определяется с помощью трех свойств:

ss – свойства простой безопасности (simple security);

$*$ – свойства звезда;

ds - свойства дискреционной безопасности (discretionary security).

ХОД РАБОТЫ

Для выполнения лабораторной работы был выбран язык программирования Python.

1. Интерфейс программы представлен на рисунке 1.

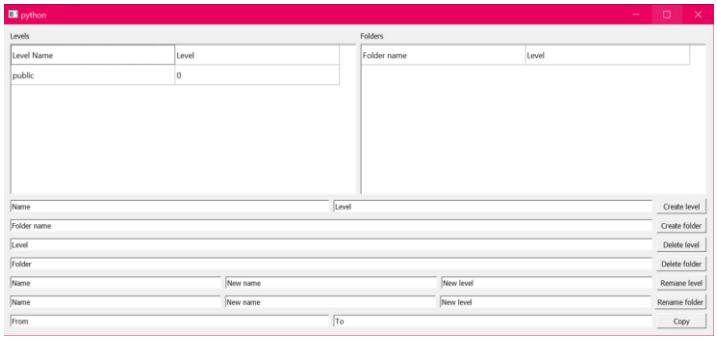

Рисунок 1 – Интерфейс программы

2. На рисунке 2 показано создание уровней конфиденциальности. Для этого необходимо ввести в соответствующие поля название и секретность, а затем нажать на кнопку. После этого можно увидеть появившиеся уровни секретности.

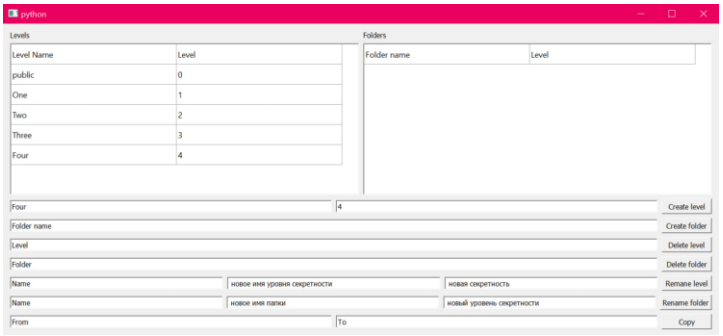

Рисунок 2 – Создание уровней конфиденциальности

3. Теперь аналогично создаем папку в корневой системе – рисунок 3. Каждая папка изначально имеет уровень конфиденциальности public.

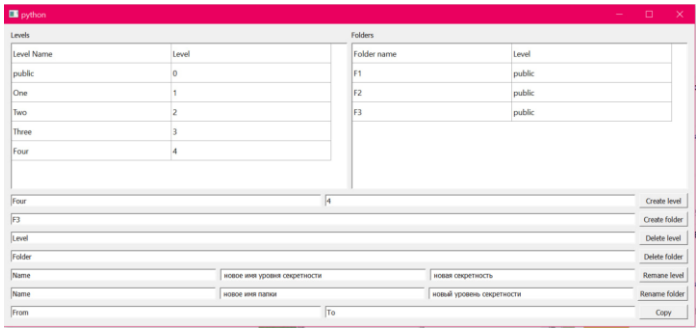

Рисунок 3 – Создание папки в корневой системе

4. Теперь поменяем уровень конфиденциальности. Данное действие показано на рисунке 4.

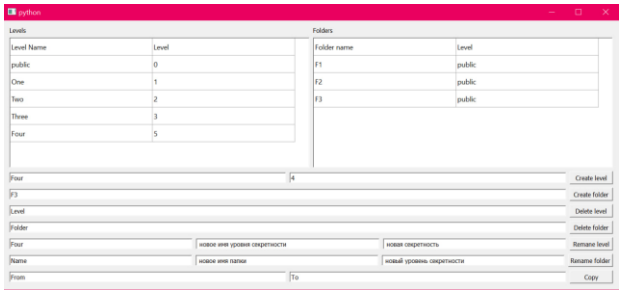

Рисунок 4 – Изменение уровня конфиденциальности

5. Далее переименовываем уровень конфиденциальности – рисунок 5.

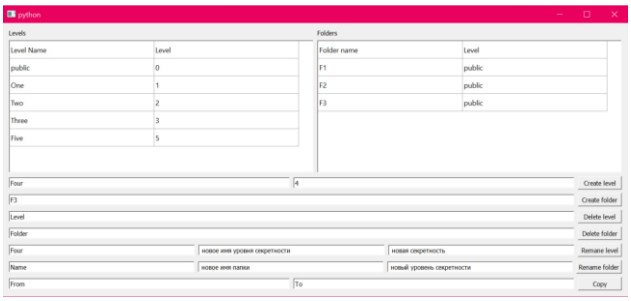

Рисунок 5 – Переименование уровня конфиденциальности

6. Работа также должна предусматривать возможность удаления уровней конфиденциальности. Данное действие представлено на рисунке 6.

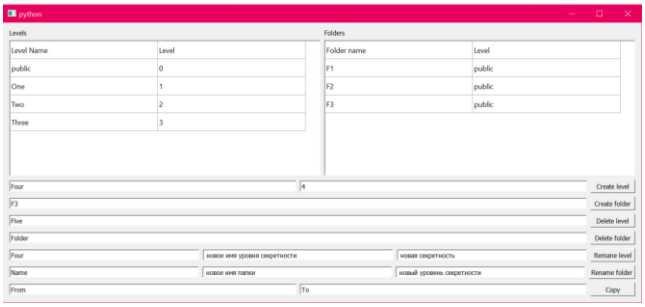

Рисунок 6 – Удаление уровня конфиденциальности

7. Теперь переименуем папку – рисунок 7.

