INDUSTRIAL INTERNSHIP REPORT ON

FILE ORGERNISER

PREPARED BY

SUHANI S. KANOJE

Executive Summary

This report provides details of the Industrial Internship provided by upskill Campus and The IoT Academy in collaboration with Industrial Partner Uni Converge Technologies Pvt Ltd (UCT).

This internship was focused on a project/problem statement provided by UCT. We had to finish the project including the report in 6 weeks' time.

My project was "FILE ORGERNIZER", it is about how to arrange a files in serial manner. Certainly! In Python, we can organize and manage files using the os and shutil modules.

os Module:
The os module provides a way of interacting with the operating system. It includes functions to manipulate file paths, directories, and perform various operations related to file management.
Example: Listing files in a directory

shutil Module:
The shutil module provides a higher-level interface for file operations. It includes functions to copy, move, and delete files and directories.
Example: Moving a file to another directory

The os.path module provides functions for common path manipulations. It helps in constructing and manipulating file paths.Example: Joining paths

The os.makedirs function is used to create directories (and subdirectories) recursively.
Example: Creating a directory

Remember to handle exceptions, such as FileNotFoundError or PermissionError, when working with files and directories to ensure robustness in your code.

This internship gave me a very good opportunity to get exposure to Industrial problems and design/implement solution for that. It was an overall great experience to have this internship.

TABLE OF CONTENTS

1 PREFACE

Our journey into the world of file organization with Python begins with this comprehensive overview. From the conceptualization to the final deployment, the project is designed to optimize file management for users.

A file organizer in Python is a program or script designed to automatically categorize and arrange files within a directory structure based on specific criteria. This can help in maintaining a well-organized file system, making it easier to locate and manage files. The organization can be done based on various factors such as file type, date, size, or any other custom rules.

Here is a more detailed explanation of how you might create a file organizer in Python:

1. DEFINE THE CRITERIA:

File Type: Organize files based on their extensions (e.g., separate folders for images, documents, videos).

- Date: Group files into folders based on creation or modification dates.
- Size: Create folders for small, medium, and large files.
- Custom Rules: Define your own criteria based on file names, keywords, or any other attribute.

2. WALK THROUGH THE SOURCE DIRECTORY:

- Use the os.listdir() function to get a list of all files in the source directory.
- Iterate through the list and check if each item is a file (not a directory).

3. DEFINE DESTINATION FOLDERS:

- Based on the criteria you defined, create the destination folders if they don't already exist.
- For example, if organizing by file type, create folders for each file extension.

4. ORGANIZE FILES:

- Use the shutil.move() function to move files to their respective destination folders.
- The destination folder is determined based on the criteria you defined.

2 PREFACE

Our journey into the world of file organization with Python begins with this comprehensive overview. From the conceptualization to the final deployment, the project is designed to optimize file management for users.

A file organizer in Python is a program or script designed to automatically categorize and arrange files within a directory structure based on specific criteria. This can help in maintaining a well-organized file system, making it easier to locate and manage files. The organization can be done based on various factors such as file type, date, size, or any other custom rules.

Here is a more detailed explanation of how you might create a file organizer in Python:

1. DEFINE THE CRITERIA:

File Type: Organize files based on their extensions (e.g., separate folders for images, documents, videos).

- Date: Group files into folders based on creation or modification dates.
- Size: Create folders for small, medium, and large files.
- Custom Rules: Define your own criteria based on file names, keywords, or any other attribute.

2. WALK THROUGH THE SOURCE DIRECTORY:

- Use the os.listdir() function to get a list of all files in the source directory.
- Iterate through the list and check if each item is a file (not a directory).

3. DEFINE DESTINATION FOLDERS:

- Based on the criteria you defined, create the destination folders if they don't already exist.
- For example, if organizing by file type, create folders for each file extension.

4. ORGANIZE FILES:

- Use the shutil.move() function to move files to their respective destination folders.
- The destination folder is determined based on the criteria you defined.

5. EXCEPTION HANDLING:

- Implement error handling to manage cases where a file cannot be moved due to permission issues or other reasons.

PROGRESS IN WHOLE WEEK

Week 1: Understanding the Requirements and Planning

- Assessing Requirements
- Gathering and analyzing the needs and expectations for the file organizer software.
- Project Planning
- Creating a roadmap and high-level plan for the successful execution of the project.

Week 2: Designing the File Structure and Operations

- File Structure
- Architecting the hierarchical layout for efficient organization and storage.
- Basic Operations
- Implementing fundamental file operations like create, open, and delete.

Week 3: Adding Advanced Features

- File Filtering
- Including the ability to apply specific filters for sorting files.
- Sorting Mechanism

Week 4: Error Handling and Project Testing

- Rigorous Testing
- Thoroughly testing the software to identify and rectify any potential weaknesses.
- Error Resolution
- Implementing strategies to handle errors and exceptions gracefully.

