## An Industrial Oriented Mini Project (CS705PC)

on

## “SMART FILE ORGANIZATION SYSTEM USING AI”

Submitted

in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology**

in

## Computer Science and Engineering (Data Science)

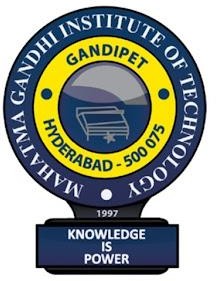
by

## Mr. BATTURI MEGHANATH

## H.T.No: 21261A6705

Under the guidance of **Dr. M. Rama Bai**

(Professor and HoD)



Department of Emerging Technologies

## MAHATMA GANDHI INSTITUTE OF TECHNOLOGY (AUTONOMOUS)

(Affiliated to Jawaharlal Nehru Technological University Hyderabad) Gandipet, Hyderabad – 500075, Telangana (India).

2024 - 2025

MAHATMA GANDHI INSTITUTE OF TECHNOLOGY

(AUTONOMOUS)

**(Affiliated to Jawaharlal Nehru Technological University Hyderabad) GANDIPET, HYDERABAD, TELANGANA– 500 075 (INDIA)**

## CERTIFICATE



This is to certify that the project entitled **“SMART FILE ORGANIZATION SYSTEM USING AI”** is being submitted by **Mr. BATTURI MEGHANATH** bearing **Roll No: 21261A6705** in partial fulfilment for the award of **Bachelor of Technology** in **Computer Science and Engineering (Data Science)** is a record of bonafide work carried out by him/her under our guidance and supervision.

The results embodied in this project have not been submitted to any other University or Institute for the award of any degree or diploma.

Project Guide Head of the Department

### Dr. M. Rama Bai Dr. M. Rama Bai

(Professor & HoD)

Department of ET

**EXTERNAL EXAMINER**

# DECLARATION

This is to certify that the work reported in this project titled “**SMART FILE ORGANIZATION SYSTEM USING AI”** is a record of work done by me in the Department of Emerging Technologies, Mahatma Gandhi Institute of Technology, Hyderabad.

No part of the work is copied from books/journals/internet and wherever the portion is taken, the same has been duly referred to in the text. The report is based on the work done entirely by me and not copied from any other source.

### BATTURI MEGHANATH

**(21261A6705)**

# ACKNOWLEDGEMENT

I would like to express my sincere thanks to **Dr. G. ChandraMohan Reddy, Principal, MGIT**, for providing the working facilities in college.

I wish to express my sincere thanks and gratitude to **Dr. M. Rama Bai, Professor and HoD**, Department of Emerging Technologies, MGIT, for all the timely support and valuable suggestions during the period of project.

I am extremely thankful to **Ms. Karuna Verma, Assistant Professor,** and **Mr. Mallela Srikanth, Assistant Professor,** Department of Emerging Technologies, MGIT, Industrial Oriented Minor Project coordinators for their encouragement and support throughout the project.

Finally, I would also like to thank all the faculty and staff of the ET Department who helped me directly or indirectly, for completing this project.

### BATTURI MEGHANATH

**(21261A6705)**

# TABLE OF CONTENTS

[CERTIFICATE i](#_Toc184842184)

[DECLARATION ii](#_Toc184842186)

[ACKNOWLEDGEMENT iii](#_Toc184842188)

[TABLE OF CONTENTS iv](#_Toc184842190)

[LIST OF FIGURES v](#_Toc184842191)

[ABSTRACT vi](#_Toc184842192)

[1. INTRODUCTION 1](#_Toc184842193)

[1.2 Existing System 2](#_Toc184842194)

[1.3 Proposed System 2](#_Toc184842195)

[1.4 Requirements Specification 3](#_Toc184842196)

[2. LITERATURE SURVEY 4](#_Toc184842199)

[3. SYSTEM DESIGN 7](#_Toc184842200)

[3.1 UML Diagrams 7](#_Toc184842201)

[3.2 Technology Stack 10](#_Toc184842206)

[3.3 Methodology 10](#_Toc184842207)

[4. IMPLEMENTATION AND RESULTS 13](#_Toc184842216)

[4.1 Implementation 13](#_Toc184842217)

[4.2 Results 14](#_Toc184842218)

[5. CONCLUSION AND FUTURE SCOPE 17](#_Toc184842224)

[5.1 Conclusion 17](#_Toc184842225)

[5.2 Future Scope 17](#_Toc184842226)

[BIBLIOGRAPHY 18](#_Toc184842227)

[APPENDIX 19](#_Toc184842228)

# LIST OF FIGURES

|  |  |  |
| --- | --- | --- |
| Figure 3.1  Figure 3.2  Figure 3.3  Figure 4.1  Figure 4.2  Figure 4.3  Figure 4.4  Figure 4.5  Figure 4.6  Figure 4.7 | Use Case Diagram Sequence Diagram Activity Diagram  Saving API key in .env file  Execution of the program  Directory before execution  Directory after execution  Sorted audio files  Sorted image files  Sorted documents | 8  9  10  14  15  15  16  16  17  17 |

# ABSTRACT

File management has become an increasingly complex task in today's digital era, with users accumulating a vast array of files in various formats. This project, titled **"Smart File Organization System Using AI"**, aims to simplify and automate file organization by leveraging artificial intelligence and advanced algorithms. The system provides a multi-functional approach to handle diverse file types, including audio files, images, documents, and more.

