DOCUMENTAZIONE PROGETTO PYTHON

Progetto: PySyncroNet

Cartella: C:\Users\Sigma\source\repos\python\PySyncroNet

Cartelle escluse:

- .continue
- .git
- .gradle
- .idea
- .metadata
- .recommenders
- .settings
- .venv
- .vs
- .vscode
- __pycache__
- bin
- build
- dist
- gradle
- jvm
- models2
- node_modules
- obj
- out
- packages
- target
- venv

File esclusi:

- *.temp
- *.tmp
- .DS_Store
- .env
- .env.local
- .env.production
- .gitattributes
- .gitignore
- config.py
- desktop.ini
- local_settings.py
- package-lock.json
- settings.py
- thumbs.db

- yarn.lock

Estensioni escluse:

- .7z
- .accdb
- .avi
- .bin
- .bmp
- .class
- .db
- .dll
- .doc
- .docx
- .ear
- .exe
- .flac
- .gif
- .gz
- .ico
- .idb
- .jar
- .jpeg
- .jpg
- .mdb
- .mkv
- .mov
- .mp3
- .mp4
- .ogg
- .pdb
- .pdf
- .png
- .ppt
- .pptx
- .pyc
- .pyd
- .pyo
- .rar
- .safetensors
- .so
- .sqlite
- .sqlite3
- .svg
- .tar
- .tiff
- .war
- .wav
- .webp
- .xls
- .xlsx
- .zip

File: folder_to_pdf.py

```
#File: folder_to_pdf.py
import tkinter as tk
from tkinter import ttk, filedialog, messagebox, scrolledtext
import threading
import os
from pathlib import Path
import sys
import webbrowser
from datetime import datetime
from fpdf import FPDF


class PythonProjectToPDF:
    def __init__(self):
        self.pdf = FPDF()
        self.pdf.set_auto_page_break(auto=True, margin=15)
        self.excluded_dirs = set()
        self.excluded_files = set()
        self.excluded_extensions = set()

    def should_exclude(self, file_path, relative_path):
        """Determina se un file o cartella dovrebbe essere escluso"""
        # Controlla se è una cartella esclusa
        if any(excluded_dir in str(relative_path).split(os.sep) for excluded_dir in self.excluded_dirs):
            return True

        # Controlla se è un file escluso
        if file_path.name in self.excluded_files:
            return True

        # Controlla l'estensione del file
        if file_path.suffix.lower() in self.excluded_extensions:
            return True

        return False

    def clean_text(self, text):
        """Pulisce il testo rimuovendo o sostituendo caratteri non compatibili con latin-1"""
        try:
            # Prova a codificare in latin-1 per verificare la compatibilità
            text.encode('latin-1')
            return text
        except UnicodeEncodeError:
            # Sostituisci i caratteri non compatibili
            cleaned_text = text.encode('latin-1', errors='replace').decode('latin-1')
            return cleaned_text

    def add_file_to_pdf(self, file_path, relative_path):
        """Aggiunge il contenuto di un file al PDF"""
        try:
            # Prova diverse codifiche
            encodings = ['utf-8', 'latin-1', 'cp1252', 'iso-8859-1']
            content = None

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

```

```
            if content is None:
                content = f"[Impossibile leggere il file {file_path} - formato binario o codifica sconosciuta]"

        except Exception as e:
            content = f"[Errore nella lettura del file {file_path}: {str(e)}]"

        # Pulisci il contenuto dai caratteri non compatibili
        content = self.clean_text(content)
        relative_path_str = self.clean_text(str(relative_path))

        # Aggiungi una nuova pagina per ogni file
        self.pdf.add_page()

        # Intestazione del file
        self.pdf.set_font('Arial', 'B', 14)
        self.pdf.cell(0, 10, f"File: {relative_path_str}", ln=True)
        self.pdf.ln(5)

        # Contenuto del file
        self.pdf.set_font('Courier', '', 8)  # Dimensione font più piccola per più contenuto

        # Dividi il contenuto in linee e aggiungi al PDF
        lines = content.split('\n')
        for i, line in enumerate(lines, 1):
            # Pulisci ogni linea
            clean_line = self.clean_text(line)

            # Gestisci linee troppo lunghe
            if len(clean_line) > 120:
                # Tronca le linee molto lunghe
                clean_line = clean_line[:120] + "... [troncato]"

            # Aggiungi numero di linea
            line_number = f"{i:4d} | {clean_line}"
            self.pdf.cell(0, 4, line_number, ln=True)  # Altezza linea più piccola

    def create_pdf(self, project_path, output_pdf="project_documentation.pdf", custom_exclusions=None):
        """Crea il PDF dalla cartella del progetto"""

        # Applica le esclusioni
        if custom_exclusions:
            if 'dirs' in custom_exclusions:
                self.excluded_dirs.update(custom_exclusions['dirs'])
            if 'files' in custom_exclusions:
                self.excluded_files.update(custom_exclusions['files'])
            if 'extensions' in custom_exclusions:
                self.excluded_extensions.update(custom_exclusions['extensions'])

        project_path = Path(project_path)

        if not project_path.exists():
            raise ValueError(f"La cartella '{project_path}' non esiste.")

        # Pagina titolo
        self.pdf.add_page()
        self.pdf.set_font('Arial', 'B', 20)
        self.pdf.cell(0, 20, "DOCUMENTAZIONE PROGETTO PYTHON", ln=True, align='C')
        self.pdf.ln(10)

        self.pdf.set_font('Arial', '', 12)
        self.pdf.cell(0, 10, f"Progetto: {project_path.name}", ln=True)
        self.pdf.cell(0, 10, f"Cartella: {project_path.absolute()}", ln=True)
        self.pdf.ln(10)

        # Informazioni sulle esclusioni
        self.pdf.set_font('Arial', 'B', 14)
        self.pdf.cell(0, 10, "Cartelle escluse:", ln=True)
```

```
        self.pdf.set_font('Arial', '', 10)
        for excluded_dir in sorted(self.excluded_dirs):
            self.pdf.cell(0, 5, f"  - {excluded_dir}", ln=True)

        self.pdf.ln(5)
        self.pdf.set_font('Arial', 'B', 14)
        self.pdf.cell(0, 10, "File esclusi:", ln=True)
        self.pdf.set_font('Arial', '', 10)
        for excluded_file in sorted(self.excluded_files):
            self.pdf.cell(0, 5, f"  - {excluded_file}", ln=True)

        self.pdf.ln(5)
        self.pdf.set_font('Arial', 'B', 14)
        self.pdf.cell(0, 10, "Estensioni escluse:", ln=True)
        self.pdf.set_font('Arial', '', 10)
        for excluded_ext in sorted(self.excluded_extensions):
            self.pdf.cell(0, 5, f"  - {excluded_ext}", ln=True)

        # Elenco dei file processati
        processed_files = []

        # Cerca ricorsivamente tutti i file
        for file_path in project_path.rglob('*'):
            if file_path.is_file():
                relative_path = file_path.relative_to(project_path)

                if not self.should_exclude(file_path, relative_path):
                    processed_files.append(str(relative_path))
                    self.add_file_to_pdf(file_path, relative_path)

        # Salva il PDF
        self.pdf.output(output_pdf)
        return len(processed_files)


class UniversalPDFToProject:
    def __init__(self):
        pass

    def recreate_project_structure(self, pdf_path, output_folder):
        """Placeholder per la ricostruzione del progetto"""
        # Per ora restituiamo True per simulare il successo
        # Implementa qui la logica di ricostruzione dal PDF
        return True


class AdvancedPDFProjectManager:
    def __init__(self, root):
        self.root = root
        self.root.title("? Advanced PDF Project Manager")
        self.root.geometry("1000x800")
        self.root.configure(bg="#2b2b2b")

        # Variabili principali
        self.project_path = tk.StringVar()
        self.output_pdf = tk.StringVar(value="project_documentation.pdf")
        self.pdf_to_read = tk.StringVar()
        self.reconstruction_output = tk.StringVar()

        # Esclusioni predefinite complete
        self.default_excluded_dirs = {
            'venv', '.venv', '__pycache__', '.git', '.vscode', '.idea',
            'node_modules', 'build', 'dist', 'models2', '.continue',
            '.vs', 'target', 'out', 'bin', 'obj', 'packages', '.gradle',
            '.settings', '.metadata', '.recommenders', 'gradle', 'jvm'
        }