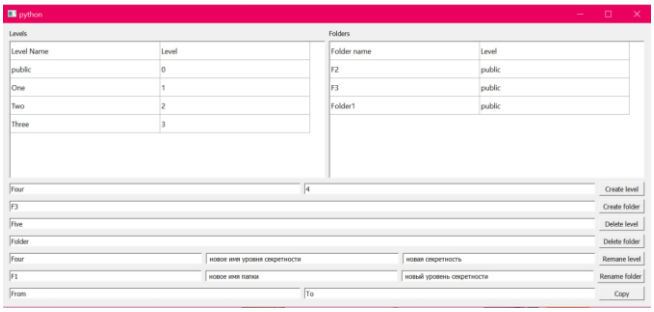

Рисунок 7 – Переименование папки

8. На рисунке 8 показана смена уровня конфиденциальности для папки.

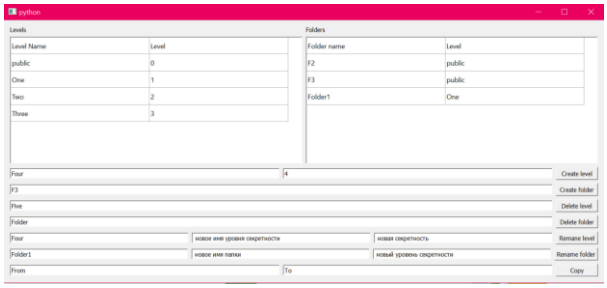

Рисунок 8 – Смена уровня конфиденциальности папки

9. Создание подпапки в папке показано на рисунке 9. Также мы можем увидеть, что уровень секретности подпапки равен уровню конфиденциальности папки.

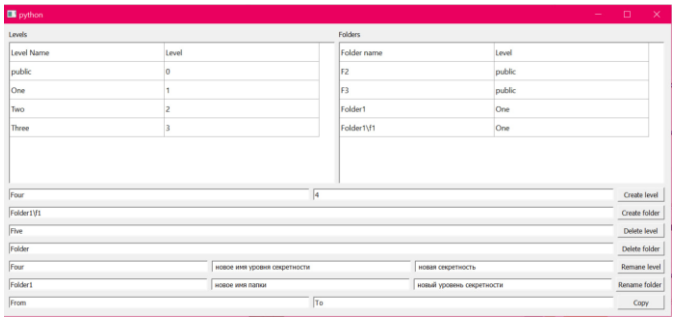

Рисунок 9 – Создание подпапки в папке

10. Результат копирования из папки с большим уровнем конфиденциальности в меньший показан на рисунке 10.