The system integrates various modules to perform specific tasks: sorting files by extension, organizing audio files based on metadata, categorizing images through object detection using **YOLOv8**, removing duplicate files using **hashing techniques**, and classifying documents into intelligent categories using **Google Gemini API**. By combining rule-based sorting methods with AI-powered categorization, the system offers a robust and efficient solution for managing cluttered directories.

This project highlights the potential of AI in enhancing daily productivity while addressing challenges such as metadata inconsistencies, duplicate file handling, and dynamic categorization. The **Smart File Organization System** is a scalable and extensible solution designed to meet the needs of modern users, paving the way for future advancements in autonomous file management.

.

* 1. **About the Project**

# INTRODUCTION

The **Smart File Organization System Using AI** is a cutting-edge solution developed to simplify file management in an era of digital overload. With a modular and automated approach, the system tackles the complexities of handling a diverse range of file types, including audio files, images, documents, and more. The project aims to reduce the time and effort required for manual file organization by employing advanced algorithms and artificial intelligence techniques.

This system integrates multiple functionalities into a single cohesive framework. Each module is designed to address a specific aspect of file management, such as sorting by file type, removing duplicates, or categorizing files based on their contents. The core functionalities include:

1. **File Sorting by Extension**: Files are grouped into predefined categories like audio, video, documents, images, and code based on their extensions.
2. **Audio File Organization**: Music files are sorted into folders based on their album metadata, leveraging tools like Mutagen for metadata extraction.
3. **Image Categorization**: Using the YOLOv8 object detection model, images are categorized into folders based on detected objects within them.
4. **Duplicate File Removal**: Duplicate files are identified and removed using hash-based comparison techniques to optimize storage.
5. **Document Classification**: Text-based content in documents is analyzed using the Google Gemini API to categorize files into intelligent groups like academic notes, financial documents, or personal letters.

The project emphasizes the use of AI for dynamic and context-aware file categorization, particularly in areas like document sorting and image detection, where traditional methods fall short. By combining these diverse capabilities, the system offers a robust solution that is not only efficient but also scalable for future enhancements.

This project showcases the potential of artificial intelligence and automation in simplifying everyday tasks, making it a valuable tool for individuals and organizations striving for better productivity and digital hygiene.

## Existing System

Existing file organization systems often rely on manual sorting or basic extension-based categorization, which lacks the ability to handle large, heterogeneous datasets efficiently. Traditional file managers and operating systems, such as Windows Explorer or macOS Finder, provide limited tools for grouping files based on attributes like type or creation date but fail to utilize content-based techniques. Open-source tools like **Tagspaces** have introduced features like tagging and metadata-based search, but they still require significant user input. More advanced approaches, such as the use of **First-Order Logic (FOL)** or kernel algorithms for layout or similarity-based clustering, attempt to address specific domains but remain constrained in adaptability and scalability. Recent machine learning-based systems leverage methods like K-Means clustering, TF-IDF for feature extraction, and neural networks for categorization, enabling automated, content-driven file organization. However, these systems often focus on specific file types (e.g., text or PDFs), leaving multimedia files largely unsupported. These limitations highlight the need for unified, scalable, and fully automated systems capable of efficiently handling diverse file formats and evolving user needs.

## Proposed System

Our proposed system, the **Smart File Organization System Using AI**, offers a unified and comprehensive solution for automating file management across diverse file types, addressing the limitations of existing systems. By integrating rule-based sorting with advanced AI-driven categorization, our system efficiently handles audio, images, documents, and general files. Unlike traditional systems that rely solely on extensions or metadata, we utilize state-of-the-art technologies like YOLOv8 for object detection in images, Google Gemini for intelligent document categorization, and hashing algorithms for duplicate removal. The inclusion of real-time sorting by content and metadata provides a significant enhancement over existing models, ensuring scalability and minimal manual intervention. Additionally, the system's modular architecture allows for seamless expansion to support emerging file formats, making it a versatile and adaptive tool for modern file management.

## Requirements Specification

### Software requirements

* Operating System : Windows 10 or later
* Programming language : Python 3.8 or higher
* IDE : VScode
* API credentials : Google Gemini API key