```

```
199 |         self.default_excluded_files = {
200 |             'config.py', 'settings.py', 'local_settings.py', '.env',
201 |             '.gitignore', '.gitattributes', '.env.local', '.env.production',
202 |             'package-lock.json', 'yarn.lock', 'thumbs.db', '.DS_Store',
203 |             'desktop.ini'
204 |         }
205 |
206 |         self.default_excluded_extensions = {
207 |             '.pyc', '.pyo', '.pyd', '.so', '.dll', '.exe', '.safetensors',
208 |             '.bin', '.jpg', '.jpeg', '.png', '.gif', '.bmp', '.tiff', '.webp',
209 |             '.ico', '.svg', '.pdf', '.doc', '.docx', '.xls', '.xlsx', '.ppt',
210 |             '.pptx', '.zip', '.rar', '.7z', '.tar', '.gz', '.mp4', '.avi',
211 |             '.mkv', '.mov', '.mp3', '.wav', '.flac', '.ogg', '.db', '.sqlite',
212 |             '.sqlite3', '.mdb', '.accdb', '.pdb', '.idb', '.class', '.jar',
213 |             '.war', '.ear'
214 |         }
215 |
216 |         # Variabili per le esclusioni
217 |         self.exclude_dirs_var = tk.StringVar(value=", ".join(sorted(self.default_excluded_dirs)))
218 |         self.exclude_files_var = tk.StringVar(value=", ".join(sorted(self.default_excluded_files)))
219 |         self.exclude_extensions_var = tk.StringVar(value=", ".join(sorted(self.default_excluded_extensions)))
220 |
221 |         self.setup_styles()
222 |         self.setup_ui()
223 |
224 |     def setup_styles(self):
225 |         """Configura gli stili per l'interfaccia"""
226 |         style = ttk.Style()
227 |
228 |         # Tema per ttk
229 |         style.configure('Custom.TFrame', background='#2b2b2b')
230 |         style.configure('Custom.TLabel', background='#2b2b2b', foreground='#ffffff')
231 |         style.configure('Custom.TButton', background='#0e639c', foreground='#ffffff')
232 |         style.configure('Success.TButton', background='#388a34', foreground='#ffffff')
233 |         style.configure('Section.TLabelframe', background='#2b2b2b', foreground='#ce9178')
234 |         style.configure('Section.TLabelframe.Label', background='#2b2b2b', foreground='#ce9178')
235 |
236 |     def setup_ui(self):
237 |         """Configura l'interfaccia utente principale"""
238 |         # Header con titolo
239 |         header_frame = ttk.Frame(self.root, style='Custom.TFrame')
240 |         header_frame.pack(fill='x', padx=15, pady=10)
241 |
242 |         # Usiamo tk.Label invece di ttk.Label per avere più controllo sullo stile
243 |         title_label = tk.Label(header_frame,
244 |                                text="? Advanced PDF Project Manager",
245 |                                font=('Segoe UI', 20, 'bold'),
246 |                                bg='#2b2b2b',
247 |                                fg='#569cd6')
248 |         title_label.pack(pady=10)
249 |
250 |         subtitle_label = tk.Label(header_frame,
251 |                                   text="Crea PDF da progetti e ricrea progetti da PDF - Supporta tutti i tipi di
252 |                                   font=('Segoe UI', 12),
253 |                                   bg='#2b2b2b',
254 |                                   fg='#9cdcfe')
255 |         subtitle_label.pack(pady=5)
256 |
257 |         # Notebook per le schede
258 |         self.notebook = ttk.Notebook(self.root)
259 |         self.notebook.pack(fill='both', expand=True, padx=15, pady=10)
260 |
261 |         # Creazione delle schede
262 |         self.create_pdf_frame = ttk.Frame(self.notebook, style='Custom.TFrame')
263 |         self.recreate_frame = ttk.Frame(self.notebook, style='Custom.TFrame')
264 |         self.exclusions_frame = ttk.Frame(self.notebook, style='Custom.TFrame')
265 |         self.settings_frame = ttk.Frame(self.notebook, style='Custom.TFrame')
```

```
|
|         self.notebook.add(self.create_pdf_frame, text="? Crea PDF da Progetto")
|         self.notebook.add(self.recreate_frame, text="? Ricrea Progetto da PDF")
|         self.notebook.add(self.exclusions_frame, text="? Gestione Esclusioni")
|         self.notebook.add(self.settings_frame, text="?? Impostazioni & Info")
|
|         self.setup_create_pdf_tab()
|         self.setup_recreate_tab()
|         self.setup_exclusions_tab()
|         self.setup_settings_tab()
|
|         # Status bar
|         self.setup_status_bar()
|
|     # ... (tutto il resto del codice dell'interfaccia rimane uguale) ...
|
|     def create_pdf_thread(self):
|         """Thread per la creazione del PDF"""
|         try:
|             self.log_create("? Inizio creazione PDF...")
|             self.log_create(f"? Progetto: {self.project_path.get()}")
|             self.log_create(f"? Output: {self.output_pdf.get()}")
|
|             # Prepara le esclusioni dalle text area
|             custom_exclusions = {}
|
|             dirs_text = self.dirs_text.get(1.0, tk.END).strip()
|             if dirs_text:
|                 dirs = [d.strip() for d in dirs_text.split(',') if d.strip()]
|                 custom_exclusions['dirs'] = dirs
|                 self.log_create(f"? Cartelle escluse: {len(dirs)} elementi")
|
|             files_text = self.files_text.get(1.0, tk.END).strip()
|             if files_text:
|                 files = [f.strip() for f in files_text.split(',') if f.strip()]
|                 custom_exclusions['files'] = files
|                 self.log_create(f"? File esclusi: {len(files)} elementi")
|
|             extensions_text = self.extensions_text.get(1.0, tk.END).strip()
|             if extensions_text:
|                 exts = [e.strip() for e in extensions_text.split(',') if e.strip()]
|                 custom_exclusions['extensions'] = exts
|                 self.log_create(f"? Estensioni escluse: {len(exts)} elementi")
|
|             # Crea il PDF
|             converter = PythonProjectToPDF()
|             files_processed = converter.create_pdf(
|                 self.project_path.get(),
|                 self.output_pdf.get(),
|                 custom_exclusions
|             )
|
|             self.log_create(f"? PDF creato con successo! File processati: {files_processed}")
|             self.update_status("? PDF creato con successo!")
|             messagebox.showinfo("Successo", f"PDF creato con successo!\nFile processati: {files_processed}")
|
|         except Exception as e:
|             error_msg = f"? Errore durante la creazione del PDF: {str(e)}"
|             self.log_create(error_msg)
|             self.update_status("? Errore nella creazione PDF")
|             messagebox.showerror("Errore", f"Errore durante la creazione del PDF:\n{str(e)}")
|         finally:
|             self.create_pdf_btn.config(state='normal')
|
|     # ... (il resto del codice rimane invariato) ...
|
| def main():
```

```
    root = tk.Tk()
    app = AdvancedPDFProjectManager(root)
    root.mainloop()

if __name__ == "__main__":
    main()
```

File: pdf_to_folder.py

```
#File: pdf_to_folder.py
import re
import os
import PyPDF2
from pathlib import Path
import datetime

class UniversalPDFToProject:
    def __init__(self):
        self.files_data = {}
        self.metadata = {}

    def extract_pdf_content(self, pdf_path):
        """Estrae il contenuto dal PDF con gestione errori migliorata"""
        try:
            with open(pdf_path, 'rb') as file:
                pdf_reader = PyPDF2.PdfReader(file)
                full_text = ""

                for page_num, page in enumerate(pdf_reader.pages):
                    page_text = page.extract_text()
                    full_text += page_text + "\n"

                return full_text
        except Exception as e:
            print(f"Errore nell'estrazione del PDF: {e}")
            return None

    def parse_files_from_pdf(self, pdf_text):
        """Analizza il PDF e estrae tutti i file con il loro contenuto"""
        lines = pdf_text.split('\n')
        current_file = None
        current_content = []
        reading_file_content = False

        for line in lines:
            line = line.strip()

            # Cerca l'inizio di un nuovo file
            if line.startswith('File:'):
                # Salva il file precedente se esiste
                if current_file and current_content:
                    # Unisci il contenuto e pulisci
                    full_content = '\n'.join(current_content).strip()
                    if full_content:
                        self.files_data[current_file] = full_content

                # Inizia un nuovo file
                file_path = line[5:].strip()  # Rimuove "File:"
                current_file = file_path
                current_content = []
                reading_file_content = True
                continue

            # Se stiamo leggendo il contenuto di un file e non è una linea di metadati
            if (reading_file_content and current_file and
                line and
                not line.startswith('DOCUMENTAZIONE') and
                not line.startswith('Cartelle') and
                not line.startswith('File') and
                not line.startswith('Estensioni') and
                not line.startswith('Progetto:') and
                not line.startswith('Cartella:')):

