File Manager App, Text Processing và Data Persistence

Thời gian: 3 tiếng (14:00-17:00)

Mục tiêu:

- Xây dựng ứng dụng File Manager hoàn chỉnh
- Xử lý text files nâng cao
- Hiểu và áp dụng Data Persistence
- Tối ưu hóa performance khi làm việc với files lớn

Kiến thức cần có:

- File handling cơ bản (đọc/ghi files)
- CSV operations
- Error handling
- Functions và classes

1. Xây dựng File Manager Application

1.1 Core File Operations

In [ ]:
```
import os
import shutil
import time
from datetime import datetime

class FileManager:
    """File Manager với các chức năng cơ bản"""

    def __init__(self, working_directory="."):
        self.current_dir = os.path.abspath(working_directory)
        self.history = []
        self.bookmarks = {}
        self.load_bookmarks()

    def get_file_info(self, file_path):
        """Lấy thông tin chi tiết của file/folder"""
        try:
            stat_info = os.stat(file_path)

            info = {
                'name': os.path.basename(file_path),
                'path': file_path,
                'size': stat_info.st_size,
                'modified': datetime.fromtimestamp(stat_info.st_mtime),
```

```
            'created': datetime.fromtimestamp(stat_info.st_ctime),
            'is_dir': os.path.isdir(file_path),
            'is_file': os.path.isfile(file_path),
            'extension': os.path.splitext(file_path)[1].lower(),
            'permissions': oct(stat_info.st_mode)[-3:]
        }

        return info
    except Exception as e:
        print(f"❌ Lỗi lấy thông tin file: {e}")
        return None

def list_files(self, directory=None, show_hidden=False, sort_by="name"):
    """Liệt kê files trong thư mục với options"""
    if directory is None:
        directory = self.current_dir

    try:
        items = []

        for item_name in os.listdir(directory):
            if not show_hidden and item_name.startswith('.'):
                continue

            item_path = os.path.join(directory, item_name)
            info = self.get_file_info(item_path)
            if info:
                items.append(info)

        # Sắp xếp
        sort_key_map = {
            'name': lambda x: x['name'].lower(),
            'size': lambda x: x['size'],
            'modified': lambda x: x['modified'],
            'type': lambda x: (x['is_file'], x['extension'])
        }

        if sort_by in sort_key_map:
            items.sort(key=sort_key_map[sort_by])

        return items

    except PermissionError:
        print(f"❌ Không có quyền truy cập: {directory}")
        return []
    except Exception as e:
        print(f"❌ Lỗi: {e}")
        return []

def format_file_size(self, size_bytes):
    """Format file size thành human readable"""
    for unit in ['B', 'KB', 'MB', 'GB', 'TB']:
        if size_bytes < 1024.0:
            return f"{size_bytes:.1f} {unit}"
        size_bytes /= 1024.0
    return f"{size_bytes:.1f} PB"
```

```
    def display_files(self, directory=None, detailed=True):
        """Hiển thị files với format đẹp"""
        if directory is None:
            directory = self.current_dir

        print(f"\n📁 Thư mục: {directory}")
        print("="*80)

        items = self.list_files(directory)

        if not items:
            print("(Thư mục trống)")
            return

        # Tách folders và files
        folders = [item for item in items if item['is_dir']]
        files = [item for item in items if item['is_file']]

        # Hiển thị folders trước
        if folders:
            print("📁 Thư mục:")
            for folder in folders:
                modified = folder['modified'].strftime("%Y-%m-%d %H:%M")
                if detailed:
                    print(f"  📁 {folder['name']:<30} {modified}")
                else:
                    print(f"  📁 {folder['name']}")

        # Hiển thị files
        if files:
            print("\n📄 Files:")
            for file in files:
                size = self.format_file_size(file['size'])
                modified = file['modified'].strftime("%Y-%m-%d %H:%M")
                ext = file['extension'] or 'no ext'

                if detailed:
                    print(f"  📄 {file['name']:<30} {size:>10} {ext:>8} {modified}")
                else:
                    print(f"  📄 {file['name']}")

        print(f"\nTổng: {len(folders)} thư mục, {len(files)} files")

# Test File Manager
fm = FileManager()
fm.display_files()
```