Week 5: Refactoring and Performance Optimization

- Code Refactoring
- Restructuring and optimizing the existing codebase for enhanced performance.
- Performance Tweaking
- Identifying and enhancing key performance metrics within the software.

Week 6: Final Testing and Deployment

- Final Testing
- Ensuring error-free functionality with comprehensive final testing.
- Software Deployment
- Deploying the optimized file organizer software for user access

……THANK TO MY WHOLE FRIEND CIRCLE (ASHU ,PRAGATI , TASHU), WHO HELPED ME DIRECTLY OR INDIRECTLY.

…….. I DON'T THINK THAT THIS REPORT IS COMPLETE WITHOUGHT HELPING MY FRIENDS…THANK YOU ALL……..

INTRODUCTION

2.1 ABOUT UNICONVERGE TECHNOLOGIES PVT LTD

A company established in 2013 and working in Digital Transformation domain and providing Industrial solutions with prime focus on sustainability and RoI.

For developing its products and solutions it is leveraging various **Cutting Edge Technologies e.g. Internet of Things (IoT), Cyber Security, Cloud computing (AWS, Azure), Machine Learning, Communication Technologies (4G/5G/LoRaWAN), Java Full Stack, Python, Front end** etc.

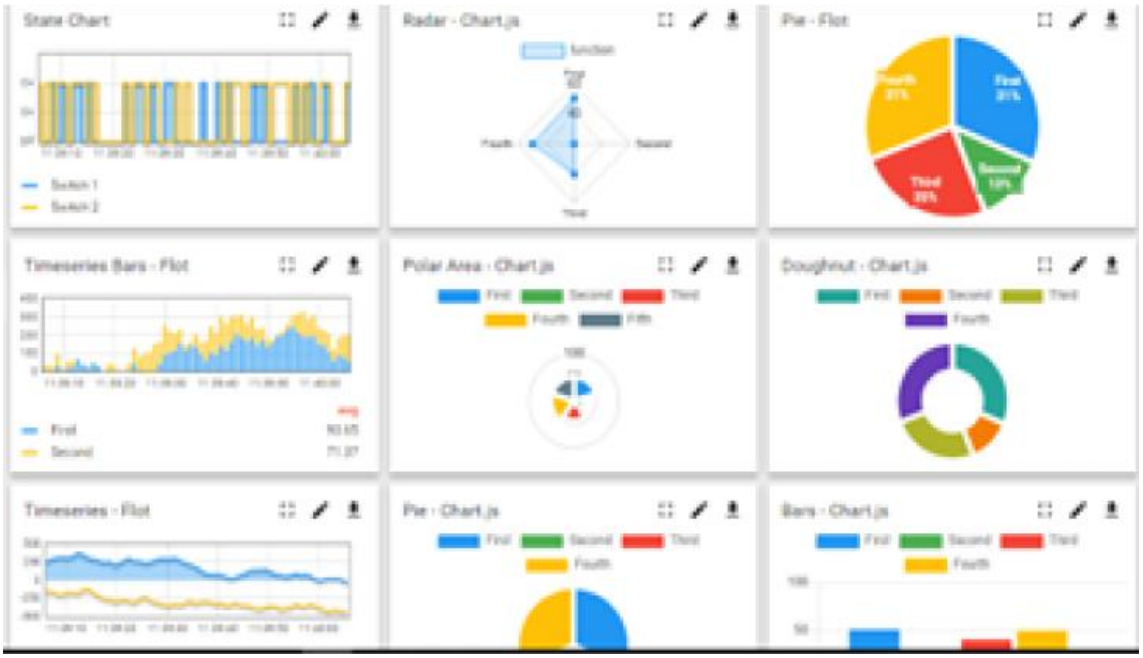
i. UCT IOT PLATFORM (uct Insight)