```

```
                # Gestisce le linee numerate (pattern: "123 | contenuto")
                line_match = re.match(r'^\s*(\d+)\s*\|\s*(.*)$', line)
                if line_match:
                    content_part = line_match.group(2)
                    current_content.append(content_part)
                else:
                    # Se non è una linea numerata ma abbiamo contenuto, potrebbe essere continuazione
                    if current_content and not re.match(r'^\s*\d+\s*\|', line):
                        current_content[-1] += ' ' + line
                    elif line:  # Nuova linea di contenuto
                        current_content.append(line)

        # Salva l'ultimo file
        if current_file and current_content:
            full_content = '\n'.join(current_content).strip()
            if full_content:
                self.files_data[current_file] = full_content

        return self.files_data

    def clean_file_content(self, content, file_extension):
        """Pulisce il contenuto in base al tipo di file"""
        if not content:
            return content

        # Rimuove i tag [troncato] se presenti
        content = content.replace('... [troncato]', '')

        # Gestione specifica per diversi tipi di file
        if file_extension in ['.py', '.js', '.java', '.c', '.cpp', '.h', '.cs']:
            # Linguaggi di programmazione - corregge l'indentazione
            content = self.fix_code_indentation(content)
        elif file_extension in ['.html', '.xml', '.svg']:
            # File markup - corregge la formattazione
            content = self.fix_markup_formatting(content)
        elif file_extension in ['.json']:
            # JSON - tenta di correggere la formattazione
            content = self.fix_json_formatting(content)
        elif file_extension in ['.css', '.scss', '.less']:
            # CSS - corregge la formattazione
            content = self.fix_css_formatting(content)
        elif file_extension in ['.md', '.rst', '.txt']:
            # Documentazione - pulizia base
            content = self.fix_text_formatting(content)

        return content

    def fix_code_indentation(self, content):
        """Corregge l'indentazione per codice sorgente"""
        lines = content.split('\n')
        fixed_lines = []
        indent_level = 0
        in_multiline_string = False
        string_delimiter = None

        for line in lines:
            stripped = line.strip()
            if not stripped:
                fixed_lines.append('')
                continue

            # Gestione stringhe multilinea
            if not in_multiline_string:
                if '"""' in stripped or "'''" in stripped:
                    in_multiline_string = True
                    string_delimiter = '"""' if '"""' in stripped else "'''"
            else:
```

```
                if string_delimiter in stripped:
                    in_multiline_string = False

            if in_multiline_string:
                fixed_lines.append('    ' * indent_level + stripped)
                continue

            # Calcola indentazione per codice
            line_indent = 0

            # Riduci indentazione per certe keyword
            if (stripped.startswith(('return', 'break', 'continue', 'pass')) or
                stripped in ('else:', 'elif:', 'except:', 'finally:')):
                line_indent = max(0, indent_level - 1)
            else:
                line_indent = indent_level

            fixed_lines.append('    ' * line_indent + stripped)

            # Aumenta indentazione dopo certe strutture
            if (stripped.endswith(':') and
                not stripped.startswith('#') and
                not any(stripped.startswith(kw) for kw in ['import', 'from', 'def ', 'class '])):
                indent_level += 1
            # Riduci indentazione per fine blocco implicito
            elif stripped in ['return', 'break', 'continue', 'pass']:
                indent_level = max(0, indent_level - 1)

        return '\n'.join(fixed_lines)

    def fix_markup_formatting(self, content):
        """Corregge la formattazione per file markup (HTML, XML, etc.)"""
        # Unisce tag che potrebbero essere stati spezzati
        content = re.sub(r'<\s*/\s*(\w+)\s*>', r'</\1>', content)
        content = re.sub(r'<(\w+)([^>]*)>\s*<\s*/\s*\1\s*>', r'<\1\2></\1>', content)

        lines = content.split('\n')
        fixed_lines = []
        indent_level = 0

        for line in lines:
            stripped = line.strip()
            if not stripped:
                fixed_lines.append('')
                continue

            # Gestione tag di chiusura
            if stripped.startswith('</'):
                indent_level = max(0, indent_level - 1)

            fixed_lines.append('  ' * indent_level + stripped)

            # Gestione tag di apertura (non auto-chiusi)
            if (stripped.startswith('<') and
                not stripped.startswith('</') and
                not stripped.endswith('/>') and
                not '<!--' in stripped and
                not stripped.endswith('-->')):
                indent_level += 1

        return '\n'.join(fixed_lines)

    def fix_json_formatting(self, content):
        """Corregge la formattazione JSON"""
        try:
            # Tenta di parsare e riformattare il JSON
            import json
```

```
            parsed = json.loads(content)
            return json.dumps(parsed, indent=2, ensure_ascii=False)
        except:
            # Se il parsing fallisce, fa del suo meglio
            content = re.sub(r',\s*', ', ', content)
            content = re.sub(r':\s*', ': ', content)
            return content

    def fix_css_formatting(self, content):
        """Corregge la formattazione CSS"""
        lines = content.split('\n')
        fixed_lines = []
        in_rule = False

        for line in lines:
            stripped = line.strip()
            if not stripped:
                fixed_lines.append('')
                continue

            if stripped.endswith('{'):
                fixed_lines.append(stripped)
                in_rule = True
            elif stripped == '}':
                fixed_lines.append(stripped)
                in_rule = False
            elif in_rule:
                fixed_lines.append('  ' + stripped)
            else:
                fixed_lines.append(stripped)

        return '\n'.join(fixed_lines)

    def fix_text_formatting(self, content):
        """Corregge la formattazione per file di testo"""
        # Unisce paragrafi spezzati
        lines = content.split('\n')
        fixed_lines = []
        current_paragraph = []

        for line in lines:
            stripped = line.strip()
            if not stripped:
                if current_paragraph:
                    fixed_lines.append(' '.join(current_paragraph))
                    current_paragraph = []
                fixed_lines.append('')
            elif stripped.startswith(('#', '-', '*', '>')):  # Mantiene formattazione markdown
                if current_paragraph:
                    fixed_lines.append(' '.join(current_paragraph))
                    current_paragraph = []
                fixed_lines.append(stripped)
            else:
                current_paragraph.append(stripped)

        if current_paragraph:
            fixed_lines.append(' '.join(current_paragraph))

        return '\n'.join(fixed_lines)

    def get_file_extension(self, file_path):
        """Restituisce l'estensione del file"""
        path = Path(file_path)
        return path.suffix.lower()

    def recreate_project_structure(self, pdf_path, output_folder):
        """Ricrea l'intera struttura del progetto dal PDF"""
```

```
        print(f"? Leggendo il PDF: {pdf_path}")
        pdf_text = self.extract_pdf_content(pdf_path)

        if not pdf_text:
            print("? Impossibile leggere il PDF")
            return False

        print("? Analizzando il contenuto del PDF...")
        files_data = self.parse_files_from_pdf(pdf_text)

        if not files_data:
            print("? Nessun file trovato nel PDF")
            return False

        print(f"? Trovati {len(files_data)} file nel PDF")

        output_path = Path(output_folder)
        output_path.mkdir(parents=True, exist_ok=True)

        files_created = 0
        errors = []

        for file_path, raw_content in files_data.items():
            try:
                full_path = output_path / file_path

                # Crea le directory necessarie
                full_path.parent.mkdir(parents=True, exist_ok=True)

                # Ottieni l'estensione del file
                file_extension = self.get_file_extension(file_path)

                # Pulisci il contenuto in base al tipo di file
                cleaned_content = self.clean_file_content(raw_content, file_extension)

                # Scrivi il file
                with open(full_path, 'w', encoding='utf-8') as f:
                    f.write(cleaned_content)

                files_created += 1
                print(f"? Creato: {file_path}")

            except Exception as e:
                error_msg = f"? Errore con {file_path}: {str(e)}"
                errors.append(error_msg)
                print(error_msg)

        # Scrivi un report di ricostruzione
        self.write_reconstruction_report(output_path, files_created, errors, files_data)

        print(f"\n? Ricostruzione completata!")
        print(f"? File creati: {files_created}")
        print(f"? Errori: {len(errors)}")
        print(f"? Output: {output_path.absolute()}")

        if errors:
            print("\nErrori riscontrati:")
            for error in errors:
                print(f"  - {error}")

        return True

    def write_reconstruction_report(self, output_path, files_created, errors, files_data):
        """Scrive un report dettagliato della ricostruzione"""
        report_content = f"""RICOSTRUZIONE PROGETTO DA PDF
==============================
```

```
| Data ricostruzione: {datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")}
| File creati con successo: {files_created}
| Errori riscontrati: {len(errors)}
|
| DETTAGLIO FILE RICOSTRUITI:
| ---------------------------
| """
|
|         for file_path in sorted(files_data.keys()):
|             file_extension = self.get_file_extension(file_path)
|             content_length = len(files_data[file_path])
|             report_content += f"- {file_path} ({file_extension}, {content_length} caratteri)\n"
|
|         if errors:
|             report_content += f"\nERRORI RISCONTRATI:\n-------------------\n"
|             for error in errors:
|                 report_content += f"- {error}\n"
|
|         report_content += f"""
| STATISTICHE:
| -----------
| File totali nel PDF: {len(files_data)}
| File creati: {files_created}
| Success rate: {(files_created/len(files_data))*100:.1f}%
|
| ESTENSIONI FILE RICOSTRUITE:
| ---------------------------
| """
|
|         # Calcola statistiche per estensione
|         extensions = {}
|         for file_path in files_data.keys():
|             ext = self.get_file_extension(file_path)
|             extensions[ext] = extensions.get(ext, 0) + 1
|
|         for ext, count in sorted(extensions.items()):
|             report_content += f"- {ext or 'Nessuna'}: {count} file\n"
|
|         with open(output_path / "RICOSTRUZIONE_REPORT.txt", 'w', encoding='utf-8') as f:
|             f.write(report_content)
|
| def main():
|     import argparse
|
|     parser = argparse.ArgumentParser(
|         description='Ricrea un intero progetto da PDF - Supporta qualsiasi tipo di file di testo/codice'
|     )
|     parser.add_argument('pdf_path', help='Percorso del PDF sorgente')
|     parser.add_argument('output_folder', help='Cartella di destinazione per il progetto ricostruito')
|
|     args = parser.parse_args()
|
|     if not os.path.exists(args.pdf_path):
|         print(f"? Errore: Il file PDF '{args.pdf_path}' non esiste.")
|         return
|
|     print("??  Ricostruzione progetto da PDF")
|     print("=" * 50)
|
|     recreator = UniversalPDFToProject()
|     success = recreator.recreate_project_structure(args.pdf_path, args.output_folder)
|
|     if success:
|         print("\n? Ricostruzione completata con successo!")
|         print(f"? Il progetto è stato ricreato in: {args.output_folder}")
|     else:
|         print("\n? Ricostruzione fallita!")
```

```
|
| if __name__ == "__main__":
|     # Esempio di utilizzo diretto
|     if len(os.sys.argv) == 1:
|         print("Ricostruzione progetto da PDF")
|         print("=" * 40)
|
|         pdf_file = input("Inserisci il percorso del PDF: ").strip()
|         output_dir = input("Inserisci la cartella di output: ").strip()
|
|         if pdf_file and output_dir:
|             if os.path.exists(pdf_file):
|                 recreator = UniversalPDFToProject()
|                 recreator.recreate_project_structure(pdf_file, output_dir)
|             else:
|                 print("? Il file PDF specificato non esiste.")
|         else:
|             print("? Devi specificare entrambi i percorsi.")
|     else:
|         main()
```