1.2 Advanced File Operations

In [ ]:

```
class AdvancedFileManager(FileManager):
    """File Manager với các chức năng nâng cao"""

    def search_files(self, pattern, search_in="name", directory=None):
        """Tìm kiếm files theo pattern"""
```

```
        if directory is None:
            directory = self.current_dir

        results = []
        pattern = pattern.lower()

        try:
            for root, dirs, files in os.walk(directory):
                # Tìm trong tên file
                if search_in in ["name", "all"]:
                    for file in files:
                        if pattern in file.lower():
                            file_path = os.path.join(root, file)
                            info = self.get_file_info(file_path)
                            if info:
                                results.append(info)

                # Tìm trong nội dung file (chỉ text files)
                if search_in in ["content", "all"]:
                    for file in files:
                        file_path = os.path.join(root, file)
                        if self.is_text_file(file_path):
                            if self.search_in_file_content(file_path, pattern):
                                info = self.get_file_info(file_path)
                                if info and info not in results:
                                    results.append(info)

        except Exception as e:
            print(f"❌ Lỗi tìm kiếm: {e}")

        return results

    def is_text_file(self, file_path):
        """Kiểm tra file có phải text file không"""
        text_extensions = {'.txt', '.py', '.js', '.html', '.css', '.md', '.json',
                           '.csv', '.xml', '.log', '.ini', '.cfg', '.conf'}

        _, ext = os.path.splitext(file_path)
        return ext.lower() in text_extensions

    def search_in_file_content(self, file_path, pattern):
        """Tìm kiếm pattern trong nội dung file"""
        try:
            with open(file_path, 'r', encoding='utf-8', errors='ignore') as file:
                content = file.read().lower()
                return pattern in content
        except:
            return False

    def copy_file(self, source, destination):
        """Copy file với progress tracking"""
        try:
            if os.path.isdir(source):
                shutil.copytree(source, destination)
                print(f"✅ Đã copy folder: {source} -> {destination}")
            else:
```

```
                shutil.copy2(source, destination)
                print(f"✅ Đã copy file: {source} -> {destination}")
            return True
        except Exception as e:
            print(f"❌ Lỗi copy: {e}")
            return False

    def move_file(self, source, destination):
        """Move file/folder"""
        try:
            shutil.move(source, destination)
            print(f"✅ Đã move: {source} -> {destination}")
            return True
        except Exception as e:
            print(f"❌ Lỗi move: {e}")

        return False

    def delete_file(self, file_path, confirm=True):
        """Xóa file/folder với confirmation"""
        if confirm:
            response = input(f"Bạn có chắc muốn xóa '{file_path}'? (y/N): ")
            if response.lower() != 'y':
                print("Hủy thao tác xóa")
                return False

        try:
            if os.path.isdir(file_path):
                shutil.rmtree(file_path)
                print(f"✅ Đã xóa thư mục: {file_path}")
            else:
                os.remove(file_path)
                print(f"✅ Đã xóa file: {file_path}")
            return True
        except Exception as e:
            print(f"❌ Lỗi xóa: {e}")
            return False

    def create_folder(self, folder_name, parent_dir=None):
        """Tạo thư mục mới"""
        if parent_dir is None:
            parent_dir = self.current_dir

        folder_path = os.path.join(parent_dir, folder_name)

        try:
            os.makedirs(folder_path, exist_ok=True)
            print(f"✅ Đã tạo thư mục: {folder_path}")
            return True
        except Exception as e:
            print(f"❌ Lỗi tạo thư mục: {e}")
            return False

    def get_directory_size(self, directory):
        """Tính tổng kích thước thư mục"""
        total_size = 0
```

```
        file_count = 0

        try:
            for root, dirs, files in os.walk(directory):
                for file in files:
                    file_path = os.path.join(root, file)
                    try:
                        size = os.path.getsize(file_path)
                        total_size += size
                        file_count += 1
                    except (OSError, FileNotFoundError):
                        continue
        except Exception as e:
            print(f"❌ Lỗi tính kích thước: {e}")

        return total_size, file_count

# Test Advanced File Manager
print("=== ADVANCED FILE MANAGER ===")
afm = AdvancedFileManager()

# Test search
print("\n🔍 Tìm kiếm files có 'test' trong tên:")
results = afm.search_files("test", "name")
for result in results[:3]:  # Chỉ hiển thị 3 kết quả đầu
    print(f"   📄 {result['name']} - {afm.format_file_size(result['size'])}")

# Test create folder
afm.create_folder("test_folder")

# Test directory size
size, count = afm.get_directory_size(".")
print(f"\n📊 Thư mục hiện tại: {afm.format_file_size(size)}, {count} files")
```