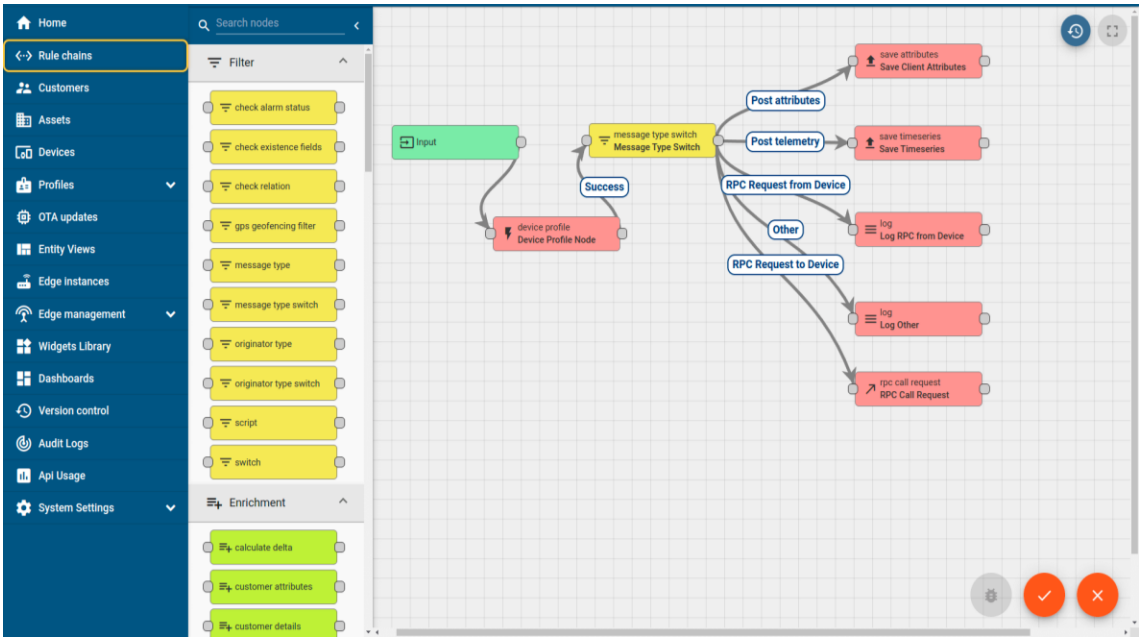
UCT Insight is an IOT platform designed for quick deployment of IOT applications on the same time providing valuable "insight" for your process/business. It has been built in Java for backend and ReactJS for Front end. It has support for MySQL and various NoSql Databases.

- It enables device connectivity via industry standard IoT protocols - MQTT, CoAP, HTTP, Modbus TCP, OPC UA
- It supports both cloud and on-premises deployments.

- It has features to
  - Build Your own dashboard

- Analytics and Reporting
- Alert and Notification
- Integration with third party application(Power BI, SAP, ERP)
- Rule Engine





ii. SMART FACTORY PLATFORM (FACTORY WATCH)

Factory watch is a platform for smart factory needs.

It provides Users/ Factory

- with a scalable solution for their Production and asset monitoring
- OEE and predictive maintenance solution scaling up to digital twin for your assets.
- to unleased the true potential of the data that their machines are generating and helps to identify the KPIs and also improve them.
- A modular architecture that allows users to choose the service that they what to start and then can scale to more complex solutions as per their demands.

Its unique SaaS model helps users to save time, cost and money.

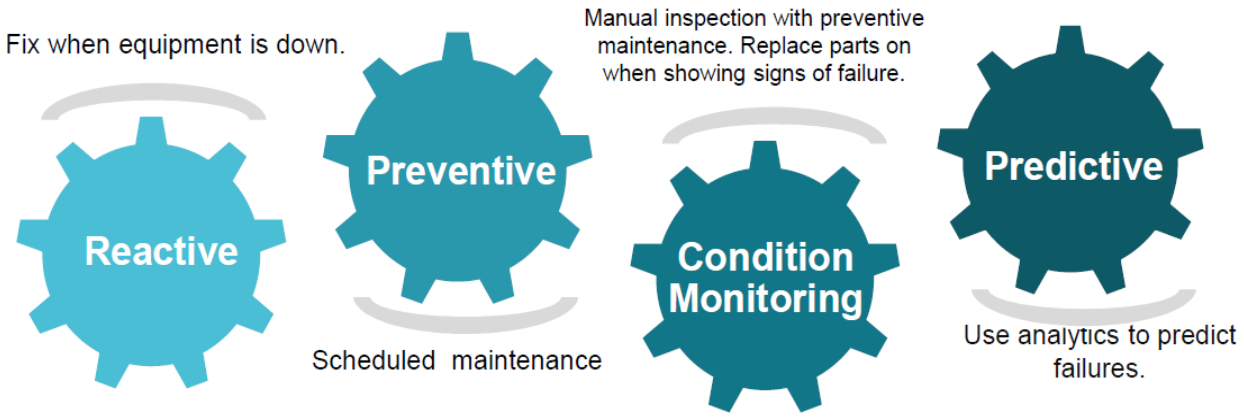
Machine	Operator	Work Order ID	Job ID	Job Performance	Job Progress		Output		Rejection	Time (mins)				Job Status	End Customer
					Start Time	End Time	Planned	Actual		Setup	Pred	Downtime	Idle		
CNC_S7_81	Operator 1	WO0405200001	4168	58%	10:30 AM		55	41	0	80	215	0	45	In Progress	i
CNC_S7_81	Operator 1	WO0405200001	4168	58%	10:30 AM		55	41	0	80	215	0	45	In Progress	i

iii. LoRaWAN BASED SOLUTION

UCT is one of the early adopters of Lo RAWAN teschnology and providing solution in Agritech, Smart cities, Industrial Monitoring, Smart Street Light, Smart Water/ Gas/ Electricity metering solutions etc.