File: syncroNet.py

```
#File: syncroNet.py
import tkinter as tk
from tkinter import ttk, filedialog, messagebox, scrolledtext
import threading
import os
from pathlib import Path
import sys
import webbrowser
from datetime import datetime

# Importa le classi dai tuoi file
from folder_to_pdf import PythonProjectToPDF
from pdf_to_folder import UniversalPDFToProject


class AdvancedPDFProjectManager:
    def __init__(self, root):
        self.root = root
        self.root.title("? Advanced PDF Project Manager")
        self.root.geometry("1000x800")
        self.root.configure(bg="#2b2b2b")

        # Variabili principali
        self.project_path = tk.StringVar()
        self.output_pdf = tk.StringVar(value="project_documentation.pdf")
        self.pdf_to_read = tk.StringVar()
        self.reconstruction_output = tk.StringVar()

        # Esclusioni predefinite complete
        self.default_excluded_dirs = {
            'venv', '.venv', '__pycache__', '.git', '.vscode', '.idea',
            'node_modules', 'build', 'dist', 'models2', '.continue',
            '.vs', 'target', 'out', 'bin', 'obj', 'packages', '.gradle',
            '.settings', '.metadata', '.recommenders', 'gradle', 'jvm'
        }

        self.default_excluded_files = {
            'config.py', 'settings.py', 'local_settings.py', '.env',
            '.gitignore', '.gitattributes', '.env.local', '.env.production',
            'package-lock.json', 'yarn.lock', 'thumbs.db', '.DS_Store',
            'desktop.ini', '*.tmp', '*.temp'
        }

        self.default_excluded_extensions = {
            '.pyc', '.pyo', '.pyd', '.so', '.dll', '.exe', '.safetensors',
            '.bin', '.jpg', '.jpeg', '.png', '.gif', '.bmp', '.tiff', '.webp',
            '.ico', '.svg', '.pdf', '.doc', '.docx', '.xls', '.xlsx', '.ppt',
            '.pptx', '.zip', '.rar', '.7z', '.tar', '.gz', '.mp4', '.avi',
            '.mkv', '.mov', '.mp3', '.wav', '.flac', '.ogg', '.db', '.sqlite',
            '.sqlite3', '.mdb', '.accdb', '.pdb', '.idb', '.class', '.jar',
            '.war', '.ear'
        }

        # Variabili per le esclusioni
        self.exclude_dirs_var = tk.StringVar(value=", ".join(sorted(self.default_excluded_dirs)))
        self.exclude_files_var = tk.StringVar(value=", ".join(sorted(self.default_excluded_files)))
        self.exclude_extensions_var = tk.StringVar(value=", ".join(sorted(self.default_excluded_extensions)))

        self.setup_styles()
        self.setup_ui()

    def setup_styles(self):
        """Configura gli stili per l'interfaccia"""
        style = ttk.Style()
```

```
65 |
66 |         # Tema per ttk
67 |         style.configure('Custom.TFrame', background='#2b2b2b')
68 |         style.configure('Custom.TLabel', background='#2b2b2b', foreground='#ffffff')
69 |         style.configure('Custom.TButton', background='#0e639c', foreground='#ffffff')
70 |         style.configure('Success.TButton', background='#388a34', foreground='#ffffff')
71 |         style.configure('Section.TLabelframe', background='#2b2b2b', foreground='#ce9178')
72 |         style.configure('Section.TLabelframe.Label', background='#2b2b2b', foreground='#ce9178')
73 |
74 |     def setup_ui(self):
75 |         """Configura l'interfaccia utente principale"""
76 |         # Header con titolo
77 |         header_frame = ttk.Frame(self.root, style='Custom.TFrame')
78 |         header_frame.pack(fill='x', padx=15, pady=10)
79 |
80 |         # Usiamo tk.Label invece di ttk.Label per avere più controllo sullo stile
81 |         title_label = tk.Label(header_frame,
82 |                                text="? Advanced PDF Project Manager",
83 |                                font=('Segoe UI', 20, 'bold'),
84 |                                bg='#2b2b2b',
85 |                                fg='#569cd6')
86 |         title_label.pack(pady=10)
87 |
88 |         subtitle_label = tk.Label(header_frame,
89 |                                   text="Crea PDF da progetti e ricrea progetti da PDF - Supporta tutti i tipi di
90 |                                   font=('Segoe UI', 12),
91 |                                   bg='#2b2b2b',
92 |                                   fg='#9cdcfe')
93 |         subtitle_label.pack(pady=5)
94 |
95 |         # Notebook per le schede
96 |         self.notebook = ttk.Notebook(self.root)
97 |         self.notebook.pack(fill='both', expand=True, padx=15, pady=10)
98 |
99 |         # Creazione delle schede
100 |         self.create_pdf_frame = ttk.Frame(self.notebook, style='Custom.TFrame')
101 |         self.recreate_frame = ttk.Frame(self.notebook, style='Custom.TFrame')
102 |         self.exclusions_frame = ttk.Frame(self.notebook, style='Custom.TFrame')
103 |         self.settings_frame = ttk.Frame(self.notebook, style='Custom.TFrame')
104 |
105 |         self.notebook.add(self.create_pdf_frame, text="? Crea PDF da Progetto")
106 |         self.notebook.add(self.recreate_frame, text="? Ricrea Progetto da PDF")
107 |         self.notebook.add(self.exclusions_frame, text="? Gestione Esclusioni")
108 |         self.notebook.add(self.settings_frame, text="?? Impostazioni & Info")
109 |
110 |         self.setup_create_pdf_tab()
111 |         self.setup_recreate_tab()
112 |         self.setup_exclusions_tab()
113 |         self.setup_settings_tab()
114 |
115 |         # Status bar
116 |         self.setup_status_bar()
117 |
118 |     def setup_create_pdf_tab(self):
119 |         """Configura la scheda per creare PDF da progetto"""
120 |         main_frame = ttk.Frame(self.create_pdf_frame, style='Custom.TFrame')
121 |         main_frame.pack(fill='both', expand=True, padx=10, pady=10)
122 |
123 |         # Sezione selezione progetto
124 |         project_section = ttk.LabelFrame(main_frame, text="? Selezione Progetto", style='Section.TLabelframe')
125 |         project_section.pack(fill='x', pady=10)
126 |
127 |         project_label = tk.Label(project_section, text="Cartella del progetto:",
128 |                                  font=('Segoe UI', 10), bg='#2b2b2b', fg='#ffffff')
129 |         project_label.grid(row=0, column=0, sticky='w', pady=8, padx=10)
130 |
131 |         project_entry = tk.Entry(project_section, textvariable=self.project_path,
```

```
                                width=70, font=('Segoe UI', 9),
                                bg='#3c3c3c', fg='#ffffff', insertbackground='#ffffff')
        project_entry.grid(row=0, column=1, padx=8, pady=8, sticky='ew')

        browse_project_btn = ttk.Button(project_section, text="Sfoglia ?",
                                        command=self.browse_project, width=15)
        browse_project_btn.grid(row=0, column=2, padx=8, pady=8)

        # Sezione output PDF
        output_section = ttk.LabelFrame(main_frame, text="? Output PDF", style='Section.TLabelframe')
        output_section.pack(fill='x', pady=10)

        output_label = tk.Label(output_section, text="File PDF di output:",
                                font=('Segoe UI', 10), bg='#2b2b2b', fg='#ffffff')
        output_label.grid(row=0, column=0, sticky='w', pady=8, padx=10)

        output_entry = tk.Entry(output_section, textvariable=self.output_pdf,
                                width=70, font=('Segoe UI', 9),
                                bg='#3c3c3c', fg='#ffffff', insertbackground='#ffffff')
        output_entry.grid(row=0, column=1, padx=8, pady=8, sticky='ew')

        browse_output_btn = ttk.Button(output_section, text="Sfoglia ?",
                                       command=self.browse_output_pdf, width=15)
        browse_output_btn.grid(row=0, column=2, padx=8, pady=8)

        # Sezione log
        log_section = ttk.LabelFrame(main_frame, text="? Log di Creazione", style='Section.TLabelframe')
        log_section.pack(fill='both', expand=True, pady=10)

        self.create_log = scrolledtext.ScrolledText(
            log_section,
            height=15,
            width=90,
            font=('Consolas', 9),
            bg='#1e1e1e',
            fg='#d4d4d4',