2. Text Processing - Xử lý văn bản nâng cao

2.1 Text Analysis Tools

In [ ]:

```
import re
from collections import Counter
import string

class TextProcessor:
    """Class xử lý và phân tích văn bản"""

    def __init__(self):
        self.stop_words = {
            'vi': {'và', 'hoặc', 'với', 'của', 'trong', 'trên', 'dưới', 'này', 'đó'
                   'một', 'các', 'những', 'cho', 'để', 'từ', 'tại', 'về', 'theo'},
            'en': {'the', 'and', 'or', 'but', 'in', 'on', 'at', 'to', 'for',
                   'of', 'with', 'by', 'from', 'this', 'that', 'a', 'an'}
        }

    def read_text_file(self, file_path):
```

```
        """Đọc file text với encoding detection"""
        encodings = ['utf-8', 'utf-8-sig', 'latin-1', 'cp1252']

        for encoding in encodings:
            try:
                with open(file_path, 'r', encoding=encoding) as file:
                    return file.read()
            except UnicodeDecodeError:
                continue

        print(f"❌ Không thể đọc file {file_path}")
        return None

    def basic_stats(self, text):
        """Thống kê cơ bản của văn bản"""
        if not text:
            return {}

        # Đếm ký tự, từ, câu, đoạn
        char_count = len(text)
        char_count_no_spaces = len(text.replace(' ', '').replace('\n', '').replace(
        word_count = len(text.split())
        sentence_count = len(re.findall(r'[.!?]+', text))
        paragraph_count = len([p for p in text.split('\n\n') if p.strip()])
        line_count = len(text.split('\n'))

        # Tính trung bình
        avg_words_per_sentence = word_count / sentence_count if sentence_count > 0
        avg_chars_per_word = char_count_no_spaces / word_count if word_count > 0 el

        return {
            'characters': char_count,
            'characters_no_spaces': char_count_no_spaces,
            'words': word_count,
            'sentences': sentence_count,
            'paragraphs': paragraph_count,
            'lines': line_count,
            'avg_words_per_sentence': avg_words_per_sentence,
            'avg_chars_per_word': avg_chars_per_word
        }

    def word_frequency(self, text, language='auto', top_n=10):
        """Phân tích tần suất từ"""
        if not text:
            return []

        # Làm sạch text
        text = text.lower()
        # Loại bỏ dấu câu
        text = text.translate(str.maketrans('', '', string.punctuation))

        words = text.split()

        # Loại bỏ stop words
        if language == 'auto':
            # Simple detection based on common words
```

```
            language = 'vi' if any(word in words for word in ['và', 'của', 'trong']

        if language in self.stop_words:
            words = [word for word in words if word not in self.stop_words[language

        # Đếm tần suất
        word_freq = Counter(words)

        return word_freq.most_common(top_n)

    def find_long_words(self, text, min_length=8):
        """Tìm từ dài"""
        words = re.findall(r'\b\w+\b', text.lower())
        long_words = [word for word in set(words) if len(word) >= min_length]
        return sorted(long_words, key=len, reverse=True)

    def extract_emails(self, text):
        """Trích xuất email addresses"""
        email_pattern = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'
        return re.findall(email_pattern, text)

    def extract_urls(self, text):
        """Trích xuất URLs"""
        url_pattern = r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\(\),]|(?:%[0-9a-
        return re.findall(url_pattern, text)