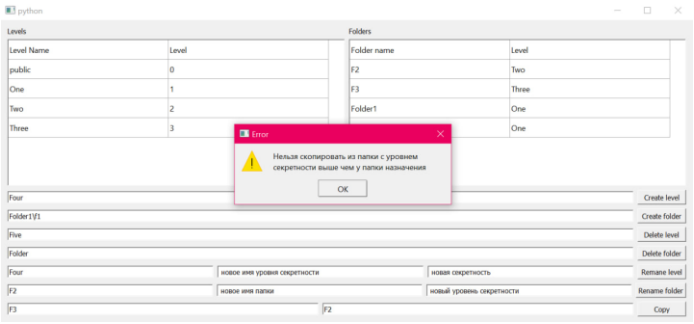

Рисунок 10 – Копирование из папки с большим уровнем конфиденциальности в меньший

11. Теперь скопируем из папки с меньшим уровнем секретности

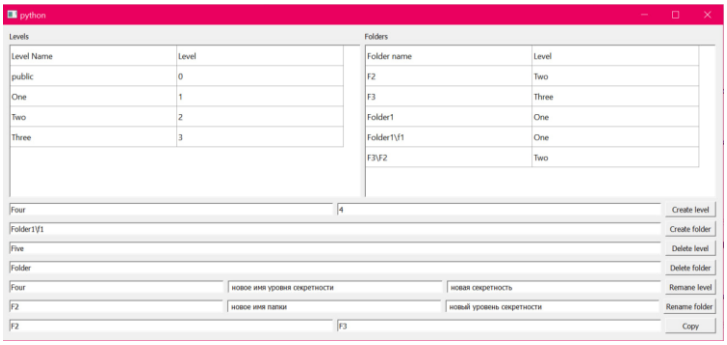

Рисунок 11 –Копирование из папки с меньшим уровнем конфиденциальности

12. Теперь изменим уровень секретности вложенной папки

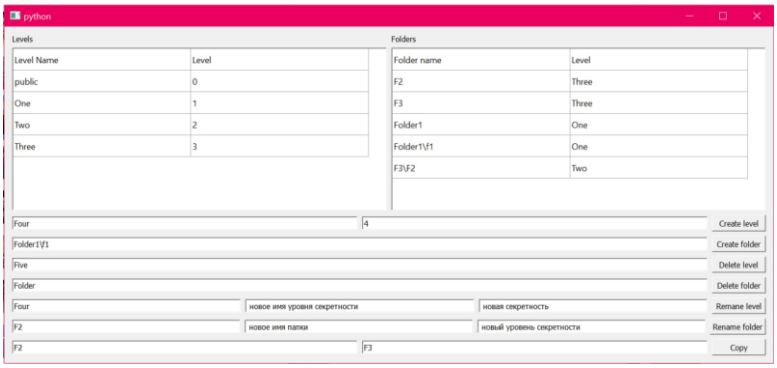

Рисунок 12 – Изменение уровней секретности у папок

13. Удаляем уровень секретности

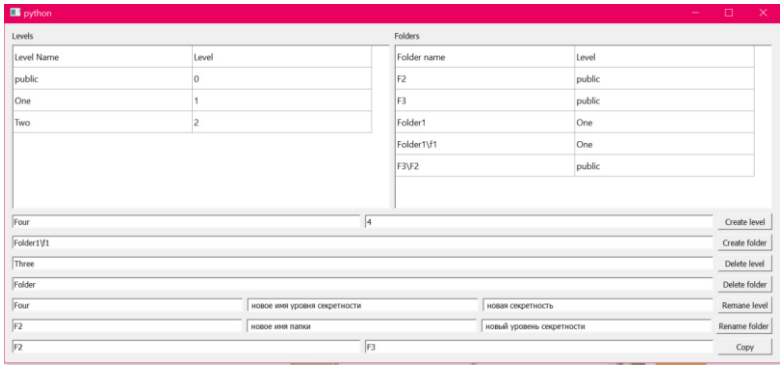

Рисунок 13 – Удаление уровня секретности

ВЫВОД

В ходе выполнения лабораторной работы была написана программа на языке программирования Python, управляющая уровнями секретности папок в файловой системе. Данная программа позволяет копировать файлы с учетом уровней секретности по модели Белла-Лападулы. Она позволяет создавать, изменять и удалять уровни секретности и папки. Также программа может назначать уровни секретности уже созданным папкам и копировать одну папку в другую, учитывая их уровни секретности и их вложенность.

