**Задание 1: Управление проектной структурой и файловой системой**

Texto

Descripción generada automáticamente

**Задание 2: Чтение, преобразование и сериализация данных**

Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Texto, Aplicación

Descripción generada automáticamente

Una captura de pantalla de una computadora

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Una captura de pantalla de una computadora

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

import os

from datetime import datetime

import chardet

import json

class Data:

    def \_\_init\_\_(self, file\_name, text, text\_processed, file\_size, last\_change\_time):

        self.file\_name = file\_name

        self.text = text

        self.text\_processed = text\_processed

        self.file\_size = file\_size

        self.last\_change\_time = last\_change\_time

def data\_to\_dict(data):

    return {

        'file\_name': data.file\_name,

        'text': data.text,

        'text\_processed': data.text\_processed,

        'file\_size': data.file\_size,

        'last\_change\_time': data.last\_change\_time

    }

def create\_project\_structure():

    # Define the directory structure

    dirs = [

        "project\_root/data/raw",

        "project\_root/data/processed",

        "project\_root/logs",

        "project\_root/backups",

        "project\_root/output"

    ]

    # Create directories if they do not exist

    for dir in dirs:

        os.makedirs(dir, exist\_ok=True)

        log(f"Directory '{dir}' created or already exists.")

def log(message):

    # Get current date and time

    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    # Write log message to file

    with open('project\_root/logs/project\_log.txt', 'a', encoding='utf-8') as logfile:

        logfile.write(f"{timestamp} - {message}\n")

def create\_example\_files():

    # Create example files with different encodings

    files = [

        ('project\_root/data/raw/example1.txt', 'Hello, World!', 'utf-8'),

        ('project\_root/data/raw/example2.txt', 'Привет, мир!', 'utf-8'),

        ('project\_root/data/raw/example3.txt', '¡Hola mundo!', 'iso-8859-1'),

        ('project\_root/data/raw/example4.txt', 'Hola mundo!', 'ascii'),

        ('project\_root/data/raw/example5.txt', 'Hello, World', 'ascii'),

    ]

    for filepath, content, encoding in files:

        with open(filepath, 'w', encoding=encoding) as file:

            file.write(content)

            log(f"File '{filepath}' created with encoding '{encoding}' and content: {content}")

    log("Example files created successfully.")

    return files

def process\_files(files):

    # Process the files by converting uppercase to lowercase and vice versa

    for filepath, \_, \_ in files:

        with open(filepath, 'rb') as file:

            raw\_data = file.read()

            result = chardet.detect(raw\_data)

            encoding = result['encoding']

            text = raw\_data.decode(encoding)

            # Convert uppercase to lowercase and vice versa

            processed\_text = text.swapcase()

            processed\_filepath = filepath.replace('/raw/', '/processed/').replace('.txt', '\_processed.txt')

            with open(processed\_filepath, 'w', encoding=encoding) as processed\_file:

                processed\_file.write(processed\_text)

                log(f"Processed file '{processed\_filepath}' created with content: {processed\_text}")

    log("File processing completed successfully.")

def serialize\_processed\_data():

    processed\_dir = 'project\_root/data/processed/'

    output\_dir = 'project\_root/output/'

    data\_list = []

    for file\_name in os.listdir(processed\_dir):

        file\_path = os.path.join(processed\_dir, file\_name)

        with open(file\_path, 'rb') as file:

            raw\_data = file.read()

            result = chardet.detect(raw\_data)

            encoding = result['encoding']

            # Decode the text using the detected encoding

            text\_processed = raw\_data.decode(encoding)

            # Reverting the processed text to get the original text

            text = text\_processed.swapcase()

            file\_size = os.path.getsize(file\_path)

            last\_change\_time = datetime.fromtimestamp(os.path.getmtime(file\_path)).strftime('%Y-%m-%d %H:%M:%S')

            data = Data(file\_name, text, text\_processed, file\_size, last\_change\_time)

            data\_list.append(data\_to\_dict(data))

    output\_file\_path = os.path.join(output\_dir, 'processed\_data.json')

    with open(output\_file\_path, 'w', encoding='utf-8') as json\_file:

        json.dump(data\_list, json\_file, ensure\_ascii=False, indent=4)

    log(f"Data serialized and saved to {output\_file\_path}")

if \_\_name\_\_ == "\_\_main\_\_":

    create\_project\_structure()

    files = create\_example\_files()

    process\_files(files)

    serialize\_processed\_data()

    log("All steps completed successfully.")