    def extract_phone_numbers(self, text):
        """Trích xuất số điện thoại"""
        phone_patterns = [
            r'\b0\d{9}\b',  # 0xxxxxxxxx
            r'\b\+84\d{9}\b',  # +84xxxxxxxxx
            r'\b84\d{9}\b',  # 84xxxxxxxxx
            r'\b0\d{2}[\s.-]?\d{3}[\s.-]?\d{4}\b',  # 0xx xxx xxxx
        ]

        phone_numbers = []
        for pattern in phone_patterns:
            phone_numbers.extend(re.findall(pattern, text))

        return list(set(phone_numbers))  # Remove duplicates

    def reading_time_estimate(self, text, words_per_minute=200):
        """Ước tính thời gian đọc"""
        word_count = len(text.split())
        minutes = word_count / words_per_minute

        if minutes < 1:
            return f"{int(minutes * 60)} giây"
        elif minutes < 60:
            return f"{minutes:.1f} phút"
        else:
            hours = int(minutes // 60)
            remaining_minutes = int(minutes % 60)
            return f"{hours} giờ {remaining_minutes} phút"

    def analyze_file(self, file_path):
```

```
        """Phân tích hoàn chỉnh một file text"""
        print(f"\n📊 PHÂN TÍCH FILE: {file_path}")
        print("="*60)

        text = self.read_text_file(file_path)
        if not text:
            return

        # Thống kê cơ bản
        stats = self.basic_stats(text)
        print("📈 Thống kê cơ bản:")
        print(f"  • Ký tự: {stats['characters']:,}")
        print(f"  • Từ: {stats['words']:,}")
        print(f"  • Câu: {stats['sentences']:,}")
        print(f"  • Đoạn văn: {stats['paragraphs']:,}")
        print(f"  • Dòng: {stats['lines']:,}")
        print(f"  • TB từ/câu: {stats['avg_words_per_sentence']:.1f}")
        print(f"  • TB ký tự/từ: {stats['avg_chars_per_word']:.1f}")

        # Thời gian đọc
        reading_time = self.reading_time_estimate(text)
        print(f"  • Thời gian đọc: ~{reading_time}")

        # Từ phổ biến
        word_freq = self.word_frequency(text, top_n=5)
        if word_freq:
            print("\n🔤 Từ phổ biến:")
            for word, count in word_freq:
                print(f"  • '{word}': {count} lần")

        # Từ dài
        long_words = self.find_long_words(text)[:5]
        if long_words:
            print("\n📏 Từ dài:")
            for word in long_words:
                print(f"  • {word} ({len(word)} ký tự)")

        # Emails và URLs
        emails = self.extract_emails(text)
        if emails:
            print(f"\n📧 Email tìm thấy: {len(emails)}")
            for email in emails[:3]:  # Chỉ hiển thị 3 cái đầu
                print(f"  • {email}")

        urls = self.extract_urls(text)
        if urls:
            print(f"\n🔗 URL tìm thấy: {len(urls)}")
            for url in urls[:3]:
                print(f"  • {url}")

        phones = self.extract_phone_numbers(text)
        if phones:
            print(f"\n📞 Số điện thoại: {len(phones)}")
            for phone in phones:
                print(f"  • {phone}")
```

```
# Test Text Processor
tp = TextProcessor()

# Tạo sample text để test
sample_text = """
Xin chào! Đây là một văn bản mẫu để test Text Processor.
Chúng ta sẽ phân tích văn bản này và xem các thống kê thú vị.

Email liên hệ: test@example.com hoặc admin@website.vn
Số điện thoại: 0987654321, +84123456789
Website: https://www.example.com

Văn bản này có nhiều từ lặp lại để test chức năng đếm tần suất.
Chức năng phân tích văn bản rất hữu ích cho việc xử lý dữ liệu.
"""

with open("sample_text.txt", "w", encoding="utf-8") as f:
    f.write(sample_text)

# Analyze
tp.analyze_file("sample_text.txt")
```