            insertbackground='#ffffff',
            selectbackground='#264f78'
        )
        self.create_log.pack(fill='both', expand=True, padx=10, pady=10)

        # Pulsanti azione
        button_frame = ttk.Frame(main_frame, style='Custom.TFrame')
        button_frame.pack(fill='x', pady=15)

        clear_log_btn = ttk.Button(button_frame, text="? Pulisci Log",
                                   command=self.clear_create_log)
        clear_log_btn.pack(side='left', padx=5)

        export_log_btn = ttk.Button(button_frame, text="? Esporta Log",
                                    command=self.export_create_log)
        export_log_btn.pack(side='left', padx=5)

        stats_btn = ttk.Button(button_frame, text="? Statistiche Progetto",
                               command=self.show_project_stats)
        stats_btn.pack(side='left', padx=5)

        self.create_pdf_btn = ttk.Button(
            button_frame,
            text="? Crea PDF",
            command=self.start_create_pdf,
            style='Success.TButton',
            width=20
        )
        self.create_pdf_btn.pack(side='right', padx=5)

    def setup_recreate_tab(self):
```

```
199 |         """Configura la scheda per ricreare progetto da PDF"""
200 |         main_frame = ttk.Frame(self.recreate_frame, style='Custom.TFrame')
201 |         main_frame.pack(fill='both', expand=True, padx=10, pady=10)
202 |
203 |         # Sezione PDF sorgente
204 |         pdf_section = ttk.LabelFrame(main_frame, text="? PDF Sorgente", style='Section.TLabelframe')
205 |         pdf_section.pack(fill='x', pady=10)
206 |
207 |         pdf_label = tk.Label(pdf_section, text="File PDF:",
208 |                              font=('Segoe UI', 10), bg='#2b2b2b', fg='#ffffff')
209 |         pdf_label.grid(row=0, column=0, sticky='w', pady=8, padx=10)
210 |
211 |         pdf_entry = tk.Entry(pdf_section, textvariable=self.pdf_to_read,
212 |                              width=70, font=('Segoe UI', 9),
213 |                              bg='#3c3c3c', fg='#ffffff', insertbackground='#ffffff')
214 |         pdf_entry.grid(row=0, column=1, padx=8, pady=8, sticky='ew')
215 |
216 |         browse_pdf_btn = ttk.Button(pdf_section, text="Sfoglia ?",
217 |                                     command=self.browse_pdf, width=15)
218 |         browse_pdf_btn.grid(row=0, column=2, padx=8, pady=8)
219 |
220 |         # Sezione output progetto
221 |         output_section = ttk.LabelFrame(main_frame, text="? Output Progetto", style='Section.TLabelframe')
222 |         output_section.pack(fill='x', pady=10)
223 |
224 |         output_label = tk.Label(output_section, text="Cartella di output:",
225 |                                 font=('Segoe UI', 10), bg='#2b2b2b', fg='#ffffff')
226 |         output_label.grid(row=0, column=0, sticky='w', pady=8, padx=10)
227 |
228 |         output_entry = tk.Entry(output_section, textvariable=self.reconstruction_output,
229 |                                 width=70, font=('Segoe UI', 9),
230 |                                 bg='#3c3c3c', fg='#ffffff', insertbackground='#ffffff')
231 |         output_entry.grid(row=0, column=1, padx=8, pady=8, sticky='ew')
232 |
233 |         browse_output_btn = ttk.Button(output_section, text="Sfoglia ?",
234 |                                        command=self.browse_reconstruction_output, width=15)
235 |         browse_output_btn.grid(row=0, column=2, padx=8, pady=8)
236 |
237 |         # Sezione opzioni
238 |         options_section = ttk.LabelFrame(main_frame, text="?? Opzioni Ricostruzione", style='Section.TLabelfram
239 |         options_section.pack(fill='x', pady=10)
240 |
241 |         self.auto_open_folder = tk.BooleanVar(value=True)
242 |         self.create_report = tk.BooleanVar(value=True)
243 |         self.overwrite_existing = tk.BooleanVar(value=False)
244 |
245 |         auto_open_cb = tk.Checkbutton(options_section, text="Apri cartella output automaticamente",
246 |                                       variable=self.auto_open_folder, font=('Segoe UI', 9),
247 |                                       bg='#2b2b2b', fg='#ffffff', selectcolor='#3c3c3c')
248 |         auto_open_cb.grid(row=0, column=0, sticky='w', pady=4, padx=10)
249 |
250 |         report_cb = tk.Checkbutton(options_section, text="Crea report di ricostruzione",
251 |                                    variable=self.create_report, font=('Segoe UI', 9),
252 |                                    bg='#2b2b2b', fg='#ffffff', selectcolor='#3c3c3c')
253 |         report_cb.grid(row=0, column=1, sticky='w', pady=4, padx=10)
254 |
255 |         overwrite_cb = tk.Checkbutton(options_section, text="Sovrascrivi file esistenti",
256 |                                       variable=self.overwrite_existing, font=('Segoe UI', 9),
257 |                                       bg='#2b2b2b', fg='#ffffff', selectcolor='#3c3c3c')
258 |         overwrite_cb.grid(row=0, column=2, sticky='w', pady=4, padx=10)
259 |
260 |         # Sezione log
261 |         log_section = ttk.LabelFrame(main_frame, text="? Log di Ricostruzione", style='Section.TLabelframe')
262 |         log_section.pack(fill='both', expand=True, pady=10)
263 |
264 |         self.recreate_log = scrolledtext.ScrolledText(
265 |             log_section,
```

```
            height=15,
            width=90,
            font=('Consolas', 9),
            bg='#1e1e1e',
            fg='#d4d4d4',
            insertbackground='#ffffff',
            selectbackground='#264f78'
        )
        self.recreate_log.pack(fill='both', expand=True, padx=10, pady=10)

        # Pulsanti azione
        button_frame = ttk.Frame(main_frame, style='Custom.TFrame')
        button_frame.pack(fill='x', pady=15)

        clear_log_btn = ttk.Button(button_frame, text="? Pulisci Log",
                                   command=self.clear_recreate_log)
        clear_log_btn.pack(side='left', padx=5)

        export_log_btn = ttk.Button(button_frame, text="? Esporta Log",
                                    command=self.export_recreate_log)
        export_log_btn.pack(side='left', padx=5)

        open_folder_btn = ttk.Button(button_frame, text="? Apri Cartella Output",
                                     command=self.open_output_folder)
        open_folder_btn.pack(side='left', padx=5)

        self.recreate_btn = ttk.Button(
            button_frame,
            text="? Ricrea Progetto",
            command=self.start_recreate_project,
            style='Success.TButton',
            width=20
        )
        self.recreate_btn.pack(side='right', padx=5)

    def setup_exclusions_tab(self):
        """Configura la scheda per gestire le esclusioni"""
        main_frame = ttk.Frame(self.exclusions_frame, style='Custom.TFrame')
        main_frame.pack(fill='both', expand=True, padx=10, pady=10)

        # Descrizione
        desc_section = ttk.LabelFrame(main_frame, text="?? Informazioni Esclusioni", style='Section.TLabelframe
        desc_section.pack(fill='x', pady=10)

        desc_text = """
Queste esclusioni vengono applicate automaticamente quando crei un PDF da un progetto.
I file e cartelle esclusi non verranno inclusi nel PDF generato.

? Cartelle: Directory intere da escludere (es: venv, node_modules)
? File: Nomi specifici di file da escludere (es: .env, config.py)
? Estensioni: Tipi di file da escludere per estensione (es: .jpg, .exe)

Separare i valori con virgole.
        """

        desc_label = tk.Label(desc_section, text=desc_text, justify='left',
                              font=('Segoe UI', 9), bg='#2b2b2b', fg='#d4d4d4')
        desc_label.pack(pady=10, padx=10, anchor='w')

        # Sezione cartelle escluse
        dirs_section = ttk.LabelFrame(main_frame, text="? Cartelle Escluse", style='Section.TLabelframe')
        dirs_section.pack(fill='x', pady=10)

        dirs_label = tk.Label(dirs_section, text="Cartelle da escludere:",
                              font=('Segoe UI', 10), bg='#2b2b2b', fg='#ffffff')
        dirs_label.pack(anchor='w', padx=10, pady=5)