ПРИЛОЖЕНИЕ 1. ИСХОДНЫЙ КОД ПРОГРАММЫ

```
# Импорт необходимых модулей
import sys
import json
import os
import shutil
import re

# импорт компонентов для графического интерфейса
from PyQt5 import QtCore
from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout,
QLabel, \
    QLineEdit, QPushButton, QHBoxLayout, QMessageBox, \
    QTableWidget,      QTableWidgetItem,      QAbstractItemView,
QListWidget, QListWidgetItem, QFrame

# Путь к файлу JSON
access_levels_json_path = ".\\accessLevelsData.json"
folders_json_path = ".\\foldersData.json"

main_directory = 'C:\\4'

access_levels = {"public": "0"} # Уровни секретности
folders = {} # Папки

# Проверка корректности имени папки
def is_valid_folder_name(name):
    if re.search(r'[.<>:"/|?*]', name) is not None:  # Проверка на
наличие недопустимых символов
        return False
    if name.startswith(' ') or name.endswith(' '):             #
Проверка на наличие пробелов в начале или конце имени
        return False
```

```
    if not name.strip():                                    #
Проверка на то, что имя папки не пустое и не состоит только из
пробелов
        return False
    return True                                             # Все
проверки прошли успешно

class MainWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.init_ui()

    # Описание интерфейса
    def init_ui(self):
        # Устанавливаем размеры окна
        self.setMinimumWidth(1600)
        self.setMinimumHeight(700)
        # Вертикальный макет
        layout = QVBoxLayout()
        # Горизонтальный макет
        extra_layout = QHBoxLayout()



        # Макет для списка уровней доступа
        access_levels_list_layout = QVBoxLayout()
        self.access_levels_list_name    =    QLabel('Levels')    #
Заголовок
        self.access_levels_list = QTableWidget()                #
Таблица
        self.access_levels_list.setMinimumSize(750,          300)
# Размеры
        # Скрываем заголовки
        self.access_levels_list.horizontalHeader().hide()
        self.access_levels_list.verticalHeader().hide()
        # Отключаем редактирование
```

```
self.access_levels_list.setEditTriggers(QAbstractItemView.NoEdit
Triggers)
        # Итнегрируем в макет
        extra_layout.addLayout(access_levels_list_layout)

access_levels_list_layout.addWidget(self.access_levels_list_name
)

access_levels_list_layout.addWidget(self.access_levels_list)




        # Макет для списка файлов
        folders_list_layout = QVBoxLayout()
        self.folders_list_name = QLabel('Folders')    # Заголовок
        self.folders_list = QTableWidget()          # Таблица
        self.folders_list.setMinimumSize(750, 300)  # Размеры
        #Скрываем заголовки
        self.folders_list.horizontalHeader().hide()
        self.folders_list.verticalHeader().hide()
        # Отключаем редактирование

self.folders_list.setEditTriggers(QAbstractItemView.NoEditTrigge
rs)
        # Интегрируем в макет
        folders_list_layout.addWidget(self.folders_list_name)
        folders_list_layout.addWidget(self.folders_list)
        extra_layout.addLayout(folders_list_layout)
        #Подключаем в макет
        layout.addLayout(extra_layout)



        #Описание поля для созадния уровней
```

```
        self.access_level_name = QLineEdit('Name')
        self.access_level_num = QLineEdit('Level')
        self.access_level_create_btn    =    QPushButton('Create
level')

self.access_level_create_btn.clicked.connect(self.create_access_
level)
        create_layout = QHBoxLayout()
        layout.addLayout(create_layout)
        create_access_level_layout = QHBoxLayout()

create_access_level_layout.addWidget(self.access_level_name)

create_access_level_layout.addWidget(self.access_level_num)

create_access_level_layout.addWidget(self.access_level_create_bt
n)
        layout.addLayout(create_access_level_layout)

        # Описание поля для созадния папок
        self.folder_name = QLineEdit('Folder name')
        self.folder_create_btn = QPushButton('Create folder')

self.folder_create_btn.clicked.connect(self.create_folder)
        create_folder_main_layout = QVBoxLayout()
        create_folder_layout = QHBoxLayout()
        create_folder_layout.addWidget(self.folder_name)
        create_folder_layout.addWidget(self.folder_create_btn)

create_folder_main_layout.addLayout(create_folder_layout)
        layout.addLayout(create_folder_main_layout)

        # Удаление уровней
        self.access_level_delete_name = QLineEdit('Level')
```

```
        self.access_level_delete_btn    =    QPushButton('Delete
level')

self.access_level_delete_btn.clicked.connect(self.delete_access_
level)
        delete_access_level_layout = QHBoxLayout()

delete_access_level_layout.addWidget(self.access_level_delete_na
me)

delete_access_level_layout.addWidget(self.access_level_delete_bt
n)
        layout.addLayout(delete_access_level_layout)
        # Удаление уровней
        self.folder_delete_name = QLineEdit('Folder')
        self.folder_delete_btn = QPushButton('Delete folder')

self.folder_delete_btn.clicked.connect(self.delete_folder)
        delete_layout = QHBoxLayout()
        layout.addLayout(delete_layout)
        delete_folder_layout = QHBoxLayout()
        delete_folder_layout.addWidget(self.folder_delete_name)
        delete_folder_layout.addWidget(self.folder_delete_btn)
        layout.addLayout(delete_folder_layout)



        # Изменение уровней
        self.access_level_edit_name_old = QLineEdit('Name')
        self.access_level_edit_name_new = QLineEdit('New name')
        self.access_level_edit_num_new = QLineEdit('New level')
        self.access_level_edit_btn = QPushButton('Remane level')

self.access_level_edit_btn.clicked.connect(self.edit_access_leve
l)
        edit_access_level_layout = QHBoxLayout()
```

```
edit_access_level_layout.addWidget(self.access_level_edit_name_o
ld)

edit_access_level_layout.addWidget(self.access_level_edit_name_n
ew)

edit_access_level_layout.addWidget(self.access_level_edit_num_ne
w)

edit_access_level_layout.addWidget(self.access_level_edit_btn)
        layout.addLayout(edit_access_level_layout)


        # Изменение уровней
        self.folder_edit_name_old = QLineEdit('Name')
        self.folder_edit_name_new = QLineEdit('New name')
        self.folder_edit_access_level_new    =    QLineEdit('New
level')
        self.folder_edit_btn = QPushButton('Rename folder')
        self.folder_edit_btn.clicked.connect(self.edit_folder)
        edit_layout = QHBoxLayout()
        layout.addLayout(edit_layout)
        edit_folder_layout = QHBoxLayout()
        edit_folder_layout.addWidget(self.folder_edit_name_old)
        edit_folder_layout.addWidget(self.folder_edit_name_new)

edit_folder_layout.addWidget(self.folder_edit_access_level_new)
        edit_folder_layout.addWidget(self.folder_edit_btn)
        layout.addLayout(edit_folder_layout)

        # Копирование
        self.folder_to_copy = QLineEdit('From')
        self.folder_destination = QLineEdit('To')
        self.folder_copy_btn = QPushButton('Copy')
```

```
        self.folder_copy_btn.clicked.connect(self.copy_folder)
        copy_layout = QHBoxLayout()
        copy_layout.addWidget(self.folder_to_copy)
        copy_layout.addWidget(self.folder_destination)
        copy_layout.addWidget(self.folder_copy_btn)
        layout.addLayout(copy_layout)
        self.setLayout(layout)

    # Отображение ошибок
    def show_error(self, msg_text):
        print(msg_text)
        msg = QMessageBox()
        msg.setIcon(QMessageBox.Warning)
        msg.setWindowTitle("Error")
        msg.setText(msg_text)
        msg.setStandardButtons(QMessageBox.Ok)
        msg.exec_()

    # Работа со словарем
    def is_in_dict(self, item, dict):
        existing_flag = False
        if len(dict) > 0:
            if item in dict:
                existing_flag = True
        return existing_flag

    def process_changes(self, old_str, new_str):
        # Разбиваем строку old_str на части по символу обратного
слэша \\
        old_parts = old_str.split('\\')
        # Аналогично разбиваем строку new_str на части