### Hardware requirements

* Processor : Intel Core-i5 and above
* RAM : 8 GB

# LITERATURE SURVEY

1. **"Automatic Document Organization Exploiting FOL Similarity-based Techniques,"** **[1]** authored by Stefano Ferilli, T.M.A. Basile, Marenglen Biba, and Floriana Esposito, was published in 2008. This paper explores innovative methods for automatically organizing large collections of digital documents based on their layout structures. It leverages First-Order Logic (FOL) formulas as a powerful representation language to describe document layouts, overcoming the limitations of classical attribute-value models. The research proposes a novel similarity framework tailored to FOL descriptions, addressing computational challenges such as indeterminacy in mappings. By focusing on layout structure rather than content, the paper introduces an alternative yet complementary approach to traditional content-based clustering methods. The proposed similarity framework includes innovative formulas that assess relationships between layout components using FOL, making it robust for handling complex relational data. Experimental validation demonstrates high precision and recall in clustering documents with similar layouts, even under challenging conditions such as high complexity and overlapping styles.
2. **"Automating the Organizing Process for PDF Files ”, [2]** authored by Samual Kahsay in 2016, addresses the need for an automated solution to organize digital files, focusing specifically on PDF files. The research highlights the manual labor involved in file organization and proposes a novel approach leveraging **similarity and metric learning**. By implementing kernel algorithms and enhancing the capabilities of an open-source tool (Tagspaces), the study presents a framework for automating file organization. The developed system incorporates machine learning techniques to group files based on their metadata and similarities, minimizing manual intervention and offering a scalable solution for virtual data management. The study explores Tagspaces, an existing open-source file organization software, as the foundation for the system. Building on its functionalities, the author integrates machine learning algorithms to automate PDF organization. The proposed approach consists of two applications: one organizes files based on extensions, while the other classifies PDFs using keyword-based metadata matching.
3. **"Content-Based Automated File Organization Using Machine Learning Approaches",** [3]**[3]** authored by Syed Ali Raza, Sagheer Abbas, Taher M. Ghazal, Muhammad Adnan Khan, Munir Ahmad, and Hussam Al Hamadi, was published in 2022. This paper addresses the growing challenges of managing heterogeneous digital files in the era of big data. The study focuses on creating a system that organizes files based on their content rather than relying on file extensions or manual sorting. Employing a hybrid approach of supervised and unsupervised machine learning techniques, the authors propose a framework that efficiently categorizes files into relevant groups using content-based similarity measures. By integrating K-Means clustering, feature extraction, and neural network fine-tuning, the proposed system demonstrates significant potential in automating file organization tasks. This research significantly advances automated file management by shifting the focus from traditional file extension-based sorting to intelligent content-based organization. The proposed system eliminates the need for manual file sorting, making it particularly beneficial for users with large and varied datasets. While the initial implementation focuses on text-based files, the study highlights its scalability to include images, audio, and video files in future iterations.
4. **"Automatic Classification of Office Documents: Review of Available Methods and Techniques,"** **[4]** authored by Dobrica Savic and published in 1995, examines existing methodologies for automating the classification of office documents. The study focuses on identifying techniques suitable for addressing the labor-intensive nature of manual document classification in office environments. It provides a detailed review of statistical, linguistic, and artificial intelligence (AI)-based approaches. Statistical methods include word frequency and weighting, offering foundational tools for indexing and sorting documents. Linguistic methods delve deeper into semantic and syntactic analysis for understanding document content. AI methods, particularly expert systems and neural networks, are explored for their potential to model human-like reasoning and self-learning in document classification. The paper concludes that while statistical and linguistic methods offer foundational capabilities, AI-driven systems show the greatest promise for effective and scalable document classification.
5. **"Object Detection using YOLO: Challenges, Architectural Successors, Datasets, and Applications"** **[5]** authored by Tausif Diwan, G. Anirudh, and Jitendra V. Tembhurne, was published in 2023. This paper provides a comprehensive review of object detection methodologies with a focus on YOLO (You Only Look Once) and its architectural advancements. YOLO, a single-stage object detector, offers a high-speed alternative to traditional two-stage detectors like RCNNs, with applications ranging from surveillance to autonomous driving. The research categorizes object detection approaches, compares single- and two-stage detectors, and evaluates YOLO against benchmarks using metrics such as accuracy and inference time. Successive YOLO versions (v1 to v4) are analyzed for their innovations, including architecture changes, training optimizations, and enhanced loss functions. The paper also examines key challenges like detecting small objects and handling complex datasets. Experimental results highlight YOLO’s adaptability and efficiency, with YOLOv4 striking an effective balance between speed and precision.

# 3.SYSTEM DESIGN

## UML Diagrams

## The SMART FILE ORGANIZATION SYSTEM USING AI is represented through various UML diagrams, each illustrating a different aspect of the system's design and functionality. These diagrams provide a structured view of the system's architecture, interactions, and workflows, aiding in better understanding and implementation. Below are the key UML diagrams for this project:

### Use Case Diagram

**Use case diagram** demonstrates the interactions between the user and the system, highlighting the major functionalities of the application. Figure 3.1 represents the use case diagram of the project.

* The user provides the directory path as input.
* The system performs tasks such as file sorting by extension, organizing audio files, categorizing images, removing duplicates, and AI-based document categorization.
* Dependencies between modules, such as document sorting relying on AI categorization, are represented.

### Sequence Diagram

Sequence diagrams illustrates the sequential flow of operations in the system, showcasing the interaction between various components and the user. Figure 3.2 represents the sequence diagram of **SMART FILE ORGANIZATION SYSTEM.**

* The user initiates the organization process by providing the directory path.
* The system sequentially calls modules like the extension sorter, audio sorter, image sorter, duplicate remover, and document sorter.
* AI categorization is invoked during document sorting for intelligent classification.

A diagram of a smart file organization system

Description automatically generated**Figure 3.1:** Use Case Diagram

A screenshot of a computer

Description automatically generated

**Figure 3.2:** Sequence Diagram

### Activity Diagram

An **activity diagram** depicts the workflow of the system, emphasizing parallel processing and decision points. Figure 3.3 represents the Activity diagram of **SMART FILE ORGANIZATION SYSYTEM**.