3. Data Persistence - Lưu trữ dữ liệu bền vững

3.1 Simple Database với Files

In [ ]:

```
import json
import csv
import pickle
from datetime import datetime
import os

class SimpleFileDB:
    """Simple database sử dụng files để persistent storage"""

    def __init__(self, db_name="simple_db"):
        self.db_name = db_name
        self.db_folder = f"{db_name}_data"
        self.tables = {}
        self.init_database()

    def init_database(self):
        """Khởi tạo database folder và metadata"""
        if not os.path.exists(self.db_folder):
            os.makedirs(self.db_folder)
            print(f"✅ Đã tạo database: {self.db_folder}")

        self.metadata_file = os.path.join(self.db_folder, "metadata.json")
        self.load_metadata()

    def load_metadata(self):
        """Load metadata của database"""
        try:
            if os.path.exists(self.metadata_file):
```

```
                with open(self.metadata_file, 'r', encoding='utf-8') as f:
                    metadata = json.load(f)
                    self.tables = metadata.get('tables', {})
            else:
                self.tables = {}
        except Exception as e:
            print(f"❌ Lỗi load metadata: {e}")
            self.tables = {}

    def save_metadata(self):
        """Lưu metadata"""
        try:
            metadata = {
                'db_name': self.db_name,
                'created': datetime.now().isoformat(),
                'tables': self.tables
            }

            with open(self.metadata_file, 'w', encoding='utf-8') as f:
                json.dump(metadata, f, indent=2, ensure_ascii=False)
        except Exception as e:
            print(f"❌ Lỗi save metadata: {e}")

    def create_table(self, table_name, columns, primary_key=None):
        """Tạo table mới"""
        if table_name in self.tables:
            print(f"⚠️ Table '{table_name}' đã tồn tại")
            return False

        table_info = {
            'columns': columns,
            'primary_key': primary_key,
            'created': datetime.now().isoformat(),
            'record_count': 0
        }

        self.tables[table_name] = table_info

        # Tạo CSV file cho table
        table_file = os.path.join(self.db_folder, f"{table_name}.csv")
        try:
            with open(table_file, 'w', newline='', encoding='utf-8') as f:
                writer = csv.writer(f)
                writer.writerow(columns)  # Header

            self.save_metadata()
            print(f"✅ Đã tạo table '{table_name}' với {len(columns)} columns")
            return True

        except Exception as e:
            print(f"❌ Lỗi tạo table: {e}")
            return False

    def insert(self, table_name, data):
        """Insert data vào table"""
        if table_name not in self.tables:
```

```
            print(f"❌ Table '{table_name}' không tồn tại")
            return False

        table_info = self.tables[table_name]
        columns = table_info['columns']

        # Validate data
        if isinstance(data, dict):
            # Convert dict to list theo thứ tự columns
            row_data = [data.get(col, '') for col in columns]
        elif isinstance(data, list):
            row_data = data
        else:
            print("❌ Data phải là dict hoặc list")
            return False

        if len(row_data) != len(columns):
            print(f"❌ Data phải có {len(columns)} values")
            return False

        # Append to CSV file
        table_file = os.path.join(self.db_folder, f"{table_name}.csv")
        try:
            with open(table_file, 'a', newline='', encoding='utf-8') as f:
                writer = csv.writer(f)
                writer.writerow(row_data)

            # Update record count
            self.tables[table_name]['record_count'] += 1
            self.save_metadata()

            print(f"✅ Đã insert 1 record vào '{table_name}'")
            return True

        except Exception as e:
            print(f"❌ Lỗi insert: {e}")
            return False

    def select_all(self, table_name):
        """Select tất cả records từ table"""
        if table_name not in self.tables:
            print(f"❌ Table '{table_name}' không tồn tại")
            return []

        table_file = os.path.join(self.db_folder, f"{table_name}.csv")

        try:
            with open(table_file, 'r', encoding='utf-8') as f:
                reader = csv.DictReader(f)
                records = list(reader)

            return records

        except Exception as e:
            print(f"❌ Lỗi select: {e}")
            return []
```

```
    def select_where(self, table_name, condition_func):
        """Select records với condition"""
        all_records = self.select_all(table_name)
        return [record for record in all_records if condition_func(record)]

    def update_where(self, table_name, condition_func, updates):
        """Update records theo condition"""
        all_records = self.select_all(table_name)
        updated_count = 0

        # Update records
        for record in all_records:
            if condition_func(record):
                record.update(updates)
                updated_count += 1

        if updated_count > 0:
            # Ghi lại toàn bộ table
            self._rewrite_table(table_name, all_records)
            print(f"✅ Đã update {updated_count} records")
        else:
            print("⚠️ Không có records nào match condition")

        return updated_count

    def delete_where(self, table_name, condition_func):
        """Delete records theo condition"""
        all_records = self.select_all(table_name)
        filtered_records = [record for record in all_records if not condition_func(

        deleted_count = len(all_records) - len(filtered_records)

        if deleted_count > 0:
            self._rewrite_table(table_name, filtered_records)
            self.tables[table_name]['record_count'] = len(filtered_records)
            self.save_metadata()
            print(f"✅ Đã delete {deleted_count} records")
        else:
            print("⚠️ Không có records nào match condition")

        return deleted_count

    def _rewrite_table(self, table_name, records):
        """Ghi lại toàn bộ table với records mới"""
        table_file = os.path.join(self.db_folder, f"{table_name}.csv")
        columns = self.tables[table_name]['columns']

        try:
            with open(table_file, 'w', newline='', encoding='utf-8') as f:
                writer = csv.DictWriter(f, fieldnames=columns)
                writer.writeheader()
                writer.writerows(records)
        except Exception as e:
            print(f"❌ Lỗi rewrite table: {e}")
```

```
    def backup_database(self, backup_name=None):
        """Backup toàn bộ database"""
        if backup_name is None:
            timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
            backup_name = f"{self.db_name}_backup_{timestamp}"

        try:
            import shutil
            shutil.copytree(self.db_folder, backup_name)
            print(f"✅ Đã backup database: {backup_name}")
            return backup_name
        except Exception as e:
            print(f"❌ Lỗi backup: {e}")
            return None

    def show_tables(self):
        """Hiển thị danh sách tables"""
        print(f"\n📊 DATABASE: {self.db_name}")
        print("="*50)

        if not self.tables:
            print("(Chưa có tables nào)")
            return

        for table_name, info in self.tables.items():
            print(f"📋 {table_name}:")
            print(f"   • Columns: {', '.join(info['columns'])}")
            print(f"   • Records: {info['record_count']}")
            if info.get('primary_key'):
                print(f"   • Primary Key: {info['primary_key']}")
            print()

# Test Simple File DB
print("=== SIMPLE FILE DATABASE ===")
db = SimpleFileDB("test_db")

# Tạo table students
db.create_table("students", ["id", "name", "age", "grade"], primary_key="id")

# Insert data
students_data = [
    {"id": "1", "name": "Nguyen Van A", "age": "20", "grade": "8.5"},
    {"id": "2", "name": "Tran Thi B", "age": "19", "grade": "9.0"},
    {"id": "3", "name": "Le Van C", "age": "21", "grade": "7.5"}
]

for student in students_data:
    db.insert("students", student)

# Show tables
db.show_tables()

# Select all
print("\n📋 Tất cả students:")
all_students = db.select_all("students")
for student in all_students:
```

```
    print(f"  {student}")

# Select with condition
print("\n🔍 Students có điểm >= 8.0:")
good_students = db.select_where("students", lambda x: float(x['grade']) >= 8.0)
for student in good_students:
    print(f"  {student['name']}: {student['grade']}")
```