import os

from datetime import datetime

import chardet

import json

import zipfile

class Data:

    def \_\_init\_\_(self, file\_name, text, text\_processed, file\_size, last\_change\_time):

        self.file\_name = file\_name

        self.text = text

        self.text\_processed = text\_processed

        self.file\_size = file\_size

        self.last\_change\_time = last\_change\_time

def data\_to\_dict(data):

    return {

        'file\_name': data.file\_name,

        'text': data.text,

        'text\_processed': data.text\_processed,

        'file\_size': data.file\_size,

        'last\_change\_time': data.last\_change\_time

    }

def create\_project\_structure():

    # Define the directory structure

    dirs = [

        "project\_root/data/raw",

        "project\_root/data/processed",

        "project\_root/logs",

        "project\_root/backups",

        "project\_root/output"

    ]

    # Create directories if they do not exist

    for dir in dirs:

        os.makedirs(dir, exist\_ok=True)

        log(f"Directory '{dir}' created or already exists.")

def log(message):

    # Get current date and time

    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    # Write log message to file

    with open('project\_root/logs/project\_log.txt', 'a', encoding='utf-8') as logfile:

        logfile.write(f"{timestamp} - {message}\n")

def create\_example\_files():

    # Create example files with different encodings

    files = [

        ('project\_root/data/raw/example1.txt', 'Hello, World!', 'utf-8'),

        ('project\_root/data/raw/example2.txt', 'Привет, мир!', 'utf-8'),

        ('project\_root/data/raw/example3.txt', '¡Hola mundo!', 'iso-8859-1'),

        ('project\_root/data/raw/example4.txt', 'Hola mundo!', 'ascii'),

        ('project\_root/data/raw/example5.txt', 'Hello, World', 'ascii'),

    ]

    for filepath, content, encoding in files:

        with open(filepath, 'w', encoding=encoding) as file:

            file.write(content)

            log(f"File '{filepath}' created with encoding '{encoding}' and content: {content}")

    log("Example files created successfully.")

    return files

def process\_files(files):

    # Process the files by converting uppercase to lowercase and vice versa

    for filepath, \_, \_ in files:

        with open(filepath, 'rb') as file:

            raw\_data = file.read()

            result = chardet.detect(raw\_data)

            encoding = result['encoding']

            text = raw\_data.decode(encoding)

            # Convert uppercase to lowercase and vice versa

            processed\_text = text.swapcase()

            processed\_filepath = filepath.replace('/raw/', '/processed/').replace('.txt', '\_processed.txt')

            with open(processed\_filepath, 'w', encoding=encoding) as processed\_file:

                processed\_file.write(processed\_text)

                log(f"Processed file '{processed\_filepath}' created with content: {processed\_text}")

    log("File processing completed successfully.")

def serialize\_processed\_data():

    processed\_dir = 'project\_root/data/processed/'

    output\_dir = 'project\_root/output/'

    data\_list = []

    for file\_name in os.listdir(processed\_dir):

        file\_path = os.path.join(processed\_dir, file\_name)

        with open(file\_path, 'rb') as file:

            raw\_data = file.read()

            result = chardet.detect(raw\_data)

            encoding = result['encoding']

            # Decode the text using the detected encoding

            text\_processed = raw\_data.decode(encoding)

            # Reverting the processed text to get the original text

            text = text\_processed.swapcase()

            file\_size = os.path.getsize(file\_path)

            last\_change\_time = datetime.fromtimestamp(os.path.getmtime(file\_path)).strftime('%Y-%m-%d %H:%M:%S')

            data = Data(file\_name, text, text\_processed, file\_size, last\_change\_time)

            data\_list.append(data\_to\_dict(data))

    output\_file\_path = os.path.join(output\_dir, 'processed\_data.json')

    with open(output\_file\_path, 'w', encoding='utf-8') as json\_file:

        json.dump(data\_list, json\_file, ensure\_ascii=False, indent=4)

    log(f"Data serialized and saved to {output\_file\_path}")

def create\_backup():

    backup\_dir = 'project\_root/backups/'

    data\_dir = 'project\_root/data/'

    date\_str = datetime.now().strftime('%Y%m%d')

    backup\_file = os.path.join(backup\_dir, f'backup\_{date\_str}.zip')

    with zipfile.ZipFile(backup\_file, 'w') as backup\_zip:

        for foldername, subfolders, filenames in os.walk(data\_dir):

            for filename in filenames:

                file\_path = os.path.join(foldername, filename)

                backup\_zip.write(file\_path, os.path.relpath(file\_path, data\_dir))

    log(f"Backup created at {backup\_file}")

def restore\_backup(backup\_file):

    backup\_dir = 'project\_root/backups/'

    data\_dir = 'project\_root/data/'

    with zipfile.ZipFile(backup\_file, 'r') as backup\_zip:

        backup\_zip.extractall(data\_dir)

    log(f"Backup restored from {backup\_file}")

if \_\_name\_\_ == "\_\_main\_\_":

    create\_project\_structure()

    # Create and log example files

    files = create\_example\_files()

    # Process the files

    process\_files(files)

    # Serialize the processed data

    serialize\_processed\_data()

    # Create a backup

    create\_backup()

    # Restore the latest backup (as an example)

    latest\_backup = os.path.join('project\_root/backups/', f'backup\_{datetime.now().strftime("%Y%m%d")}.zip')

    restore\_backup(latest\_backup)

    log("Completed successfully.")

####### General comments

"""

Чтобы восстановить данные из конкретного архива нужно

вызвать функцию restore\_backup() с полным путем к нему.

To restore the data from a specific file you need to call

`restore\_backup()` function with full path to the file.

"""

# restore\_backup('project\_root/backups/backup\_20240824.zip')

# final

import os

import json

import zipfile

from datetime import datetime

import chardet

from jsonschema import validate, ValidationError, SchemaError

# Класс Data для хранения информации о данных файлов

class Data:

    def \_\_init\_\_(self, file\_name, text, text\_processed, file\_size, last\_change\_time):

        self.file\_name = file\_name

        self.text = text

        self.text\_processed = text\_processed

        self.file\_size = file\_size

        self.last\_change\_time = last\_change\_time

# Класс FileInfo для хранения информации о файлах

class FileInfo:

    def \_\_init\_\_(self, file\_name, full\_path, file\_size, creation\_time, last\_change\_time):

        self.file\_name = file\_name

        self.full\_path = full\_path

        self.file\_size = file\_size

        self.creation\_time = creation\_time

        self.last\_change\_time = last\_change\_time

def data\_to\_dict(data):

    return {

        'file\_name': data.file\_name,

        'text': data.text,

        'text\_processed': data.text\_processed,

        'file\_size': data.file\_size,

        'last\_change\_time': data.last\_change\_time

    }

def file\_info\_to\_dict(file\_info):

    return {

        'file\_name': file\_info.file\_name,

        'full\_path': file\_info.full\_path,

        'file\_size': file\_info.file\_size,

        'creation\_time': file\_info.creation\_time,

        'last\_change\_time': file\_info.last\_change\_time

    }

# Функция для создания структуры директорий проекта

def create\_project\_structure():

    dirs = [

        "project\_root/data/raw",

        "project\_root/data/processed",

        "project\_root/logs",

        "project\_root/backups",

        "project\_root/output"

    ]

    for dir in dirs:

        os.makedirs(dir, exist\_ok=True)

# Функция для логирования действий

def log(message):

    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    with open('project\_root/logs/project\_log.txt', 'a', encoding='utf-8') as logfile:

        logfile.write(f"{timestamp} - {message}\n")

# Функция для создания примерных файлов с разными кодировками

def create\_example\_files():

    files = [

        ('project\_root/data/raw/example1.txt', 'Hello, World!', 'utf-8'),

        ('project\_root/data/raw/example2.txt', 'Привет, мир!', 'utf-8'),

        ('project\_root/data/raw/example3.txt', '¡Hola mundo!', 'iso-8859-1'),

        ('project\_root/data/raw/example4.txt', 'Hola mundo!', 'ascii'),

        ('project\_root/data/raw/example5.txt', 'Hello, World', 'ascii'),

    ]

    for filepath, content, encoding in files:

        with open(filepath, 'w', encoding=encoding) as file:

            file.write(content)

            log(f"File '{filepath}' created with encoding '{encoding}' and content: {content}")

    log("Example files created successfully.")

    return files

# Функция для обработки файлов: преобразование регистра и сохранение в другую директорию

def process\_files(files):

    for filepath, \_, \_ in files:

        with open(filepath, 'rb') as file:

            raw\_data = file.read()

            result = chardet.detect(raw\_data)

            encoding = result['encoding']

            text = raw\_data.decode(encoding)

            processed\_text = text.swapcase()

            processed\_filepath = filepath.replace('/raw/', '/processed/').replace('.txt', '\_processed.txt')

            with open(processed\_filepath, 'w', encoding=encoding) as processed\_file:

                processed\_file.write(processed\_text)

                log(f"Processed file '{processed\_filepath}' created with content: {processed\_text}")

    log("File processing completed successfully.")