* The system begins by validating the user-provided directory.
* Files are first sorted by extension, followed by parallel tasks such as audio organization, image categorization, duplicate removal, and document processing.
* Conditional steps, like duplicate detection and metadata extraction, ensure efficient organization.
* A diagram of a system

  Description automatically generatedThe process concludes with the creation of a fully organized directory structure

**Figure 3.3:** Activity Diagram

## Technology Stack

The technology stack for the **Smart File Organization System Using AI** integrates a range of programming languages, libraries, and tools to enable efficient file organization, metadata extraction, and AI-driven categorization. Below is a breakdown of the technology stack:

* **Python:** The core programming language used across the project for implementing file organization logic, object detection, and AI-based categorization workflows.
* **Mutagen:** A Python library for audio metadata handling, enabling the extraction and organization of audio files based on properties like album or artist.
* **Ultralytics YOLO:** A state-of-the-art object detection library used for categorizing images by identifying objects within them.
* **PyPDF2 and python-docx:** Libraries used for reading and extracting content from PDF and DOCX files to analyze their content for categorization.
* **google.generativeai:** Provides AI-powered document analysis and intelligent category generation using Google's Gemini API for enhanced document sorting.
* **dotenv:** A library used to manage environment variables securely, such as api keys required for google gemini integration.
* **shutil and os.** : Standard Python libraries for file and directory manipulation, enabling operations like file movement and folder creation.
* **hashlib:** A Python library for generating file hashes to identify and remove duplicate files effectively**.**
* **Pandas**: Used for data organization and handling intermediate categorization results efficiently during the document processing pipeline.
* **Streamlit (Optional)**: A web application framework considered for building user-friendly interfaces to interact with the file organization system, if required.

## Methodology

### Input Collection and Directory Initialization

The process begins with the user providing the directory path that contains unorganized files. The system validates the provided directory to ensure it exists and is accessible. Once the directory is initialized, the system scans all files within it, identifying their extensions and metadata. This step lays the foundation for subsequent sorting operations by ensuring that the entire file structure is mapped. The directory path serves as the central point for processing, and subdirectories are created dynamically as per the classification needs.

### File Sorting by Extension

In the first layer of organization, the files are sorted into predefined categories based on their extensions. Categories include Audio, Video, Documents, Photos, Codes, and Others. The system creates corresponding folders if they do not already exist and moves files into these folders. This step is implemented using Python's os and shutil libraries, ensuring a systematic arrangement. By handling the files based on extensions, this stage provides an initial structure, simplifying further specialized operations for each file type.

### Audio File Organization

Audio files are processed using the **Mutagen** library, which extracts metadata such as album names, artists, and genres. The extracted metadata is used to create folder names, and the files are moved into their respective folders. For instance, songs from the same album are grouped together. If metadata is missing or incomplete, the system places such files in a default folder like "Unknown Album." This approach ensures that audio files are logically grouped, making it easier for users to navigate their music collections.

### Image Categorization

Image categorization is achieved using the **YOLOv8 object detection model**, a state-of-the-art tool for identifying objects within images. The system scans each image, detects objects, and creates folders named after the identified objects. For example, images containing "dogs" are moved into a "Dogs" folder. This content-based approach provides a highly intuitive organization for images, allowing users to find images based on their content rather than file names or extensions. YOLO’s accuracy ensures robust performance even with diverse and complex image datasets.

### Duplicate File Detection and Removal

Duplicate files are identified using a hashing mechanism. The system computes an MD5 hash for each file, which uniquely represents its content. Files with identical hashes are flagged as duplicates. The system then provides users with a choice to review and delete these duplicate files. This step is essential for optimizing storage and preventing redundant data clutter. The use of hashing ensures that even if file names differ, duplicates are accurately detected based on their content.

### Document Analysis and Categorization

Documents, including PDFs, DOCX, and TXT files, are analyzed to extract their text content using libraries like PyPDF2 and python-docx. The extracted text is passed to the Google Gemini API, which leverages AI to categorize the documents into meaningful folders such as "Banking Documents," "Academic Notes," or "Letters." This step uses advanced natural language processing to understand the context of the documents, going beyond simple keyword matching. It ensures that documents are not just sorted by format but also by relevance and purpose.

### Intelligent Category Refinement

After the initial categorization, the system refines the generated folder names to ensure clarity and eliminate redundancy. Similar categories are merged, and ambiguous folder names are renamed for better organization. For instance, folders like "Bank Statements" and "Financial Documents" might be consolidated into a single "Finance" folder. This refinement process enhances the user experience by reducing folder clutter and ensuring consistent naming conventions across the directory.

### Integration and Execution

All individual modules are executed in a sequential manner within the main application, ensuring that the output of one step seamlessly feeds into the next. For example, files are first sorted by extension, and duplicates are removed before being passed on to content-based categorization modules. The integration ensures that all file types—audio, images, and documents—are processed comprehensively without overlapping tasks. The final organized directory structure is presented to the user, ready for access.

# IMPLEMENTATION AND RESULTS

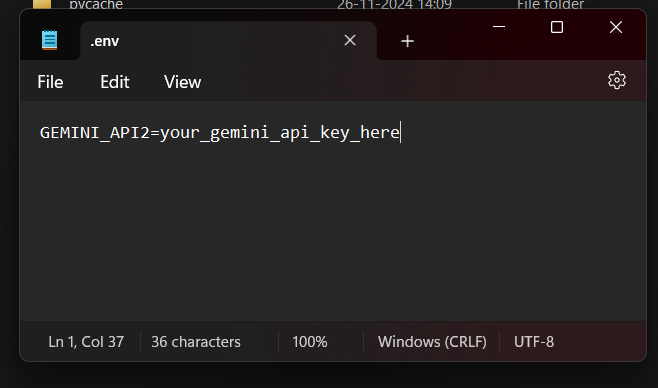
## Implementation

Our project's implementation seamlessly integrates modular functionality for intelligent file organization. The system's main application, developed in Python, acts as the central controller, executing individual modules for sorting, categorization, and duplicate removal. Users interact with the system through a command-line interface (CLI), providing the directory path for organization. Each module processes specific file types and contributes to a fully structured and user-friendly directory.