```

```
        self.dirs_text = scrolledtext.ScrolledText(
            dirs_section,
            height=4,
            width=90,
            font=('Consolas', 9),
            bg='#1e1e1e',
            fg='#d4d4d4'
        )
        self.dirs_text.pack(fill='x', padx=10, pady=5)
        self.dirs_text.insert('1.0', ", ".join(sorted(self.default_excluded_dirs)))

        # Sezione file esclusi
        files_section = ttk.LabelFrame(main_frame, text="? File Esclusi", style='Section.TLabelframe')
        files_section.pack(fill='x', pady=10)

        files_label = tk.Label(files_section, text="File da escludere:",
                               font=('Segoe UI', 10), bg='#2b2b2b', fg='#ffffff')
        files_label.pack(anchor='w', padx=10, pady=5)

        self.files_text = scrolledtext.ScrolledText(
            files_section,
            height=4,
            width=90,
            font=('Consolas', 9),
            bg='#1e1e1e',
            fg='#d4d4d4'
        )
        self.files_text.pack(fill='x', padx=10, pady=5)
        self.files_text.insert('1.0', ", ".join(sorted(self.default_excluded_files)))

        # Sezione estensioni escluse
        extensions_section = ttk.LabelFrame(main_frame, text="? Estensioni Escluse", style='Section.TLabelframe
        extensions_section.pack(fill='x', pady=10)

        extensions_label = tk.Label(extensions_section, text="Estensioni da escludere:",
                                    font=('Segoe UI', 10), bg='#2b2b2b', fg='#ffffff')
        extensions_label.pack(anchor='w', padx=10, pady=5)

        self.extensions_text = scrolledtext.ScrolledText(
            extensions_section,
            height=4,
            width=90,
            font=('Consolas', 9),
            bg='#1e1e1e',
            fg='#d4d4d4'
        )
        self.extensions_text.pack(fill='x', padx=10, pady=5)
        self.extensions_text.insert('1.0', ", ".join(sorted(self.default_excluded_extensions)))

        # Pulsanti gestione esclusioni
        button_frame = ttk.Frame(main_frame, style='Custom.TFrame')
        button_frame.pack(fill='x', pady=15)

        save_btn = ttk.Button(button_frame, text="? Salva Esclusioni",
                              command=self.save_exclusions, style='Success.TButton')
        save_btn.pack(side='left', padx=5)

        reset_btn = ttk.Button(button_frame, text="? Ripristina Default",
                               command=self.reset_exclusions)
        reset_btn.pack(side='left', padx=5)

        export_btn = ttk.Button(button_frame, text="? Esporta Configurazione",
                                command=self.export_exclusions)
        export_btn.pack(side='left', padx=5)

        import_btn = ttk.Button(button_frame, text="? Importa Configurazione",
                                command=self.import_exclusions)
```

```
        import_btn.pack(side='left', padx=5)

    def setup_settings_tab(self):
        """Configura la scheda impostazioni e informazioni"""
        main_frame = ttk.Frame(self.settings_frame, style='Custom.TFrame')
        main_frame.pack(fill='both', expand=True, padx=10, pady=10)

        # Informazioni applicazione
        info_section = ttk.LabelFrame(main_frame, text="?? Informazioni Applicazione", style='Section.TLabelfra
        info_section.pack(fill='x', pady=10)

        info_text = """
? Advanced PDF Project Manager - Versione 3.0

Funzionalità Principali:
? ? Crea PDF da progetti (supporta tutti i tipi di file di testo/codice)
? ? Ricrea progetti completi da PDF
? ? Gestione avanzata delle esclusioni (file binari, immagini, etc.)
? ? Interfaccia moderna con tema scuro
? ? Operazioni in background non bloccanti

Supporta:
? Python, JavaScript, Java, C++, C#, HTML, CSS, JSON, XML, Markdown
? Tutti i linguaggi di programmazione e file di testo
? Strutture progetto complesse con sottocartelle

Tecnologie:
? Python 3.8+
? Tkinter per l'interfaccia
? PyPDF2 per l'elaborazione PDF
? Threading per operazioni non bloccanti
        """

        info_label = tk.Label(info_section, text=info_text, justify='left',
                              font=('Segoe UI', 9), bg='#2b2b2b', fg='#d4d4d4')
        info_label.pack(pady=10, padx=10, anchor='w')

        # Statistiche e utilità
        stats_section = ttk.LabelFrame(main_frame, text="? Statistiche & Utilità", style='Section.TLabelframe')
        stats_section.pack(fill='x', pady=10)

        # Pulsanti utilità
        utils_frame = ttk.Frame(stats_section, style='Custom.TFrame')
        utils_frame.pack(fill='x', pady=10)

        docs_btn = ttk.Button(utils_frame, text="? Documentazione",
                              command=self.open_documentation, width=20)
        docs_btn.pack(side='left', padx=5)

        bug_btn = ttk.Button(utils_frame, text="? Segnala Bug",
                             command=self.report_bug, width=20)
        bug_btn.pack(side='left', padx=5)

        suggest_btn = ttk.Button(utils_frame, text="? Suggerimenti",
                                 command=self.suggest_features, width=20)
        suggest_btn.pack(side='left', padx=5)

        update_btn = ttk.Button(utils_frame, text="? Controlla Aggiornamenti",
                                command=self.check_updates, width=20)
        update_btn.pack(side='left', padx=5)

    def setup_status_bar(self):
        """Configura la status bar"""
        status_frame = ttk.Frame(self.root, relief='sunken', padding=2, style='Custom.TFrame')
        status_frame.pack(fill='x', side='bottom')

        self.status_var = tk.StringVar(value="? Pronto - Seleziona un'operazione per iniziare")
```

```
        status_label = tk.Label(status_frame, textvariable=self.status_var,
                                font=('Segoe UI', 9), bg='#2b2b2b', fg='#569cd6')
        status_label.pack(side='left')

    def browse_project(self):
        path = filedialog.askdirectory(title="? Seleziona cartella progetto")
        if path:
            self.project_path.set(path)
            self.update_status(f"? Progetto selezionato: {Path(path).name}")

    def browse_output_pdf(self):
        path = filedialog.asksaveasfilename(
            title="? Salva PDF come",
            defaultextension=".pdf",
            filetypes=[("PDF files", "*.pdf"), ("Tutti i file", "*.*")]
        )
        if path:
            self.output_pdf.set(path)

    def browse_pdf(self):
        path = filedialog.askopenfilename(
            title="? Seleziona PDF",
            filetypes=[("PDF files", "*.pdf"), ("Tutti i file", "*.*")]
        )
        if path:
            self.pdf_to_read.set(path)
            self.update_status(f"? PDF selezionato: {Path(path).name}")

    def browse_reconstruction_output(self):
        path = filedialog.askdirectory(title="? Seleziona cartella output")
        if path:
            self.reconstruction_output.set(path)

    def update_status(self, message):
        self.status_var.set(f"? {datetime.now().strftime('%H:%M:%S')} - {message}")