iv. PREDICTIVE MAINTENANCE

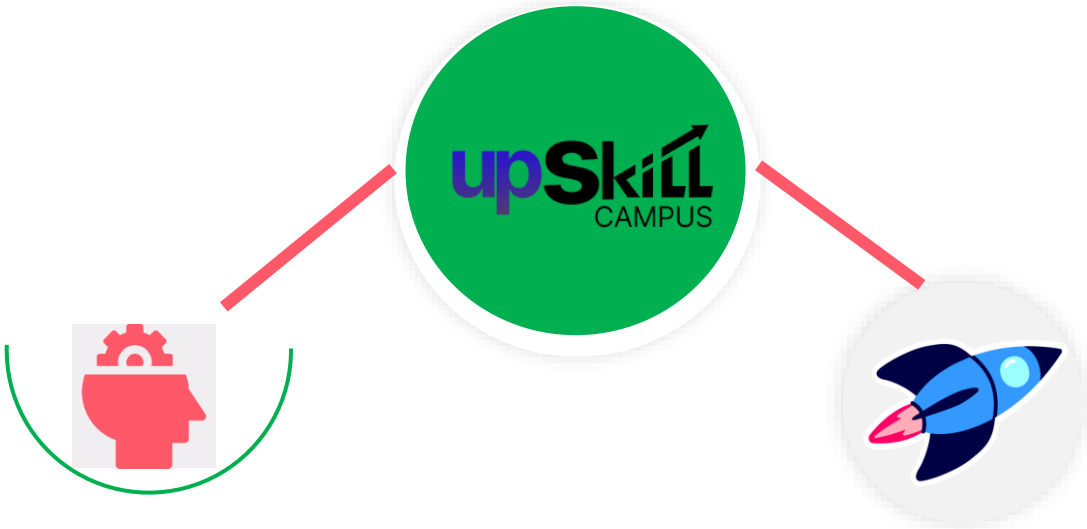
UCT is providing Industrial Machine health monitoring and Predictive maintenance solution leveraging Embedded system, Industrial IoT and Machine Learning Technologies by finding Remaining useful life time of various Machines used in production process.



2.2 ABOUT UPSKILL CAMPUS (USC)

upskill Campus along with The IoT Academy and in association with Uniconverge technologies has facilitated the smooth execution of the complete internship process.

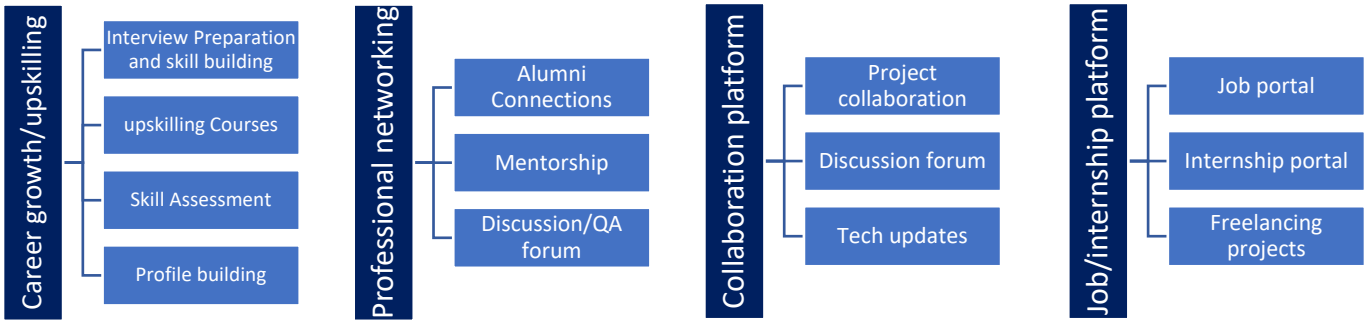
USC is a career development platform that delivers **personalized executive coaching** in a more affordable, scalable and measurable way.



Seeing need of upskilling in self paced manner along-with additional support services e.g. Internship, projects, interaction with Industry experts, Career growth Services

upSkill Campus aiming to upskill 1 million learners in next 5 year

https://www.upskillcampus.com/



2.3 The IoT Academy

The IoT academy is EdTech Division of UCT that is running long executive certification programs in collaboration with EICT Academy, IITK, IITR and IITG in multiple domains.

2.4 Objectives of this Internship program

The objective for this internship program was to

- ☛ get practical experience of working in the industry.
- ☛ to solve real world problems.
- ☛ to have improved job prospects.
- ☛ to have Improved understanding of our field and its applications.
- ☛ to have Personal growth like better communication and problem solving.