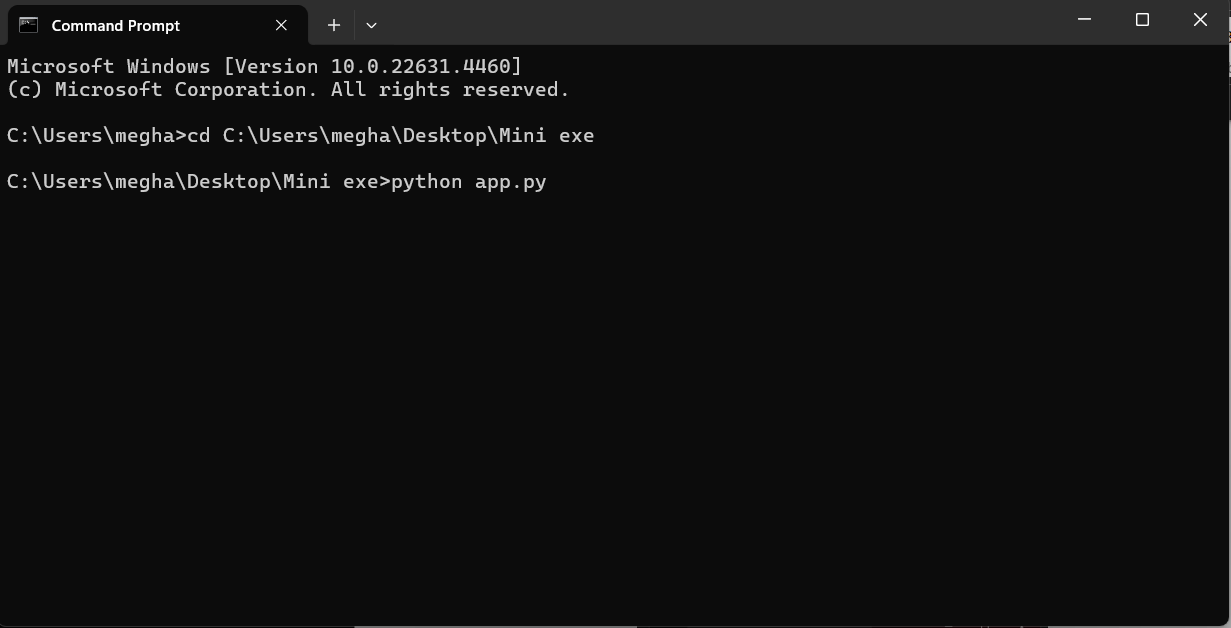
Key modules include:

* **Extension-Based Sorting**: Files are initially categorized into folders by type using the extension\_sorter module.
* **Audio File Organization**: The audio\_sorter module organizes music files by album metadata extracted with the Mutagen library.
* **Image Categorization**: The image\_sorter module leverages the YOLOv8 model for object detection, creating meaningful categories based on image content.
* **Document Categorization**: The doc\_sorter module utilizes text extraction libraries and AI-based analysis via the Google Gemini API to classify documents.
* **Duplicate File Removal**: The duplicate\_remover module detects and removes duplicate files using hash-based comparison.

To execute the application:

* + - Create an API key by signing up for the **Google Gemini API** via the Google Cloud Platform.

**Figure 4.1:** Saving API key in .env file

* + - Save the API key in a .env file located in the project directory
    - Run the main application script by providing the directory path:

**Figure 4.2:** Execution of the program

* + - Follow the prompts to complete the file organization process. Users can review duplicates and provide feedback on the results if desired.

## Results

### The directory before executing the program

**A screenshot of a computer

Description automatically generated**

**Figure 4.3:** Directory before execution

### The directory after executing the program

**A screenshot of a computer

Description automatically generated**

**Figure 4.4:** Directory after execution

All the files are sorted based on the type of extension they have. (i.e., audio, video, photos, codes, documents, and other for unidentified extensions).

### Sorted audio files

**A screenshot of a computer

Description automatically generated**

**Figure 4.5:** Sorted audio files

### Sorted image files

**A screenshot of a computer

Description automatically generated**

**Figure 4.6:** Sorted image files

### Sorted documents

**A screenshot of a computer

Description automatically generated**

**Figure 4.7:** Sorted documents

# CONCLUSION AND FUTURE SCOPE

## Conclusion

The **Smart File Organization System Using AI** successfully addresses the challenges of managing unstructured and cluttered digital files. By combining rule-based sorting techniques with advanced AI-driven categorization, the system provides an automated and scalable solution for organizing diverse file types, including audio files, images, documents, and more.

The integration of technologies like the YOLOv8 object detection model, Google Gemini API, and metadata extraction libraries ensures a robust and intelligent file organization process. Additionally, features like duplicate file detection and intelligent category refinement further enhance its functionality, reducing storage inefficiencies and improving user experience.

This project demonstrates the potential of artificial intelligence and automation in simplifying everyday digital tasks. It not only saves time and effort but also adapts dynamically to user needs, making it a versatile tool for modern file management. With scalability and modularity at its core, the system can be extended to support additional file types and advanced features in the future, ensuring its relevance in the evolving digital landscape.