        new_parts = new_str.split('\\')

        # Определяем индекс последнего вхождения символа обратного
слэша в old_str
```

```
        last_index = old_str.count('\\')

        # Проходим по меньшему количеству частей между old_parts
и new_parts
        for i in range(min(len(old_parts), len(new_parts))):
            # Если текущий индекс больше или равен индексу
последнего вхождения обратного слэша,
            # заменим соответствующую часть old_parts на часть из
new_parts
            if i >= last_index:
                old_parts[i] = new_parts[i]

        # Соединяем обработанные части обратным слэшем в одну
строку
        processed_str = '\\'.join(old_parts)
        # Возвращаем итоговую строку
        return processed_str

    def can_edit_folder_access_level(self, folder_name, level):
        flag = True
        if folder_name.count('\\') > 0:
            folder_name_parts = folder_name.split('\\')
            folder_name_parts.pop(len(folder_name_parts) - 1)
            parent_folder_name = '\\'.join(folder_name_parts)
            if int(access_levels[folders[parent_folder_name]]) >=
int(level):
                for f in folders:
                    a = folder_name.count('\\')
                    b = f.count('\\')
                    if (folder_name in f) and ((b - a) == 1):
                        if              int(level)              <
int(access_levels[folders[f]]):
                            flag = False
                            break
            else:
```

```
                flag = False
        else:
            for f in folders:
                a = folder_name.count('\\')
                b = f.count('\\')
                if (folder_name in f) and ((b - a) == 1):
                    if                int(level)                <
int(access_levels[folders[f]]):
                        flag = False
                        break

        return flag

    def can_create_folder(self, folder_name):
        flag = False
        if folder_name.count('\\') > 0:
            for folder in folders:
                if folder in folder_name:
                    folder_name_parts = folder_name.split('\\')
                    folder_name_parts.pop(len(folder_name_parts)
- 1)
                    parent_folder_name                          =
'\\'.join(folder_name_parts)
                    if folder == parent_folder_name:
                        flag = True
        else:
            flag = True
        return flag

    def can_edit_access_level(self, level_name, level):
        folders_to_check = []
        for folder in folders:
            if folders[folder] == level_name:
                folders_to_check.append(folder)
```

```
        can_edit = True

        for folder in folders_to_check:
            if folder.count('\\') == 0:
                for f in folders:
                    a = folder.count('\\')
                    b = f.count('\\')
                    if (folder in f) and ((b - a) == 1):
                        if              int(level)              <
int(access_levels[folders[f]]):
                            can_edit = False
            else:
                folder_name_parts = folder.split('\\')
                folder_name_parts.pop(len(folder_name_parts) - 1)
                parent_folder_name = '\\'.join(folder_name_parts)
                if
int(access_levels[folders[parent_folder_name]]) >= int(level):
                    for f in folders:
                        a = folder.count('\\')
                        b = f.count('\\')
                        if (folder in f) and ((b - a) == 1):
                            if            int(level)            <
int(access_levels[folders[f]]):
                                can_edit = False
                else:
                    can_edit = False

        return can_edit

    def ui_reset(self):
        self.access_level_edit_name_new.setText('    новое    имя
уровня секретности')
        self.access_level_edit_num_new.setText('            новая
секретность')
        self.folder_edit_name_new.setText(' новое имя папки')
```

```
        self.folder_edit_access_level_new.setText(' новый уровень
секретности')

    # основные функции
    def create_access_level(self):
        name = self.access_level_name.text()
        num = self.access_level_num.text()
        if name.isalnum() and (len(name) > 0):
            if not (self.is_in_dict(name, access_levels)):
                if num.isdigit() and (len(num) > 0) and (int(num)
<= 10 and int(num) >= 0):
                    access_levels[name] = num
                    # обновление ui
                else:
                    self.show_error("Неккоректное        значение
секретности")
            else:
                self.show_error("Уровень     секретности     уже
существует")
        else:
            self.show_error("Неккоректное       имя       уровня
секретности")
        print(access_levels)
        json_update()
        self.access_levels_table_update()
        self.folders_table_update()
        self.ui_reset()

    def create_folder(self):
        name = self.folder_name.text()
        if self.is_in_dict(name, folders):
            self.show_error("Папка уже существует")
            return
        if is_valid_folder_name(name) and (len(name) > 0):
            if self.can_create_folder(name):
```

```
                # Получаем уровень секретности родительской папки
                parent_folder_name = os.path.dirname(name)
                if parent_folder_name in folders:
                    parent_level = folders[parent_folder_name]
                else:
                    parent_level = 'public'  # По умолчанию, если нет родительской папки, уровень секретности будет 'public'

                # Присваиваем уровень секретности родительской папки новой папке
                folders[name] = parent_level
                script_dir = main_directory
                new_folder_path = os.path.join(script_dir, name)
                if not (os.path.exists(new_folder_path) and os.path.isdir(new_folder_path)):
                    os.makedirs(new_folder_path)
            else:
                self.show_error("Вы не можете создать сразу несколько папок")
        else:
            self.show_error("Неккоректное имя папки")
        print(folders)
        json_update()
        self.access_levels_table_update()
        self.folders_table_update()
        self.ui_reset()

    def delete_access_level(self):
        name = self.access_level_delete_name.text()
        if name == 'public':
            self.show_error("Нельзя удалить уровень секретности public")
            return
        for folder in folders:
```

```
            if name in folder and folders[folder] != 'public' and
folder != name:
                self.show_error("Нельзя удалить секретность")
                return
        parent_folders = []
        if name.isalnum() and (len(name) > 0):
            if self.is_in_dict(name, access_levels):
                for folder in folders:
                    if folders[folder] == name:
                        folders[folder] = 'public'
                        parent_folders.append(folder)
                del access_levels[name]

                for folder in folders:
                    for parent_folder in parent_folders:
                        if parent_folder in folder:
                            folders[folder] = 'public'

            else:
                self.show_error("Нет уровня секретности с таким
именем")
        else:
            self.show_error("Неккоректное название уровня
секретности")
        json_update()
        self.access_levels_table_update()
        self.folders_table_update()
        self.ui_reset()

    def delete_folder(self):
        name = self.folder_delete_name.text()
        if is_valid_folder_name(name) and (len(name) > 0):
            if self.is_in_dict(name, folders):
                script_dir = main_directory
                folder_path = os.path.join(script_dir, name)
```

```
                shutil.rmtree(folder_path)

                update_dict = folders.copy()