2.5 Reference

[1] https://www.python-engineer.com/posts/file-organization

[2] https://www.geeksforgeeks.org/junk-file-organizer-python

[3] https://github.com/topics/file-organizer?l=python

[4] https://blog.devgenius.io/python-for-beginners-how-to-write-a-simple-file-organizer-code-

PROBLEM STATEMENT

- Error Handling and Project Testing
- Rigorous Testing
- Thoroughly testing the software to identify and rectify any potential weaknesses.
- Error Resolution
- Implementing strategies to handle errors and exceptions gracefully.

EXISTING AND PROPOSED SOLUTION

- Problem: Incomplete or Incorrect File Moves/Copies:
- Existing Solution: Verify the success of file moves/copies by checking return values or using try-except blocks. Log errors and failures for further investigation.

- Problem: Lack of Source or Destination Validation:
- Existing Solution: Validate the existence of source and destination directories before proceeding with file operations. Use os.path.exists() or similar functions.

- Problem: Overwriting Existing Files Without Confirmation:
- Existing Solution: Add a check to confirm before overwriting existing files. Utilize os.path.exists() or os.path.isfile().

PROPOSED SOLUTIONS TO ADDRESS POTENTIAL ISSUES:

- Issue: Lack of Backup Mechanism:
- Proposed Solution: Implement a backup mechanism before making changes. This could involve creating a copy of the source directory or using version control for more advanced projects.

- Issue: No Confirmation for Large File Operations:
- Proposed Solution: For large-scale operations, provide a confirmation prompt to the user before proceeding. This can prevent accidental or undesired file organization.

- Issue: Limited User Feedback:
- Proposed Solution: Enhance user feedback by incorporating progress indicators, completion messages, and detailed logging for both successful and unsuccessful operations.

- Issue: Hardcoded Paths and Lack of Configurability:
- Proposed Solution: Allow users to configure source and destination paths as well as organization criteria through configuration files or command-line arguments.

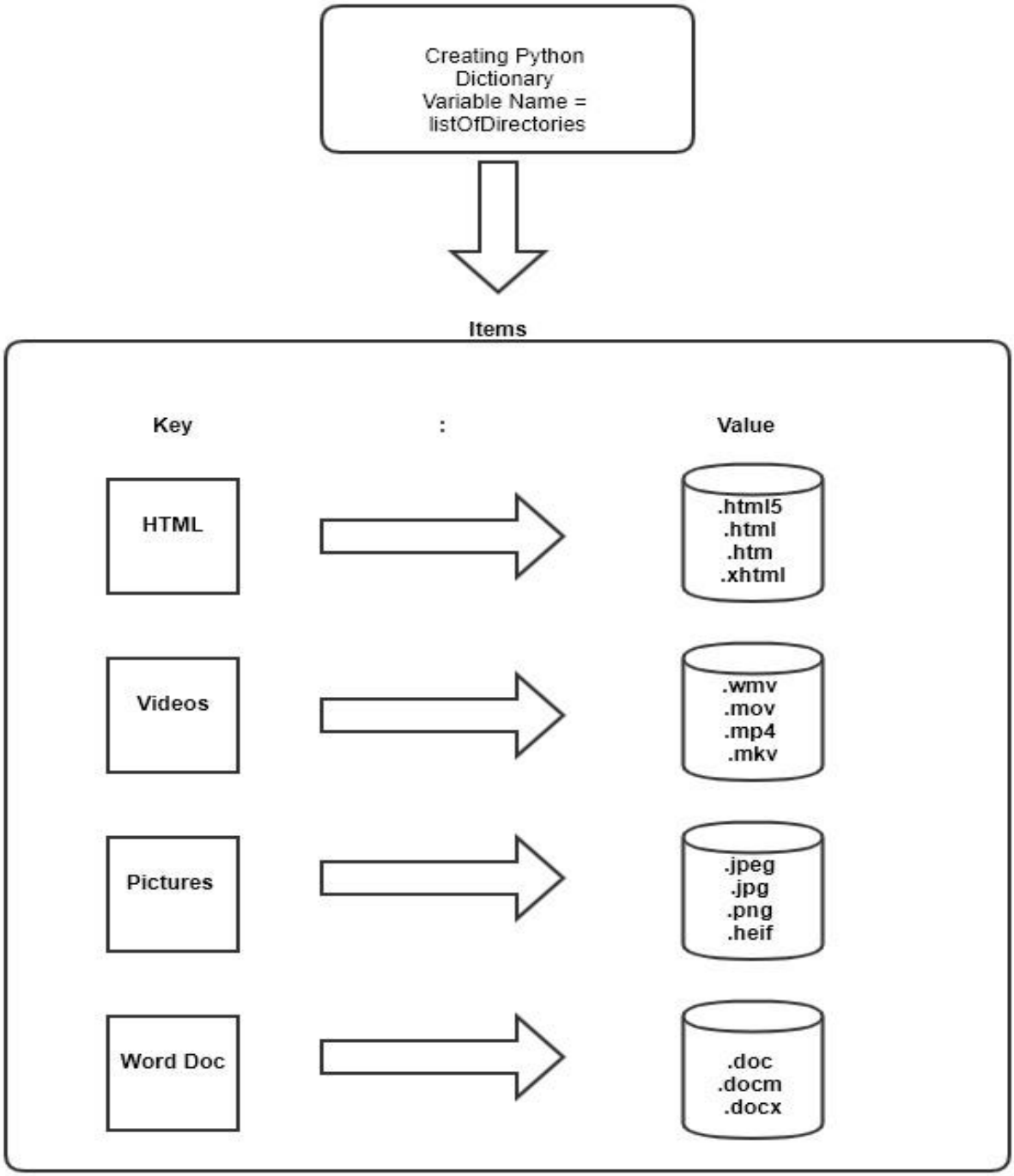
2.1 Code submission (Github link)