## Future Scope

The **Smart File Organization System Using AI** provides a solid foundation for intelligent and automated file management. However, there are numerous opportunities to extend and enhance the system's capabilities. Expand the system to handle more complex file types, such as videos, CAD files, and compressed archives. Extend functionality to organize files in cloud storage platforms like Google Drive, OneDrive, and Dropbox. Develop a graphical user interface (GUI) or web-based dashboard for easier interaction and file management. Enable multi-language support for analyzing and categorizing documents in various languages. Introduce a learning mechanism where the system adapts to user preferences over time, such as frequently used categories or custom rules. Provide recommendations for archiving or deleting rarely accessed files to optimize storage usage and reduce digital clutter.

# BIBLIOGRAPHY

1. Stefano Ferilli, T. M. A. Basile, Marenglen Biba, and Floriana Esposito (2008). *Automatic Document Organization Exploiting FOL Similarity-based Techniques.* DOI: 10.1007/978-3-540-85458-0\_25.
2. Samual Kahsay (2016). *Automating the Organizing Process for PDF Files.* Tagspaces, kernel algorithms, and file metadata-based classification.
3. Syed Ali Raza, Sagheer Abbas, Taher M. Ghazal, Muhammad Adnan Khan, Munir Ahmad, and Hussam Al Hamadi (2022). *Content-Based Automated File Organization Using Machine Learning Approaches.* Computers, Materials & Continua, DOI: 10.32604/cmc.2022.029400.
4. Dobrica Savic (1995). *Automatic Classification of Office Documents: Review of Available Methods and Techniques*.
5. Tausif Diwan, G. Anirudh, Jitendra V. Tembhurne (2023). *Object Detection using YOLO: Challenges, Architectural Successors, Datasets, and Applications*. DOI: 10.1007/s11042-022-13644-y.

# APPENDIX

### app.py:

import os  # File and directory operations

'''import shutil  # High-level file operations

import re  # Regular expressions for text matching and manipulation

import hashlib  # Hashing for duplicate detection

# Libraries for audio metadata processing

from mutagen import File  # General mutagen operations

from mutagen.id3 import ID3NoHeaderError  # Specific to ID3 handling

# Libraries for handling documents

from PyPDF2 import PdfReader  # PDF reading and parsing

from docx import Document  # Word document (.docx) processing

# Google Generative AI

from dotenv import load\_dotenv  # Load environment variables from a .env file

import google.generativeai as genai  # Google Gemini API for AI-based processing

import ast  # Abstract syntax tree for safely evaluating expressions

# YOLO for object detection in images

from ultralytics import YOLO  # Object detection library'''

#codes of different sorter

import extension\_sorter

import audio\_sorter

import image\_sorter

import duplicates\_remover

import doc\_sorter

'''# Load environment variables from the .env file

load\_dotenv()

api\_key = os.getenv("GEMINI\_API2")

genai.configure(api\_key=api\_key)

model = genai.GenerativeModel(

    "gemini-1.5-flash",

    system\_instruction="You are a file organization system. You get files as input. You read the content of the file and return a category name as output such as letters, college mails, Subject notes, banking documents, etc."

)'''

def main(base\_directory):

    # Step 1: Sort all files by extensions

    extension\_sorter.sort\_files(base\_directory)

    # Step 2: Process audio files

    audio\_directory = os.path.join(base\_directory, "audio")

    if os.path.exists(audio\_directory):

        audio\_sorter.organize\_music\_in\_directory(audio\_directory)

    else:

        print(f"Audio directory not found: {audio\_directory}")

    # Step 3: Process photos

    photos\_directory = os.path.join(base\_directory, "Photos")

    if os.path.exists(photos\_directory):

        image\_sorter.organize\_by\_objects(photos\_directory)

    else:

        print(f"Photos directory not found: {photos\_directory}")

    # Step 4: Process documents

    documents\_directory = os.path.join(base\_directory, "Documents")

    if os.path.exists(documents\_directory):

        duplicates\_remover.remove\_dup(documents\_directory)

        doc\_sorter.sort\_doc(documents\_directory)

    else:

        print(f"Documents directory not found: {documents\_directory}")

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

    # Replace this with the path of the base directory

    base\_directory\_path = input("Enter the path to the base directory: ").strip()

    if os.path.exists(base\_directory\_path):

        main(base\_directory\_path)

    else:

        print(f"Invalid directory path: {base\_directory\_path}")

### extension\_sorter.py:

import os

import shutil

def sort\_files(directory):

  """Sorts files in a directory by their extension type and moves them into respective folders.

  Args:

      directory: The directory to sort.