# Функция для сериализации обработанных данных в JSON-файл

def serialize\_processed\_data():

    processed\_dir = 'project\_root/data/processed/'

    output\_dir = 'project\_root/output/'

    data\_list = []

    for file\_name in os.listdir(processed\_dir):

        file\_path = os.path.join(processed\_dir, file\_name)

        with open(file\_path, 'rb') as file:

            raw\_data = file.read()

            result = chardet.detect(raw\_data)

            encoding = result['encoding']

            text\_processed = raw\_data.decode(encoding)

            text = text\_processed.swapcase()

            file\_size = os.path.getsize(file\_path)

            last\_change\_time = datetime.fromtimestamp(os.path.getmtime(file\_path)).strftime('%Y-%m-%d %H:%M:%S')

            data = Data(file\_name, text, text\_processed, file\_size, last\_change\_time)

            data\_list.append(data\_to\_dict(data))

    output\_file\_path = os.path.join(output\_dir, 'processed\_data.json')

    with open(output\_file\_path, 'w', encoding='utf-8') as json\_file:

        json.dump(data\_list, json\_file, ensure\_ascii=False, indent=4)

    log(f"Data serialized and saved to {output\_file\_path}")

# Функция для создания резервной копии всех данных из директории data/

def create\_backup():

    backup\_dir = 'project\_root/backups/'

    data\_dir = 'project\_root/data/'

    date\_str = datetime.now().strftime('%Y%m%d')

    backup\_file = os.path.join(backup\_dir, f'backup\_{date\_str}.zip')

    with zipfile.ZipFile(backup\_file, 'w') as backup\_zip:

        for foldername, subfolders, filenames in os.walk(data\_dir):

            for filename in filenames:

                file\_path = os.path.join(foldername, filename)

                backup\_zip.write(file\_path, os.path.relpath(file\_path, data\_dir))

    log(f"Backup created at {backup\_file}")

# Функция для восстановления данных из резервной копии

def restore\_backup(backup\_file):

    data\_dir = 'project\_root/data/'

    with zipfile.ZipFile(backup\_file, 'r') as backup\_zip:

        backup\_zip.extractall(data\_dir)

    log(f"Backup restored from {backup\_file}")

# Функция для сбора информации о всех файлах в директории data/processed/ и сериализации их в JSON

def collect\_file\_info():

    processed\_dir = 'project\_root/data/processed/'

    file\_info\_list = []

    for file\_name in os.listdir(processed\_dir):

        file\_path = os.path.join(processed\_dir, file\_name)

        file\_size = os.path.getsize(file\_path)

        creation\_time = datetime.fromtimestamp(os.path.getctime(file\_path)).strftime('%Y-%m-%d %H:%M:%S')

        last\_change\_time = datetime.fromtimestamp(os.path.getmtime(file\_path)).strftime('%Y-%m-%d %H:%M:%S')

        file\_info = FileInfo(file\_name, file\_path, file\_size, creation\_time, last\_change\_time)

        file\_info\_list.append(file\_info\_to\_dict(file\_info))

    output\_file\_path = 'project\_root/output/file\_info.json'

    with open(output\_file\_path, 'w', encoding='utf-8') as json\_file:

        json.dump(file\_info\_list, json\_file, ensure\_ascii=False, indent=4)

    log(f"File info serialized and saved to {output\_file\_path}")

    return output\_file\_path

# Функция для десериализации данных из JSON

def deserialize\_file\_info(json\_file\_path):

    with open(json\_file\_path, 'r', encoding='utf-8') as json\_file:

        file\_info\_list = json.load(json\_file)

    deserialized\_info = [FileInfo(\*\*file\_info) for file\_info in file\_info\_list]

    log("File info deserialized successfully")

    return deserialized\_info

# Функция для получения схемы JSON для валидации

def get\_file\_info\_schema():

    return {

        "type": "array",

        "items": {

            "type": "object",

            "properties": {

                "file\_name": {"type": "string"},

                "full\_path": {"type": "string"},

                "file\_size": {"type": "integer"},

                "creation\_time": {"type": "string", "format": "date-time"},

                "last\_change\_time": {"type": "string", "format": "date-time"}

            },

            "required": ["file\_name", "full\_path", "file\_size", "creation\_time", "last\_change\_time"]