                for folder in folders:
                    if name in folder:
                        del update_dict[folder]

                folders.clear()
                folders.update(update_dict)

            else:
                self.show_error("Нет папки с таким именем")
        else:
            self.show_error("Неккоректное имя папки")
        print(folders)
        json_update()
        self.access_levels_table_update()
        self.folders_table_update()
        self.ui_reset()

    def edit_access_level(self):
        name_old = self.access_level_edit_name_old.text()
        name_new = self.access_level_edit_name_new.text()
        num = self.access_level_edit_num_new.text()
        if name_old.isalnum() and (len(name_old) > 0):
            if self.is_in_dict(name_old, access_levels):
                if (name_new == ' новое имя уровня секретности')
or (name_new == ''):
                    if (num == ' новую секретность') or (num ==
''):
                        old_num = access_levels[name_old]
                        access_levels[name_old] = old_num
                    else:
```

```
                        if num.isdigit() and (len(num) > 0) and
(int(num) <= 10 and int(num) >= 0):
                            if
self.can_edit_access_level(name_old, num):
                                access_levels[name_old] = num
                            else:
                                self.show_error(
                                    "Нельзя задать уровень
секретности больше, чем у родительской папки/меньше, чем у
дочерней")
                        else:
                            self.show_error("Неккоректное
значение секретности")
                else:
                    if name_new.isalnum() and (len(name_new) > 0):
                        if not (self.is_in_dict(name_new,
access_levels)):
                            if (num == ' новый уровень
секретности') or (num == ''):
                                old_num = access_levels[name_old]
                                del access_levels[name_old]
                                access_levels[name_new] = old_num
                                if len(folders) > 0:
                                    for folder in folders:
                                        if folders[folder] ==
name_old:
                                            folders[folder] =
name_new
                            else:
                                if num.isdigit() and (len(num) >
0) and (int(num) <= 10 and int(num) >= 0):
                                    if
self.can_edit_access_level(name_old, num):
                                        del
access_levels[name_old]
```

```
                                        access_levels[name_new] =
num
                                        if len(folders) > 0:
                                            for     folder     in
folders:
                                                if
folders[folder] == name_old:

folders[folder] = name_new
                                    else:
                                        self.show_error(
                                            "Нельзя        задать
уровень секретности больше, чем у родительской папки/меньше, чем
у дочерней")
                                else:

self.show_error("Неккоректное значение секретности")
                        else:
                            self.show_error("Уровень  секретности
уже существует")
                    else:
                        self.show_error("Неккоректное имя уровня
секретности")
            else:
                self.show_error("Нет уровня секретности с таким
именем")
        else:
            self.show_error("Неккоректное       имя       уровня
секретности")
        print(access_levels)
        json_update()
        self.access_levels_table_update()
        self.folders_table_update()
        self.ui_reset()
```

```
    def edit_folder(self):
        name_old = self.folder_edit_name_old.text()
        name_new = self.folder_edit_name_new.text()
        level = self.folder_edit_access_level_new.text()
        if is_valid_folder_name(name_old) and (len(name_old) > 0):
            if self.is_in_dict(name_old, folders):
                if (name_new == ' новое имя папки') or (name_new
== ''):
                    if (level == ' новый уровень секретности') or
(level == ''):
                        old_level = folders[name_old]
                        folders[name_old] = old_level
                    else:
                        if level.isalnum() and (len(level) > 0):
                            if             self.is_in_dict(level,
access_levels):
                                if
self.can_edit_folder_access_level(name_old,
access_levels[level]):
                                    folders[name_old] = level
                                else:
                                    self.show_error(
                                        "Нельзя задать уровень
секретности больше, чем у родительской папки/меньше, чем у
дочерней")
                            else:
                                self.show_error("Нет       уровня
секретности с таким именем")
                        else:
                            self.show_error("Неккоректное     имя
уровня секретности")
                else:
                    if     is_valid_folder_name(name_new)     and
(len(name_new) > 0):
```

```
                        if     not     (self.is_in_dict(name_new,
folders)):
                            if   (level   ==   '   новый   уровень
секретности') or (level == ''):
                                processed_str                   =
self.process_changes(name_old, name_new)
                                if name_old != processed_str:
                                    old_level = folders[name_old]
                                    del folders[name_old]
                                    folders[name_new] = old_level
                                else:
                                    self.show_error("Вы    можете
изменять только крайние папки в пути!")
                                    return

                                updated_dict = {}
                                for      key,      value      in
folders.items():
                                    if name_old in key:
                                        if name_new == key:
                                            updated_dict[key]   =
value
                                        else:
                                            new_key = key
                                            new_key             =
new_key.replace(name_old, name_new)

updated_dict[new_key] = value
                                    else:
                                        updated_dict[key] = value
                                folders.clear()
                                folders.update(updated_dict)

                                script_dir = main_directory
```

```
                                old_folder_path                 =
os.path.join(script_dir, name_old)
                                new_folder_path                 =
os.path.join(script_dir, name_new)
                                os.rename(old_folder_path,
new_folder_path)
                            else:
                                if      level.isalnum()      and
(len(level) > 0):
                                    if     self.is_in_dict(level,
access_levels):
                                        processed_str           =
self.process_changes(name_old, name_new)
                                        if      name_old      !=
processed_str:
                                            if
self.can_edit_folder_access_level(name_old,
access_levels[level]):
                                                del
folders[name_old]

folders[name_new] = level

                                                updated_dict = {}
                                                for key, value in
folders.items():
                                                    if   name_old
in key:
                                                        new_key =
