

File Processing Tool

Group Members

Ahmet Sevinç (2001833)

Project Description

The File Processing Tool is a desktop application designed to efficiently handle file compression and decompression operations. It provides a user-friendly graphical interface that allows users to compress individual files or entire folders, as well as decompress previously compressed files. The tool supports various file formats and implements robust error handling to ensure reliable operation.

Implemented Features

- Single file compression using gzip algorithm
- Batch compression of folders
- File decompression capability
- Support for multiple file formats (.txt, .csv, .json, .xml, .log)
- Modern graphical user interface using PyQt5
- Comprehensive error handling and user feedback
- Cross-platform compatibility

Technologies and Libraries

Technology	Purpose
Python 3.8+	Core programming language
PyQt5	Graphical user interface framework
gzip	File compression algorithm
pytest	Testing framework
pathlib	File system operations
typing	Type hints and annotations

Challenges and Solutions

Cross-platform Compatibility

Challenge: Ensuring consistent file path handling across different operating systems. Solution: Implemented pathlib for platform-independent path manipulation.

Error Handling

Challenge: Graceful handling of various error scenarios (invalid files, permissions, etc.). Solution: Implemented comprehensive try-except blocks and user-friendly error messages.

User Interface Design

Challenge: Creating an intuitive and responsive interface. Solution: Utilized PyQt5's layout management system and implemented progress feedback.

Code Implementation

Core Processing Class (FileProcessor)

```
import os
import gzip
import shutil
from typing import Union, List, Set
from pathlib import Path

class FileProcessor:
    """Core class for handling file operations."""

    def __init__(self):
        self.allowed_formats: Set[str] = {'.txt', '.csv', '.json', '.xml', '.log'}

    def pack_single(self, source_path: Union[str, Path]) -> str:
        """Pack a single file using gzip algorithm."""
        input_path = Path(source_path)
        if not input_path.exists():
            raise FileNotFoundError(f"Unable to locate file: {input_path}")

        if input_path.suffix not in self.allowed_formats:
            raise ValueError(f"Format not supported: {input_path.suffix}")

        result_path = str(input_path) + '.gz'

        with open(input_path, 'rb') as source, gzip.open(result_path, 'wb') as target:
            shutil.copyfileobj(source, target)

        return result_path

    def unpack_single(self, source_path: Union[str, Path]) -> str:
        """Unpack a gzip packed file."""
        input_path = Path(source_path)
        if not input_path.exists():
            raise FileNotFoundError(f"Unable to locate file: {input_path}")

        if not str(input_path).endswith('.gz'):
            raise ValueError("File must be in gzip format (.gz)")

        result_path = str(input_path)[: -3]

        with gzip.open(input_path, 'rb') as source, open(result_path, 'wb') as target:
            shutil.copyfileobj(source, target)

        return result_path

    def pack_folder(self, source_path: Union[str, Path]) -> List[str]:
        """Pack all supported files in a folder."""
        input_path = Path(source_path)
        if not input_path.is_dir():
            raise NotADirectoryError(f"Not a valid folder: {input_path}")

        processed_files = []
        for file_path in input_path.rglob('*'):
            if file_path.is_file() and file_path.suffix in self.allowed_formats:
                try:
                    packed_file = self.pack_single(file_path)
                    processed_files.append(packed_file)
                except Exception as e:
                    print(f"Error processing {file_path}: {str(e)}")
```

```
return processed_files
```

Graphical User Interface (ProcessorInterface)

```
import sys
from pathlib import Path
from PyQt5.QtWidgets import (QApplication, QMainWindow, QPushButton, QVBoxLayout,
                             QHBoxLayout, QFileDialog, QLabel, QWidget, QProgressBar)
from PyQt5.QtCore import Qt
from compression.compressor import FileProcessor

class ProcessorInterface(QMainWindow):
    def __init__(self):
        super().__init__()
        self.processor = FileProcessor()
        self.setup_interface()

    def setup_interface(self):
        """Configure the user interface."""
        self.setWindowTitle('File Processing Tool')
        self.setGeometry(100, 100, 600, 400)

        # Setup main container
        main_container = QWidget()
        self.setCentralWidget(main_container)
        main_layout = QVBoxLayout(main_container)

        # Setup display elements
        self.info_display = QLabel('Select files or folders for processing')
        self.info_display.setAlignment(Qt.AlignmentFlag.AlignCenter)

        # Setup action buttons
        pack_btn = QPushButton('Pack File')
        unpack_btn = QPushButton('Unpack File')
        folder_btn = QPushButton('Process Folder')

        # Setup progress indicator
        self.progress_indicator = QProgressBar()
        self.progress_indicator.setVisible(False)

        # Arrange elements
        main_layout.addWidget(self.info_display)
        main_layout.addWidget(pack_btn)
        main_layout.addWidget(unpack_btn)
        main_layout.addWidget(folder_btn)
        main_layout.addWidget(self.progress_indicator)

        # Connect actions
        pack_btn.clicked.connect(self.handle_pack)
        unpack_btn.clicked.connect(self.handle_unpack)
        folder_btn.clicked.connect(self.handle_folder)

    def handle_pack(self):
        """Process pack file request."""
        source_path, _ = QFileDialog.getOpenFileName(
            self,
            "Choose File to Pack",
            "",
            f"Supported Formats ({' '.join('*' + ext for ext in self.processor.allowed_formats)})"
        )

        if source_path:
            try:
                result_path = self.processor.pack_single(source_path)
                self.info_display.setText(f'Successfully packed: {result_path}')
            except Exception as e:
                self.info_display.setText(f'Error: {str(e)}')

    def handle_unpack(self):
```

```

        """Process unpack file request."""
        source_path, _ = QFileDialog.getOpenFileName(
            self,
            "Choose File to Unpack",
            "",
            "Packed Files (*.gz)"
        )

        if source_path:
            try:
                result_path = self.processor.unpack_single(source_path)
                self.info_display.setText(f'Successfully unpacked: {result_path}')
            except Exception as e:
                self.info_display.setText(f'Error: {str(e)}')

    def handle_folder(self):
        """Process folder request."""
        source_path = QFileDialog.getExistingDirectory(
            self,
            "Choose Folder to Process"
        )

        if source_path:
            try:
                processed_files = self.processor.pack_folder(source_path)
                self.info_display.setText(
                    f'Successfully processed {len(processed_files)} files in folder'
                )
            except Exception as e:
                self.info_display.setText(f'Error: {str(e)}')

```

Main Application Entry Point

```

import sys
from PyQt5.QtWidgets import QApplication
from compression.gui import ProcessorInterface

def main():
    """Application entry point."""
    app = QApplication(sys.argv)
    window = ProcessorInterface()
    window.show()
    sys.exit(app.exec_())

if __name__ == '__main__':
    main()

```

References

- Python Documentation - <https://docs.python.org/>
- PyQt5 Documentation - <https://www.riverbankcomputing.com/static/Docs/PyQt5/>
- gzip Module Documentation - <https://docs.python.org/3/library/gzip.html>