3.2 Configuration Management

In [ ]:

```
import json
import configparser
from datetime import datetime

class ConfigManager:
    """Quản lý configuration cho applications"""

    def __init__(self, app_name="myapp"):
        self.app_name = app_name
        self.config_folder = "config"
        self.config_file = os.path.join(self.config_folder, f"{app_name}.json")
        self.ini_file = os.path.join(self.config_folder, f"{app_name}.ini")
        self.default_config = {}
        self.current_config = {}
        self.init_config_folder()

    def init_config_folder(self):
        """Tạo config folder nếu chưa có"""
        if not os.path.exists(self.config_folder):
            os.makedirs(self.config_folder)
            print(f"📁 Đã tạo config folder: {self.config_folder}")

    def set_defaults(self, defaults):
        """Set default configuration values"""
        self.default_config = defaults
        print(f"✅ Đã set {len(defaults)} default settings")

    def load_config(self, format_type="json"):
        """Load configuration từ file"""
        if format_type == "json":
            return self._load_json_config()
        elif format_type == "ini":
            return self._load_ini_config()
        else:
            print(f"❌ Format không support: {format_type}")
            return False

    def _load_json_config(self):
        """Load JSON config"""
        try:
            if os.path.exists(self.config_file):
                with open(self.config_file, 'r', encoding='utf-8') as f:
                    self.current_config = json.load(f)
                print(f"✅ Đã load config từ {self.config_file}")
            else:
```

```
                self.current_config = self.default_config.copy()
                print("⚠️ Config file không tồn tại, sử dụng defaults")
            return True
        except Exception as e:
            print(f"❌ Lỗi load JSON config: {e}")
            self.current_config = self.default_config.copy()
            return False

    def _load_ini_config(self):
        """Load INI config"""
        try:
            config = configparser.ConfigParser()

            if os.path.exists(self.ini_file):
                config.read(self.ini_file, encoding='utf-8')

                # Convert to dict
                self.current_config = {}
                for section in config.sections():
                    self.current_config[section] = dict(config[section])

                print(f"✅ Đã load INI config từ {self.ini_file}")
            else:
                self.current_config = self.default_config.copy()
                print("⚠️ INI file không tồn tại, sử dụng defaults")

            return True
        except Exception as e:
            print(f"❌ Lỗi load INI config: {e}")
            return False

    def save_config(self, format_type="json"):
        """Save configuration ra file"""
        if format_type == "json":
            return self._save_json_config()
        elif format_type == "ini":
            return self._save_ini_config()
        else:
            print(f"❌ Format không support: {format_type}")
            return False

    def _save_json_config(self):
        """Save JSON config"""
        try:
            # Add metadata
            config_with_meta = {
                '_metadata': {
                    'app_name': self.app_name,
                    'saved_at': datetime.now().isoformat(),
                    'version': '1.0'
                },
                'settings': self.current_config
            }

            with open(self.config_file, 'w', encoding='utf-8') as f:
                json.dump(config_with_meta, f, indent=2, ensure_ascii=False)
```

```
            print(f"✅ Đã save config: {self.config_file}")
            return True
        except Exception as e:
            print(f"❌ Lỗi save JSON config: {e}")
            return False

    def _save_ini_config(self):
        """Save INI config"""
        try:
            config = configparser.ConfigParser()

            # Add metadata section
            config['_metadata'] = {
                'app_name': self.app_name,
                'saved_at': datetime.now().isoformat(),
                'version': '1.0'
            }

            # Add settings
            for section_name, section_data in self.current_config.items():
                if isinstance(section_data, dict):
                    config[section_name] = section_data
                else:
                    # Flat settings go to DEFAULT section
                    config['DEFAULT'][section_name] = str(section_data)

            with open(self.ini_file, 'w', encoding='utf-8') as f:
                config.write(f)

            print(f"✅ Đã save INI config: {self.ini_file}")
            return True
        except Exception as e:
            print(f"❌ Lỗi save INI config: {e}")
            return False

    def get(self, key, default=None):
        """Lấy config value"""
        return self.current_config.get(key, default)

    def set(self, key, value):
        """Set config value"""
        self.current_config[key] = value
        print(f"✅ Set {key} = {value}")

    def update(self, updates):
        """Update multiple config values"""
        self.current_config.update(updates)
        print(f"✅ Updated {len(updates)} settings")

    def reset_to_defaults(self):
        """Reset về default values"""
        self.current_config = self.default_config.copy()
        print("🔄 Reset config về defaults")

    def show_config(self):
```

```
        """Hiển thị current config"""
        print(f"\n⚙️ CONFIG - {self.app_name.upper()}")
        print("="*40)

        if not self.current_config:
            print("(Config trống)")
            return

        for key, value in self.current_config.items():
            if isinstance(value, dict):
                print(f"📁 {key}:")
                for sub_key, sub_value in value.items():
                    print(f"   • {sub_key}: {sub_value}")
            else:
                print(f"⚙️ {key}: {value}")

    def backup_config(self):
        """Backup current config"""
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        backup_file = os.path.join(self.config_folder, f"{self.app_name}_backup_{ti

        try:
            with open(backup_file, 'w', encoding='utf-8') as f:
                json.dump(self.current_config, f, indent=2, ensure_ascii=False)

            print(f"💾 Đã backup config: {backup_file}")
            return backup_file
        except Exception as e:
            print(f"❌ Lỗi backup: {e}")
            return None

# Test Config Manager
print("\n=== CONFIG MANAGER ===")
cm = ConfigManager("calculator")

# Set defaults
defaults = {
    'ui': {
        'theme': 'dark',
        'font_size': 12,
        'language': 'vi'
    },
    'calculation': {
        'decimal_places': 2,
        'angle_unit': 'degrees',
        'scientific_mode': False
    },
    'features': {
        'auto_save': True,
        'history_limit': 100,
        'sound_enabled': False
    }
}

cm.set_defaults(defaults)
cm.load_config()
```

```
# Show config
cm.show_config()

# Update some settings
cm.set('ui', {'theme': 'light', 'font_size': 14, 'language': 'en'})
cm.set('calculation', {'decimal_places': 4, 'angle_unit': 'radians', 'scientific_mo

# Save config
cm.save_config('json')
cm.save_config('ini')

# Backup
backup_file = cm.backup_config()
```