        }

    }

# Функция для валидации JSON-файла с использованием схемы

def validate\_json(json\_file\_path):

    schema = get\_file\_info\_schema()

    with open(json\_file\_path, 'r', encoding='utf-8') as json\_file:

        data = json.load(json\_file)

    try:

        validate(instance=data, schema=schema)

        log("JSON file is valid according to the schema")

        return True

    except ValidationError as e:

        log(f"Validation error: {e.message}")

    except SchemaError as e:

        log(f"Schema error: {e.message}")

    return False

# Функция для генерации итогового отчета

def generate\_report(start\_time, json\_validation\_result):

    end\_time = datetime.now()

    elapsed\_time = end\_time - start\_time

    report = {

        "start\_time": start\_time.strftime('%Y-%m-%d %H:%M:%S'),

        "end\_time": end\_time.strftime('%Y-%m-%d %H:%M:%S'),

        "elapsed\_time": str(elapsed\_time),

        "tasks": [

            {

                "task": "Create project structure",

                "status": "Completed",

                "details": "Directories for the project were created."

            },

            {

                "task": "Create and process files",

                "status": "Completed",

                "details": "Files were created and processed, and data was serialized."

            },

            {

                "task": "Validate JSON",

                "status": "Completed" if json\_validation\_result else "Failed",

                "details": "JSON validation result."

            }

        ],

        "conclusions": "All tasks were completed. Consider adding more error handling and validation checks."

    }

    report\_path = 'project\_root/output/report.json'

    with open(report\_path, 'w', encoding='utf-8') as json\_file:

        json.dump(report, json\_file, ensure\_ascii=False, indent=4)

    txt\_report\_path = 'project\_root/output/report.txt'

    with open(txt\_report\_path, 'w', encoding='utf-8') as txt\_file:

        for key, value in report.items():

            txt\_file.write(f"{key}: {value}\n\n")

    log(f"Report generated and saved to {report\_path} and {txt\_report\_path}")

# Основной блок выполнения всех задач

if \_\_name\_\_ == "\_\_main\_\_":

    start\_time = datetime.now()

    # Сначала создаем структуру директорий

    create\_project\_structure()

    # Теперь можем начать логирование

    log("Project structure created")

    # Создание и обработка файлов

    files = create\_example\_files()

    process\_files(files)

    # Сериализация данных

    serialize\_processed\_data()

    # Сбор информации о файлах и её сериализация

    json\_file\_path = collect\_file\_info()

    # Валидация JSON

    json\_validation\_result = validate\_json(json\_file\_path)

    # Генерация отчета

    generate\_report(start\_time, json\_validation\_result)

    # Создание резервной копии

    create\_backup()

    # Пример восстановления данных из последней резервной копии

    latest\_backup = os.path.join('project\_root/backups/', f'backup\_{datetime.now().strftime("%Y%m%d")}.zip')

    restore\_backup(latest\_backup)

    log("All steps completed successfully.")

#report

start\_time: 2024-08-24 22:25:10

end\_time: 2024-08-24 22:25:10

elapsed\_time: 0:00:00.051552

tasks: [{'task': 'Create project structure', 'status': 'Completed', 'details': 'Directories for the project were created.'}, {'task': 'Create and process files', 'status': 'Completed', 'details': 'Files were created and processed, and data was serialized.'}, {'task': 'Validate JSON', 'status': 'Completed', 'details': 'JSON validation result.'}]

conclusions: All tasks were completed.

# report.json

{

    "start\_time": "2024-08-24 22:30:06",

    "end\_time": "2024-08-24 22:30:06",

    "elapsed\_time": "0:00:00.050644",

    "tasks": [

        {

            "task": "Create project structure",

            "status": "Completed",

            "details": "Directories for the project were created."

        },

        {

            "task": "Create and process files",

            "status": "Completed",

            "details": "Files were created and processed, and data was serialized."

        },

        {

            "task": "Validate JSON",

            "status": "Completed",

            "details": "JSON validation result."

        }

    ],

    "conclusions": "All tasks were completed. "

}