https://github.com/ssuhanikanoje/Upskillcampus/blob/main/fileOrgenizerSystem.py

2.2 Report submission (Github link) :

https://github.com/ssuhanikanoje/Upskillcampus/blob/main/fileOrganiser_SUHANI_USC_UCT.pdf

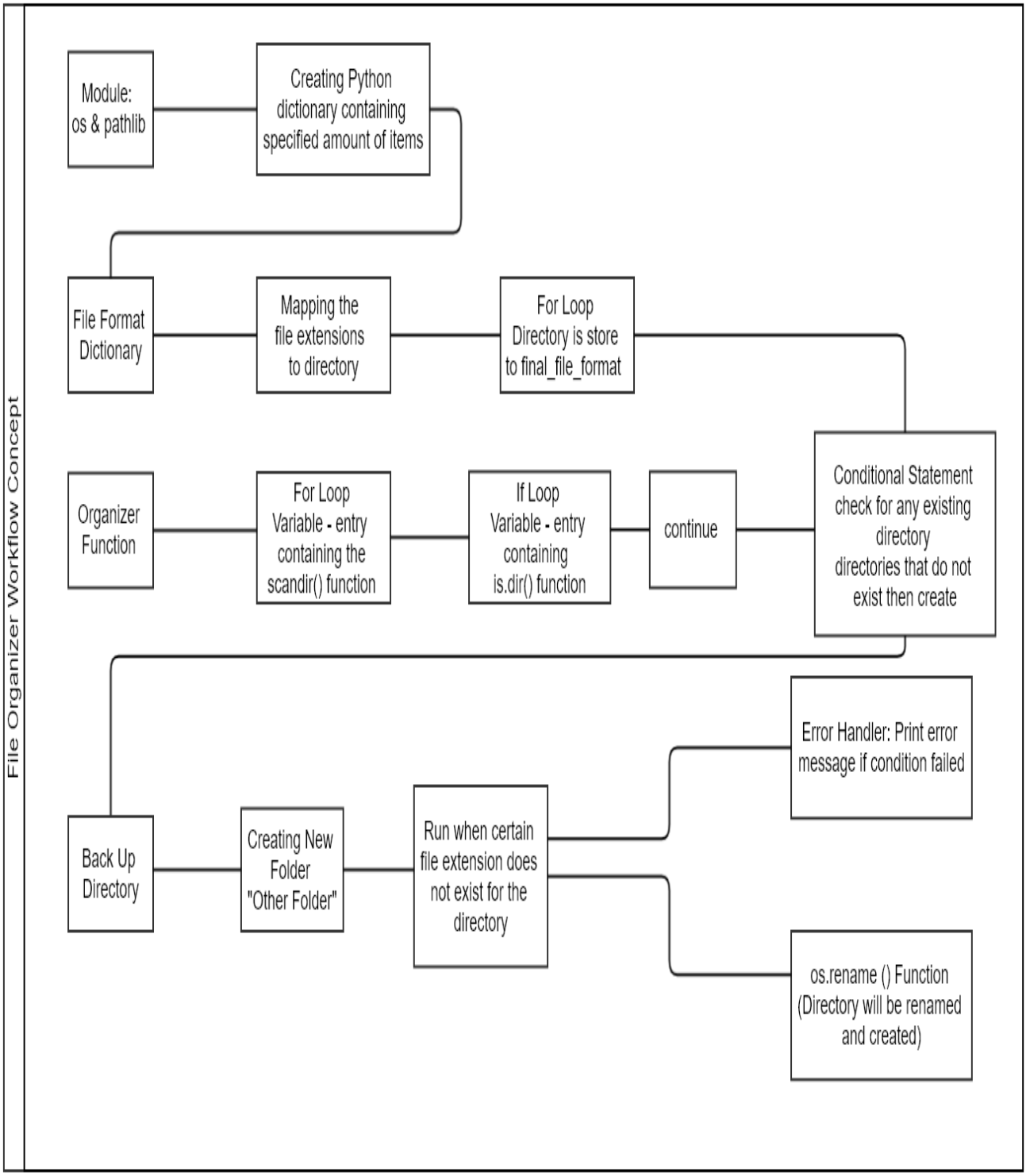
PROPOSED DESIGN/ MODEL



3.1 High Level Diagram (if applicable)

Figure 1: HIGH LEVEL DIAGRAM OF THE SYSTEM

2.2 Low Level Diagram (if applicable)



PERFORMANCE TEST

This is very important part and defines why this work is meant of Real industries, instead of being just academic project.

Here we need to first find the constraints.

How those constraints were taken care in your design?

What were test results around those constraints?

Constraints can be e.g. memory, MIPS (speed, operations per second), accuracy, durability, power consumption etc.

In case you could not test them, but still you should mention how identified constraints can impact your design, and what are recommendations to handle them.

4.1TEST PLAN/ TEST CASES

Organizing Files in a Directory:

Input: Source directory containing files.

Action: Run the file organizer to organize files.

Expected Output: Files are moved to their respective folders based on file types or categories.

Organizing Empty Directory:

Input: Empty directory

Action: Run the file organizer on an empty directory.

Expected Output: No changes, the directory remains empty.

Organizing Files in Nested Directories:

Input: Directory with nested subdirectories and files.

Action: Run the file organizer on the directory.

Expected Output: Files in all subdirectories are organized properly.

Handling Unsupported File Types:

Input: Directory with unsupported file types.

Action: Run the file organizer.

Expected Output: Unsupported files are identified and logged or skipped.

Handling Empty File Types:

Input: Directory with files of unknown types (no file extensions).

Action: Run the file organizer.

Expected Output: Files are either ignored or moved to a default folder.

Handling Duplicate Files:

Input: Directory with duplicate files.

Action: Run the file organizer.

Expected Output: Duplicate files are identified, and a strategy (e.g., renaming) is applied.

Undo Operation - Rollback Changes:

Input: Directory after files have been organized.

Action: Run the undo operation.

Expected Output: Directory is reverted to the state before the organization.

Handling Large Number of Files:

Input: Directory with a large number of files.

Action: Run the file organizer.

Expected Output: Organizer can handle a large number of files efficiently.

Handling File Organization Rules:

Input: Custom file organization rules (e.g., based on keywords in file names).