key.replace(name_old, name_new)

updated_dict[new_key] = value
                                                    else:

updated_dict[key] = value
```

```
                                                folders.clear()

folders.update(updated_dict)
                                            else:
                                                self.show_error(
                                                    "Нельзя дать
уровень секретности с значением больше, чем у родительской папки")
                                        else:
                                            self.show_error("Вы
можете изменять только крайние папки в пути!")
                                            return

                                        script_dir              =
main_directory
                                        old_folder_path         =
os.path.join(script_dir, name_old)
                                        new_folder_path         =
os.path.join(script_dir, name_new)

os.rename(old_folder_path, new_folder_path)
                                    else:
                                        self.show_error("Нет
уровня секретности с таким именем")
                                else:

self.show_error("Неккоректное имя уровня секретности")
                            else:
                                self.show_error("Папка            уже
существует")
                        else:
                            self.show_error("Неккоректное имя папки")
                else:
                    self.show_error("Нет папки с таким именем")
            else:
                self.show_error("Неккоректное имя папки")
```

```
        print(folders)
        json_update()
        self.access_levels_table_update()
        self.folders_table_update()
        self.ui_reset()

    def access_levels_table_update(self):
        self.access_levels_list.clear()
        for j in range(0, self.access_levels_list.rowCount()):
            self.access_levels_list.removeRow(j - 1)
        for k in range(0, self.access_levels_list.columnCount()):
            self.access_levels_list.removeColumn(k - 1)
        self.access_levels_list.setColumnCount(2)
        self.access_levels_list.setColumnWidth(0, 370)
        self.access_levels_list.setColumnWidth(1, 370)
        self.access_levels_list.setRowCount(1)

        self.access_levels_list.setItem(0,                    0,
QTableWidgetItem("Level Name"))
        self.access_levels_list.setItem(0,                    1,
QTableWidgetItem("Level"))

        i = 1
        for access_level in access_levels:

self.access_levels_list.insertRow(self.access_levels_list.rowCou
nt())
            self.access_levels_list.setItem(i,                0,
QTableWidgetItem(access_level))
            self.access_levels_list.setItem(i,                1,
QTableWidgetItem(access_levels[access_level]))
            i += 1

    def folders_table_update(self):
        self.folders_list.clear()
```

```
        for j in range(0, self.folders_list.rowCount()):
            self.folders_list.removeRow(j - 1)
        for k in range(0, self.folders_list.columnCount()):
            self.folders_list.removeColumn(k - 1)
        self.folders_list.setColumnCount(2)
        self.folders_list.setColumnWidth(0, 370)
        self.folders_list.setColumnWidth(1, 370)
        self.folders_list.setRowCount(1)

        self.folders_list.setItem(0, 0, QTableWidgetItem("Folder
name"))
        self.folders_list.setItem(0,                          1,
QTableWidgetItem("Level"))

        i = 1
        for folder in folders:

self.folders_list.insertRow(self.folders_list.rowCount())
            self.folders_list.setItem(i,                      0,
QTableWidgetItem(folder))
            self.folders_list.setItem(i,                      1,
QTableWidgetItem(folders[folder]))
            i += 1

    def copy_folder(self):
        source = self.folder_to_copy.text()
        destination = self.folder_destination.text()
        fileToCopy = source.split('\\')[-1]
        if fileToCopy in destination:
            return
        main_patch = ''
        folders_to_copy = []

        if source.count('\\') > 0:
            patch_from_splitted = source.split('\\')
```

```
            patch_from_splitted.pop(len(patch_from_splitted) - 1)
            patch_from = '\\'.join(patch_from_splitted)
            main_patch = patch_from + '\\'

        if is_valid_folder_name(source):
            if self.is_in_dict(source, folders):
                print('source')
                print(source)
            else:
                self.show_error("Такой папки не существует")
                return
        else:
            self.show_error("Неккоректное название папки для
копирования")
            return

        if is_valid_folder_name(destination):
            if self.is_in_dict(destination, folders):
                print('destination')
                print(destination)
            else:
                self.show_error("Такой папки не существует")
                return
        else:
            self.show_error("Неккоректное название папки
назначения")
            return

        if source.count('\\') == 0:
            if      int(access_levels[folders[source]])      <=
int(access_levels[folders[destination]]):

                print('Copy')
                new_folders = []
```

```
                for folder in folders:
                    if folder.startswith(source):
                        folders_to_copy.append(folder)

                for f_t_c in folders_to_copy:
                    new_folders.append(destination + '\\' +
f_t_c.replace(main_patch, ''))
                    folders[destination + '\\' +
f_t_c.replace(main_patch, '')] = folders[f_t_c]

                print('new_folders')
                print(new_folders)

                for folder in new_folders:
                    script_dir = main_directory
                    new_folder_path = os.path.join(script_dir,
folder)
                    if not (os.path.exists(new_folder_path) and
os.path.isdir(new_folder_path)):
                        os.makedirs(new_folder_path)

                for folder in new_folders:
                    script_dir = main_directory
                    folder_path = os.path.join(script_dir,
folder)
                    if self.is_in_dict(destination + '\\' +
folder.replace(main_patch, ''), folders):
                        new_folder_path =
os.path.join(script_dir,

destination + '\\' + folder.replace(main_patch, ''))
                    else:
                        new_folder_path =
os.path.join(script_dir, destination)
                    print(folder_path)
```

```
                    print(new_folder_path)
                    list = os.listdir(folder_path)
                    print(list)
                    if len(list) > 0:
                        for item in list:
                            if '.' in item:
                                print(item)
                                shutil.copy(folder_path + '\\' +
item, new_folder_path + '\\' + item)

            else:
                self.show_error('Нельзя скопировать из папки с
уровнем секретности выше чем у папки назначения')
        else:
            parent_folders = []
            folder_name_parts = source.split('\\')
            while (len(folder_name_parts) > 1):
                folder_name_parts.pop(len(folder_name_parts) - 1)
                parent_folder_name = '\\'.join(folder_name_parts)
                parent_folders.append(parent_folder_name)

            can_copy = True
            for parent in parent_folders:
                if     int(access_levels[folders[parent]])     >
int(access_levels[folders[destination]]):
                    can_copy = False
            if can_copy:
                if     int(access_levels[folders[source]])     <=
int(access_levels[folders[destination]]):
                    print('Copy')
                    new_folders = []

                    for folder in folders:
                        if folder.startswith(source):
                            folders_to_copy.append(folder)