  """

  # Create folders for different file types

  audio\_dir = os.path.join(directory, 'Audio')

  video\_dir = os.path.join(directory, 'Video')

  docs\_dir = os.path.join(directory, 'Documents')

  pictures\_dir = os.path.join(directory, 'Photos')

  codes\_dir = os.path.join(directory,'Codes')

  others\_dir = os.path.join(directory, 'others')

  os.makedirs(audio\_dir, exist\_ok=True)

  os.makedirs(video\_dir, exist\_ok=True)

  os.makedirs(docs\_dir, exist\_ok=True)

  os.makedirs(pictures\_dir, exist\_ok=True)

  os.makedirs(codes\_dir, exist\_ok=True)

  os.makedirs(others\_dir, exist\_ok=True)

  # Define file extensions for each category

  audio\_extensions = ['.mp3', '.wav', '.ogg', '.flac', '.aac', '.m4a']

  video\_extensions = ['.mp4', '.avi', '.mov', '.mkv', '.webm', '.wmv']

  docs\_extensions = ['.pdf', '.doc', '.docx', '.xls', '.xlsx', '.ppt', '.pptx', '.txt']

  codes\_extensions = [".py", ".java", ".c", ".cpp",".html"]

  pictures\_extensions = ['.jpg', '.jpeg', '.png', '.gif', '.bmp', '.svg']

  # Iterate through files in the directory

  for filename in os.listdir(directory):

    filepath = os.path.join(directory, filename)

    if os.path.isfile(filepath):

      # Get file extension

      file\_extension = os.path.splitext(filename)[1].lower()

      # Move file to appropriate folder

      if file\_extension in audio\_extensions:

        shutil.move(filepath, audio\_dir)

      elif file\_extension in video\_extensions:

        shutil.move(filepath, video\_dir)

      elif file\_extension in docs\_extensions:

        shutil.move(filepath, docs\_dir)

      elif file\_extension in pictures\_extensions:

        shutil.move(filepath, pictures\_dir)

      elif file\_extension in codes\_extensions:

        shutil.move(filepath, codes\_dir)

      else:

        shutil.move(filepath, others\_dir)

### audio\_sorter.py:

import os

import shutil

from mutagen import File

from mutagen.id3 import ID3NoHeaderError

import re

import os

import re

import shutil

from mutagen import File

from mutagen.id3 import ID3NoHeaderError

def sanitize\_folder\_name(name):

    """

    Remove or replace invalid characters in the folder name to make it valid for the filesystem.

    """

    return re.sub(r'[<>:"/\\|?\*]', '\_', name)

def get\_album\_info(audio\_file\_path):

    """

    Get the album information from the audio file's metadata.

    Supports various audio formats using the mutagen File class.

    """

    try:

        audio = File(audio\_file\_path, easy=True)

        if audio is None:

            album = 'Unknown Album'

        else:

            album = audio.get('album', ['Unknown Album'])[0]

    except ID3NoHeaderError:

        album = 'Unknown Album'

    return sanitize\_folder\_name(album)

def organize\_by\_album(audio\_file\_path, base\_directory):

    """

    Organize the audio file by album. If the album directory doesn't exist, create it.

    """

    album = get\_album\_info(audio\_file\_path)

    # Create the album directory if it doesn't exist

    album\_directory = os.path.join(base\_directory, album)

    if not os.path.exists(album\_directory):

        os.makedirs(album\_directory)

        print(f"Created directory: {album\_directory}")

    # Move the audio file to the album directory

    try:

        shutil.move(audio\_file\_path, album\_directory)

        print(f"Moved '{audio\_file\_path}' to '{album\_directory}'")

    except Exception as e:

        print(f"Error moving file: {e}")

def organize\_music\_in\_directory(directory):

    """

    Organize all audio files in the directory into album folders within the same directory.

    """

    supported\_formats = ('.mp3', '.flac', '.aac', '.m4a', '.wav', '.ogg', '.wma')  # Add more formats as needed

    for root, dirs, files in os.walk(directory):

        for file in files:

            if file.lower().endswith(supported\_formats):

                file\_path = os.path.join(root, file)

                organize\_by\_album(file\_path, directory)

'''if \_\_name\_\_ == '\_\_main\_\_':

    # Define the directory where the audio files are located

    directory = input("Enter the path to the directory containing audio files: ").strip()

    if os.path.exists(directory):

        # Organize the audio files into albums

        organize\_music\_in\_directory(directory)

    else:

        print(f"Invalid directory path: {directory}")'''

### image\_sorter.py:

from ultralytics import YOLO

import os

import shutil

# Load YOLO model

model = YOLO("yolov8l.pt")  # Load the YOLOv8 model

# Function to detect objects in an image

def detect\_objects(image\_path):

    results = model(image\_path)  # Run inference on the image

    detections = results[0].boxes  # Extract bounding boxes from the first image's results

    detected\_labels = []

    for box in detections:

        label = box.cls.item()  # Extract the class index as a number

        detected\_labels.append(model.names[int(label)])  # Map index to class name using model.names

    return detected\_labels

# Function to organize images based on detected objects

def organize\_by\_objects(directory):

    image\_paths = [os.path.join(directory, f) for f in os.listdir(directory)

                   if f.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.tiff'))]

    for path in image\_paths:

        try:

            objects = detect\_objects(path)  # Detect objects in the image

            if objects:

                for obj in objects:

                    obj\_dir = os.path.join(directory, obj)  # Ensure the object name is used as a string

                    os.makedirs(obj\_dir, exist\_ok=True)

                    shutil.move(path, os.path.join(obj\_dir, os.path.basename(path)))

                print(f"Processed {path}, detected: {', '.join(objects)}")

            else:

                print(f"No objects detected in {path}.")

        except Exception as e:

            print(f"Error processing {path}: {e}")

# Main execution

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

#    directory = "C:/Users/megha/Desktop/Download - Copy/picture"  # Update to your directory path

#    organize\_by\_objects(directory)

### duplicate\_remover.py:

import PyPDF2

import hashlib

import os

def check\_for\_duplicates(directory):

    duplicates = []

    file\_hashes = {}

    for filename in os.listdir(directory):