Action: Run the file organizer.

Expected Output: Files are organized according to the specified rules.

Handling File Permissions:

Input: Directory with files of different permissions.

Action: Run the file organizer.

Expected Output: File permissions are maintained or adjusted as needed during organization.

Handling Read-Only Files:

Input: Directory with read-only files.

Action: Run the file organizer.

Expected Output: Ability to organize files even if some are set to read-only.

These test cases cover various aspects of a file organizer module, including its ability to handle different file types, directory structures, errors, and undo operations. Adjust the test cases based on the specific features and requirements of your file organizer module.

4.3TEST PROCEDURE

1. Setup:

Set up a test environment with a dedicated directory for testing.

Populate the test directory with a variety of files, including different file types, nested directories, and some duplicates.

2. Pre-Test Conditions:

Verify that the file organizer module is properly installed.

Ensure the test directory is correctly set up.

3. Positive Test Cases:

a. Organizing Files:

- Input: Test directory with various files.

- Action: Run the file organizer on the test directory.

- Expected Result: Files are organized into appropriate folders based on their types.

b. Nested Directories:

- Input: Test directory with nested subdirectories.

- Action: Run the file organizer on the test directory.

- Expected Result: All files, including those in nested directories, are organized correctly.

c. Undo Operation:

- Input: Test directory after organizing files.

- Action: Run the undo operation.

- Expected Result: Directory is reverted to its initial state before the organization.

4. Negative Test Cases:

a. Unsupported File Types:

- Input: Test directory with unsupported file types.

- Action: Run the file organizer.

- Expected Result: Unsupported files are identified, and the organizer handles them appropriately.

b. Handling Duplicate Files:

- Input: Test directory with duplicate files.

- Action: Run the file organizer.

- Expected Result: Duplicate files are identified, and the organizer applies a strategy (e.g., renaming).

c. Empty Directory:

- Input: Empty test directory.

- Action: Run the file organizer.

- Expected Result: No changes to the directory.

5. CUSTOMIZATION TEST CASES:

a. Custom Organization Rules:

- Input: Test directory with custom organization rules.

- Action: Run the file organizer.

- Expected Result: Files are organized according to the specified rules.

b. Handling Read-Only Files:

- Input: Test directory with read-only files.