        self.root.update_idletasks()

    def log_create(self, message):
        self.create_log.insert(tk.END, f"{datetime.now().strftime('%H:%M:%S')} - {message}\n")
        self.create_log.see(tk.END)
        self.root.update_idletasks()

    def log_recreate(self, message):
        self.recreate_log.insert(tk.END, f"{datetime.now().strftime('%H:%M:%S')} - {message}\n")
        self.recreate_log.see(tk.END)
        self.root.update_idletasks()

    def clear_create_log(self):
        self.create_log.delete(1.0, tk.END)
        self.update_status("Log di creazione pulito")

    def clear_recreate_log(self):
        self.recreate_log.delete(1.0, tk.END)
        self.update_status("Log di ricostruzione pulito")

    def export_create_log(self):
        self.export_log(self.create_log, "create_log.txt")

    def export_recreate_log(self):
        self.export_log(self.recreate_log, "recreate_log.txt")

    def export_log(self, log_widget, filename):
        content = log_widget.get(1.0, tk.END)
        path = filedialog.asksaveasfilename(
            title="? Esporta log",
            defaultextension=".txt",
            initialfile=filename,
```

```
534 |             filetypes=[("Text files", "*.txt"), ("Tutti i file", "*.*")]
535 |         )
536 |         if path:
537 |             with open(path, 'w', encoding='utf-8') as f:
538 |                 f.write(content)
539 |             self.update_status(f"Log esportato: {Path(path).name}")
540 |
541 |     def save_exclusions(self):
542 |         """Salva le esclusioni dalle text area"""
543 |         try:
544 |             dirs_text = self.dirs_text.get(1.0, tk.END).strip()
545 |             files_text = self.files_text.get(1.0, tk.END).strip()
546 |             extensions_text = self.extensions_text.get(1.0, tk.END).strip()
547 |
548 |             self.exclude_dirs_var.set(dirs_text)
549 |             self.exclude_files_var.set(files_text)
550 |             self.exclude_extensions_var.set(extensions_text)
551 |
552 |             messagebox.showinfo("Successo", "Esclusioni salvate con successo!")
553 |             self.update_status("Configurazione esclusioni salvata")
554 |
555 |         except Exception as e:
556 |             messagebox.showerror("Errore", f"Errore nel salvataggio: {str(e)}")
557 |
558 |     def reset_exclusions(self):
559 |         """Ripristina le esclusioni predefinite"""
560 |         self.dirs_text.delete(1.0, tk.END)
561 |         self.dirs_text.insert('1.0', ", ".join(sorted(self.default_excluded_dirs)))
562 |
563 |         self.files_text.delete(1.0, tk.END)
564 |         self.files_text.insert('1.0', ", ".join(sorted(self.default_excluded_files)))
565 |
566 |         self.extensions_text.delete(1.0, tk.END)
567 |         self.extensions_text.insert('1.0', ", ".join(sorted(self.default_excluded_extensions)))
568 |
569 |         self.update_status("Esclusioni ripristinate ai valori predefiniti")
570 |
571 |     def export_exclusions(self):
572 |         """Esporta la configurazione delle esclusioni"""
573 |         try:
574 |             path = filedialog.asksaveasfilename(
575 |                 title="? Esporta configurazione esclusioni",
576 |                 defaultextension=".txt",
577 |                 initialfile="exclusions_config.txt",
578 |                 filetypes=[("Text files", "*.txt"), ("Tutti i file", "*.*")]
579 |             )
580 |             if path:
581 |                 config = f"""# Advanced PDF Project Manager - Configurazione Esclusioni
582 | # Esportato il: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}
583 |
584 | [DIRS]
585 | {self.dirs_text.get(1.0, tk.END).strip()}
586 |
587 | [FILES]
588 | {self.files_text.get(1.0, tk.END).strip()}
589 |
590 | [EXTENSIONS]
591 | {self.extensions_text.get(1.0, tk.END).strip()}
592 | """
593 |                 with open(path, 'w', encoding='utf-8') as f:
594 |                     f.write(config)
595 |                 self.update_status(f"Configurazione esportata: {Path(path).name}")
596 |
597 |         except Exception as e:
598 |             messagebox.showerror("Errore", f"Errore nell'esportazione: {str(e)}")
599 |
600 |     def import_exclusions(self):
```

```
601 |         """Importa la configurazione delle esclusioni"""
602 |         try:
603 |             path = filedialog.askopenfilename(
604 |                 title="? Importa configurazione esclusioni",
605 |                 filetypes=[("Text files", "*.txt"), ("Tutti i file", "*.*")]
606 |             )
607 |             if path:
608 |                 with open(path, 'r', encoding='utf-8') as f:
609 |                     content = f.read()
610 |
611 |                 # Parsing semplice del file di configurazione
612 |                 sections = content.split('[')
613 |                 for section in sections:
614 |                     if section.startswith('DIRS]'):
615 |                         dirs_content = section[5:].split('[')[0].strip()
616 |                         self.dirs_text.delete(1.0, tk.END)
617 |                         self.dirs_text.insert('1.0', dirs_content)
618 |                     elif section.startswith('FILES]'):
619 |                         files_content = section[6:].split('[')[0].strip()
620 |                         self.files_text.delete(1.0, tk.END)
621 |                         self.files_text.insert('1.0', files_content)
622 |                     elif section.startswith('EXTENSIONS]'):
623 |                         extensions_content = section[10:].split('[')[0].strip()
624 |                         self.extensions_text.delete(1.0, tk.END)
625 |                         self.extensions_text.insert('1.0', extensions_content)
626 |
627 |                 self.update_status(f"Configurazione importata: {Path(path).name}")
628 |
629 |         except Exception as e:
630 |             messagebox.showerror("Errore", f"Errore nell'importazione: {str(e)}")
631 |
632 |     def show_project_stats(self):
633 |         """Mostra statistiche del progetto selezionato"""
634 |         if not self.project_path.get() or not os.path.exists(self.project_path.get()):
635 |             messagebox.showwarning("Attenzione", "Seleziona prima un progetto valido")
636 |             return
637 |
638 |         try:
639 |             project_path = Path(self.project_path.get())
640 |             total_files = 0
641 |             total_size = 0
642 |             extensions = {}
643 |
644 |             for file_path in project_path.rglob('*'):
645 |                 if file_path.is_file():
646 |                     total_files += 1
647 |                     total_size += file_path.stat().st_size
648 |                     ext = file_path.suffix.lower()
649 |                     extensions[ext] = extensions.get(ext, 0) + 1
650 |
651 |             stats_text = f"""
652 | Statistiche Progetto: {project_path.name}
653 |
654 | File totali: {total_files}
655 | Dimensione totale: {total_size / (1024*1024):.2f} MB
656 |