```

```
                    for f_t_c in folders_to_copy:
                        new_folders.append(destination + '\\' +
f_t_c.replace(main_patch, ''))
                        folders[destination     +     '\\'     +
f_t_c.replace(main_patch, '')] = folders[f_t_c]

                    print('new_folders')
                    print(new_folders)

                    for folder in new_folders:
                        script_dir = main_directory
                        new_folder_path                          =
os.path.join(script_dir, folder)
                        if  not  (os.path.exists(new_folder_path)
and os.path.isdir(new_folder_path)):
                            os.makedirs(new_folder_path)

                    for folder in new_folders:
                        script_dir = main_directory
                        folder_path   =   os.path.join(script_dir,
folder)
                        if  self.is_in_dict(destination  +  '\\'  +
folder.replace(main_patch, ''), folders):
                            new_folder_path                      =
os.path.join(script_dir,

destination + '\\' + folder.replace(main_patch, ''))
                        else:
                            new_folder_path                      =
os.path.join(script_dir, destination)
                        print(folder_path)
                        print(new_folder_path)
                        list = os.listdir(folder_path)
                        print(list)
```

```
                        if len(list) > 0:
                            for item in list:
                                if '.' in item:
                                    print(item)
                                    shutil.copy(folder_path     +
'\\' + item, new_folder_path + '\\' + item)

                else:
                    self.show_error('Нельзя скопировать из папки
с уровнем секретности выше чем у папки назначения')
            else:
                self.show_error('Нельзя скопировать из папки с
уровнем секретности выше чем у папки назначения')

        json_update()
        self.access_levels_table_update()
        self.folders_table_update()
        self.ui_reset()

    def copy_folder_ex(self):
        source = self.folder_to_copy.text()
        destination = self.folder_destination.text()

        if is_valid_folder_name(source):
            if self.is_in_dict(source, folders):
                print(source)
            else:
                self.show_error("Такой папки не существует")
                return
        else:
            self.show_error("Неккоректное    имя    папки    для
копирования")
            return

        if is_valid_folder_name(destination):
```

```
            if self.is_in_dict(destination, folders):
                print(destination)
            else:
                self.show_error("Такой папки не существует")
                return
        else:
            self.show_error("Неккоректное имя папки назначения")
            return

        for_folders = []
        not_for_files = []
        for_files = []

        main_patch = ''

        if source.count('\\') > 0:
            patch_from_splitted = source.split('\\')
            patch_from_splitted.pop(len(patch_from_splitted) - 1)
            patch_from = '\\'.join(patch_from_splitted)
            print(patch_from)
            main_patch = patch_from + '\\'
            if  not  (int(access_levels[folders[patch_from]])  <=
int(access_levels[folders[destination]])):
                self.show_error("Нельзя  копировать  в  папку  с
меньшим уровнем секретности")
                return

        print('*копирование*')
        for folder in folders:
            if folder.startswith(source):
                if     int(access_levels[folders[folder]])     <=
int(access_levels[folders[destination]]):
                    for_files.append(folder)
                else:
                    not_for_files.append(folder)
```

```
        for folder in folders:
            for from_for_files in for_files:
                if     (from_for_files     !=     folder)     and
folder.startswith(from_for_files):
                    patch_from_splitted = folder.split('\\')

patch_from_splitted.pop(len(patch_from_splitted) - 1)
                    patch_from = '\\'.join(patch_from_splitted)
                    for_folders.append(patch_from)
                    break

        extra_for_files = []
        for f in for_files:
            for n_f in not_for_files:
                if f.startswith(n_f) and (n_f != f):
                    extra_for_files.append(f.replace(n_f  +  '\\',
'',))
                else:
                    extra_for_files.append(f)

        print('for_files')
        print(for_files)
        print('not_for_files')
        print(not_for_files)
        print('for_folders')
        print(for_folders)

        new_folders = []
        for f_fol in for_folders:
            new_folders.append(destination      +      '\\'      +
f_fol.replace(main_patch, ''))
            folders[destination          +          '\\'          +
f_fol.replace(main_patch, '')] = folders[f_fol]
```

```
        print('new_folders')
        print(new_folders)

        for folder in new_folders:
            script_dir = main_directory
            new_folder_path = os.path.join(script_dir, folder)
            if    not    (os.path.exists(new_folder_path)    and
os.path.isdir(new_folder_path)):
                os.makedirs(new_folder_path)

        for folder in for_files:
            script_dir = main_directory
            folder_path = os.path.join(script_dir, folder)
            if    self.is_in_dict(destination    +    '\\'    +
folder.replace(main_patch, ''), folders):
                new_folder_path    =    os.path.join(script_dir,
destination + '\\' + folder.replace(main_patch, ''))
            else:
                new_folder_path     =     new_folder_path     =
os.path.join(script_dir, destination)
            print(folder_path)
            print(new_folder_path)
            list = os.listdir(folder_path)
            print(list)
            if len(list) > 0:
                for item in list:
                    if '.' in item:
                        print(item)
                        shutil.copy(folder_path  +  '\\'  +  item,
new_folder_path + '\\' + item)

        json_update()
        self.access_levels_table_update()
        self.folders_table_update()
```

```
def json_update():
    # Создание данных для записи в JSON файл
    data = {"access_levels": access_levels}

    # Запись данных в JSON файл
    with open(access_levels_json_path, "w") as json_file:
        json.dump(data, json_file)
        json_file.close()
    print(f"Данные записаны в JSON файл:
{access_levels_json_path}")

    # Создание данных для записи в JSON файл
    data = {"folders": folders}

    # Запись данных в JSON файл
    with open(folders_json_path, "w") as json_file:
        json.dump(data, json_file)
        json_file.close()
    print(f"Данные записаны в JSON файл: {folders_json_path}")

def get_data_from_json():
    # Чтение данных из JSON файла при старте программы
    data_from_json = {}
    try:
        with open(access_levels_json_path, "r") as json_data:
            data_from_json = json.load(json_data)
            json_data.close()
    except FileNotFoundError:
        print("Файл не найден.")
        return

    for access_level in data_from_json["access_levels"]:
        access_levels[access_level] =
data_from_json["access_levels"][access_level]
```

```
    try:
        with open(folders_json_path, "r") as json_data:
            data_from_json = json.load(json_data)
            json_data.close()
    except FileNotFoundError:
        print("Файл не найден.")
        return

    for folder in data_from_json["folders"]:
        folders[folder] = data_from_json["folders"][folder]

if __name__ == '__main__':
    app = QApplication(sys.argv)
    app.setStyle('Windows')

    window = MainWindow()
    window.show()
    get_data_from_json()
    window.access_levels_table_update()
    window.folders_table_update()
    sys.exit(app.exec_())
```