- Action: Run the file organizer.

- Expected Result: The organizer can handle read-only files, and their permissions are maintained or adjusted.

6. PERFORMANCE TEST:

Input: Test directory with a large number of files.

Action: Measure the time taken to organize files.

Expected Result: The organizer efficiently handles a large number of files within an acceptable time frame.

7. ERROR HANDLING:

a. Invalid Input:

- Input: Incorrect input directory path.

- Action: Run the file organizer.

- Expected Result: Appropriate error message is displayed.

b. Undo Without Changes:

- Input: Test directory without any organization changes.

- Action: Run the undo operation.

- Expected Result: No changes occur, and an appropriate message is displayed.

8. Post-Test Conditions:

Verify that the test directory and its contents are in the expected state after all tests.

Ensure the file organizer module does not leave any artifacts or unintended changes.

9. Documentation:

Document any issues encountered, including error messages.

Record the time taken for the performance test.

10. Clean-Up:

Remove any temporary files or artifacts created during testing.

Restore the test environment to its original state.

4.5 PERFORMANCE OUTCOME

1. Execution Time:

Goal: Ensure the file organizer can process files efficiently, even with a large number of files.

Metric: Measure the time taken to organize a directory with a substantial number of files.

Acceptance Criteria: The organizer completes the organization within a reasonable time frame.

2. Scalability:

Goal: Assess how well the file organizer scales with an increasing number of files.

Metric: Gradually increase the number of files in the test directory and observe the impact on performance.

Acceptance Criteria: The organizer maintains acceptable performance as the number of files increases.

3. Resource Usage:

Goal: Ensure the file organizer uses system resources efficiently.

Metric: Monitor CPU and memory usage during file organization.

Acceptance Criteria: Resource usage remains within acceptable limits, avoiding excessive CPU usage or memory leaks.

4. Handling Large Files:

Goal: Evaluate how well the organizer handles large files.

Metric: Test the organizer with files of varying sizes, including very large files.

Acceptance Criteria: The organizer efficiently processes large files without significant performance degradation.

5. Undo Operation Performance:

Goal: Assess the speed of the undo operation.

Metric: Measure the time taken to undo file organization changes.

Acceptance Criteria: The undo operation is performed quickly, regardless of the number of changes made.

6. Error Handling and Recovery:

Goal: Evaluate the performance of error handling and recovery mechanisms.

Metric: Introduce errors (e.g., unsupported file types) and measure how quickly the organizer identifies and recovers from them.

Acceptance Criteria: The organizer promptly identifies errors and recovers without significant impact on overall performance.

MY LEARNINGS

File Handling in Python:
Acquiring knowledge on how to work with files in Python, including reading, writing, moving, and deleting files.

Directory and File Organization:
Understanding different strategies for organizing directories and files based on types, categories, or custom rules.

Algorithm Design for Organization:
Developing algorithms to efficiently categorize and organize files based on specified criteria, considering factors such as file types, names, and sizes.

Error Handling and Logging:
Implementing robust error-handling mechanisms to deal with unexpected situations, and incorporating logging to track the execution flow and potential issues.

Undo Mechanism Implementation:
Creating a mechanism to undo file organization changes, providing users with a way to revert to the previous state.

Performance Optimization:
Understanding the importance of optimizing code for performance, especially when dealing with a large number of files or complex organization rules.

Scalability Considerations:
Learning how to design a file organizer that can scale effectively as the number of files or the complexity of organization rules increases.

Customization and Flexibility:

Implementing features that allow users to customize organization rules, providing flexibility for different use cases.

User Interface (UI) Design (if applicable):

If your project includes a user interface, you may have gained experience in designing a user-friendly and intuitive interface for users to interact with the file organizer.

Testing Strategies:

Developing and executing comprehensive test cases to ensure the file organizer functions correctly under various scenarios, including positive and negative cases, performance testing, and edge cases.

FUTURE WORK SCOPE

The future work scope for a file organizer in a Python project can involve expanding features, improving performance, enhancing user experience, and addressing potential limitations. Here are some ideas for future work on a file organizer:

Cloud Integration:

Extend the file organizer to support cloud storage platforms (e.g., Dropbox, Google Drive). Allow users to organize and manage files both locally and in the cloud.

Automated Organization Rules:

Implement more advanced and automated organization rules, such as machine learning-based categorization or natural language processing for smarter file classification.

File Versioning:

Introduce versioning capabilities to track changes made to files over time. This can be particularly useful for collaborative projects.

Search and Filtering:

Add a search functionality and advanced filtering options to help users quickly locate specific files or types of files within the organized structure.

Integration with Task Management:

Integrate the file organizer with task management features, allowing users to associate files with specific tasks or projects.

Mobile Application:

Develop a mobile application for the file organizer, enabling users to manage and organize files on the go.