657 | Estensioni principali:
658 | """
659 |             for ext, count in sorted(extensions.items(), key=lambda x: x[1], reverse=True)[:10]:
660 |                 stats_text += f"  {ext or 'Nessuna'}: {count} file\n"
661 |
662 |             messagebox.showinfo("? Statistiche Progetto", stats_text)
663 |
664 |         except Exception as e:
665 |             messagebox.showerror("Errore", f"Errore nel calcolo statistiche: {str(e)}")
666 |
667 |     def open_output_folder(self):
```

```
668 |         """Apre la cartella di output nel file explorer"""
669 |         if self.reconstruction_output.get() and os.path.exists(self.reconstruction_output.get()):
670 |             os.startfile(self.reconstruction_output.get())
671 |             self.update_status("Cartella output aperta")
672 |         else:
673 |             messagebox.showwarning("Attenzione", "Cartella di output non valida")
674 |
675 |     def open_documentation(self):
676 |         webbrowser.open("https://github.com/your-repo/documentation")
677 |
678 |     def report_bug(self):
679 |         webbrowser.open("https://github.com/your-repo/issues")
680 |
681 |     def suggest_features(self):
682 |         webbrowser.open("https://github.com/your-repo/discussions")
683 |
684 |     def check_updates(self):
685 |         messagebox.showinfo("Aggiornamenti", "? Controllo aggiornamenti in sviluppo")
686 |
687 |     def start_create_pdf(self):
688 |         """Avvia la creazione del PDF in un thread separato"""
689 |         if not self.project_path.get():
690 |             messagebox.showerror("Errore", "Seleziona una cartella progetto")
691 |             return
692 |
693 |         if not self.output_pdf.get():
694 |             messagebox.showerror("Errore", "Specifica un file PDF di output")
695 |             return
696 |
697 |         self.create_pdf_btn.config(state='disabled')
698 |         self.update_status("? Avvio creazione PDF...")
699 |
700 |         thread = threading.Thread(target=self.create_pdf_thread)
701 |         thread.daemon = True
702 |         thread.start()
703 |
704 |     def create_pdf_thread(self):
705 |         """Thread per la creazione del PDF"""
706 |         try:
707 |             self.log_create("? Inizio creazione PDF...")
708 |             self.log_create(f"? Progetto: {self.project_path.get()}")
709 |             self.log_create(f"? Output: {self.output_pdf.get()}")
710 |
711 |             # Prepara le esclusioni dalle text area
712 |             custom_exclusions = {}
713 |
714 |             dirs_text = self.dirs_text.get(1.0, tk.END).strip()
715 |             if dirs_text:
716 |                 dirs = [d.strip() for d in dirs_text.split(',') if d.strip()]
717 |                 custom_exclusions['dirs'] = dirs
718 |                 self.log_create(f"? Cartelle escluse: {len(dirs)} elementi")
719 |
720 |             files_text = self.files_text.get(1.0, tk.END).strip()
721 |             if files_text:
722 |                 files = [f.strip() for f in files_text.split(',') if f.strip()]
723 |                 custom_exclusions['files'] = files
724 |                 self.log_create(f"? File esclusi: {len(files)} elementi")
725 |
726 |             extensions_text = self.extensions_text.get(1.0, tk.END).strip()
727 |             if extensions_text:
728 |                 exts = [e.strip() for e in extensions_text.split(',') if e.strip()]
729 |                 custom_exclusions['extensions'] = exts
730 |                 self.log_create(f"? Estensioni escluse: {len(exts)} elementi")
731 |
732 |             # Crea il PDF usando la classe dal tuo file
733 |             converter = PythonProjectToPDF()
734 |             files_processed = converter.create_pdf(
```

```
                self.project_path.get(),
                self.output_pdf.get(),
                custom_exclusions
            )

            self.log_create(f"? PDF creato con successo! File processati: {files_processed}")
            self.update_status("? PDF creato con successo!")
            messagebox.showinfo("Successo", f"PDF creato con successo!\nFile processati: {files_processed}")

        except Exception as e:
            error_msg = f"? Errore durante la creazione del PDF: {str(e)}"
            self.log_create(error_msg)
            self.update_status("? Errore nella creazione PDF")
            messagebox.showerror("Errore", f"Errore durante la creazione del PDF:\n{str(e)}")
        finally:
            self.create_pdf_btn.config(state='normal')

    def start_recreate_project(self):
        """Avvia la ricostruzione del progetto in un thread separato"""
        if not self.pdf_to_read.get():
            messagebox.showerror("Errore", "Seleziona un file PDF")
            return

        if not self.reconstruction_output.get():
            messagebox.showerror("Errore", "Specifica una cartella di output")
            return

        if not os.path.exists(self.pdf_to_read.get()):
            messagebox.showerror("Errore", "Il file PDF specificato non esiste")
            return

        self.recreate_btn.config(state='disabled')
        self.update_status("? Avvio ricostruzione progetto...")

        thread = threading.Thread(target=self.recreate_project_thread)
        thread.daemon = True
        thread.start()

    def recreate_project_thread(self):
        """Thread per la ricostruzione del progetto"""
        try:
            self.log_recreate("? Inizio ricostruzione progetto...")
            self.log_recreate(f"? PDF sorgente: {self.pdf_to_read.get()}")
            self.log_recreate(f"? Output: {self.reconstruction_output.get()}")

            # Ricrea il progetto usando la classe dal tuo file
            recreator = UniversalPDFToProject()
            success = recreator.recreate_project_structure(
                self.pdf_to_read.get(),
                self.reconstruction_output.get()
            )

            if success:
                self.log_recreate("? Progetto ricostruito con successo!")
                self.update_status("? Progetto ricostruito con successo!")

                if self.auto_open_folder.get():
                    self.open_output_folder()

                messagebox.showinfo("Successo", "Progetto ricostruito con successo!")
            else:
                self.log_recreate("? Ricostruzione fallita!")
                self.update_status("? Ricostruzione fallita")
                messagebox.showerror("Errore", "Ricostruzione del progetto fallita!")

        except Exception as e:
            error_msg = f"? Errore durante la ricostruzione: {str(e)}"
```

```
            self.log_recreate(error_msg)
            self.update_status("? Errore nella ricostruzione")
            messagebox.showerror("Errore", f"Errore durante la ricostruzione:\n{str(e)}")
        finally:
            self.recreate_btn.config(state='normal')

def main():
    root = tk.Tk()
    app = AdvancedPDFProjectManager(root)
    root.mainloop()

if __name__ == "__main__":
    main()
```