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

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

            file\_hash = hashlib.md5(f.read()).hexdigest()

        if file\_hash in file\_hashes:

            duplicates.append((file\_hashes[file\_hash], file\_path))

        else:

            file\_hashes[file\_hash] = file\_path

    return duplicates

def delete\_duplicates(duplicates):

    for duplicate\_pair in duplicates:

        print(f"Duplicate files: {duplicate\_pair[0]} and {duplicate\_pair[1]}")

    user\_input = input("Do you want to delete the duplicate files? (yes/no): ")

    if user\_input.lower() == 'yes':

        for duplicate\_pair in duplicates:

            try:

                os.remove(duplicate\_pair[1])

                print(f"Deleted: {duplicate\_pair[1]}")

            except OSError as e:

                print(f"Error deleting {duplicate\_pair[1]}: {e}")

# Example usage:

'''directory\_path = 'C:/Users/megha/Desktop/del on21-11-2024'  # Current directory'''

def remove\_dup(directory\_path):

    duplicates = check\_for\_duplicates(directory\_path)

    delete\_duplicates(duplicates)

**doc\_sorter.py**:

import os

from dotenv import load\_dotenv

import google.generativeai as genai

import ast

from PyPDF2 import PdfReader

from docx import Document

# Load environment variables from the .env file

load\_dotenv()

api\_key = os.getenv("GEMINI\_API2")

genai.configure(api\_key=api\_key)

model = genai.GenerativeModel(

    "gemini-1.5-flash",

    system\_instruction="You are a file organization system. You get files as input. You read the content of the file and return a category name as output such as letters, college mails, Subject notes, banking documents, etc."

)

#directory\_path = "C:/Users/megha/Desktop/Sample files - Copy/Documents"

initial\_categories = []

def extract\_text(filepath):

    """Extracts text from the first 3 pages or lines of a file."""

    text = ""

    if filepath.endswith('.pdf'):

        try:

            reader = PdfReader(filepath)

            for page in reader.pages[:3]:  # First 3 pages

                text += page.extract\_text()

        except Exception as e:

            print(f"Error reading PDF file {filepath}: {e}")

    elif filepath.endswith('.docx'):

        try:

            doc = Document(filepath)

            for paragraph in doc.paragraphs[:100]:  # First 100 lines

                text += paragraph.text

        except Exception as e:

            print(f"Error reading DOCX file {filepath}: {e}")

    elif filepath.endswith('.txt'):

        try:

            with open(filepath, 'r') as f:

                lines = f.readlines()[:100]  # First 100 lines

                text += "".join(lines)

        except Exception as e:

            print(f"Error reading TXT file {filepath}: {e}")

    return text.strip()

def sort\_doc(directory\_path):

    for filename in os.listdir(directory\_path):

        if filename.endswith(('.pdf', '.txt', '.docx')):  # Supported file types

            filepath = os.path.join(directory\_path, filename)

            content\_excerpt = extract\_text(filepath)

            if not content\_excerpt:

                category = "unidentified"

            else:

                # Generate category based on extracted text

                result = model.generate\_content([

                    content\_excerpt,

                    "\n\n",

                    '''1) Just read the document until wherever you want so that you could identify what is there inside the doc.

                    2) Return a category name (in 1-2 words) so that the document will be sorted into that category directory.

                    3) If you cannot process the content of the document, then return as unidentified.

                    4) Do not return any other things (it has to be a category name or else unidentified).'''

                ])

                category = result.text.strip()

            initial\_categories.append(category)

            print(f"Processed {filename}: {category}")

    print(initial\_categories)

    # Deduplicate and clean categories

    while 1:

        try:

            response = model.generate\_content(f'read all these categories, remove duplicates or make similar categories into a same name category. Rename the names if necessary.just return a python list of category names. Any other text is not required: {', '.join(initial\_categories)}.')

            # Use ast.literal\_eval to safely convert the string to a list

            final\_categories = ast.literal\_eval(response.text)

        except Exception:

            continue

        break

    #print(response)

    #Use ast.literal\_eval to safely convert the string to a list

    final\_categories = ast.literal\_eval(response.text)

    print("Final responses :",final\_categories)

    # Sorting docs into those categories

    categorized\_documents = {}

    for filename in os.listdir(directory\_path):

        if filename.endswith(('.pdf', '.txt', '.docx')):

            print("...")

            filepath = os.path.join(directory\_path, filename)

            content\_excerpt = extract\_text(filepath)

            if not content\_excerpt:

                category = "unidentified"

            else:

                # Get the category for the document

                result = model.generate\_content(

                    [

                        content\_excerpt,

                        "\n\n",

                        "Select one category name for the document from the given list only. Just return the category name: " + ', '.join(final\_categories),

                    ]

                )

                best\_category = result.text.strip()

            # Handle case where the category directory doesn't exist

            category\_path = os.path.join(directory\_path, best\_category)

            os.makedirs(category\_path, exist\_ok=True)  # Create directory if it doesn't exist

            # Move the file to its category directory

            destination\_path = os.path.join(category\_path, filename)

            os.rename(filepath, destination\_path)

            # Add to categorized documents dictionary

            if best\_category not in categorized\_documents:

                categorized\_documents[best\_category] = []

            categorized\_documents[best\_category].append(filepath)

    print("Categorization complete.")