**Industrial Internship Report on**

**”Python”**

**Prepared by**

**[Vishvam sanghadiya]**

|  |
| --- |
| *Executive Summary* |
| This report provides details of the Industrial Internship provided by upskill Campus and The IoT Academy in collaboration with Industrial Partner UniConverge 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 (Tell about ur Project)  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 3](#_Toc139702806)

[2 Introduction 4](#_Toc139702807)

[2.1 About UniConverge Technologies Pvt Ltd 4](#_Toc139702808)

[2.2 About upskill Campus 8](#_Toc139702809)

[2.3 Objective 9](#_Toc139702810)

[2.4 Reference 9](#_Toc139702811)

[2.5 Glossary 10](#_Toc139702812)

[3 Problem Statement 11](#_Toc139702813)

[4 Existing and Proposed solution 12](#_Toc139702814)

[5 My learnings 15](#_Toc139702823)

# Preface

Summary of the whole 6 weeks’ work.

About need of relevant Internship in career development.

About my project : Propose a pseudo code for the solution with all needed libraries and components needed.

I have done writing codes for 2 projects named: URL shortener and File organizer .

How Program was planned



I have learned soo many things during this 4 weeks period especially the libraries like Flask, Flask-SQL alchemy and pyshortener in url shortener and in File organizer OS and Shutil module. Also I make a good level of command in this language but I’m still in a learning phase.

My message to my juniors and peers is that make a basics strong practice basic level of problem multiples times by using different technique every time and also learn every other language not just py like its your feeling it while writing you will enjoy the the whole process.

# Introduction

## 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.



1. UCT IoT Platform **(****)**

**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

1. **Smart Factory Platform (****)**

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.

1.  based Solution

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

1. 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.



## 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.

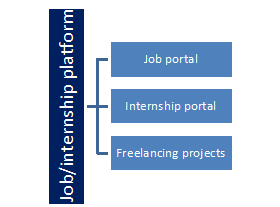
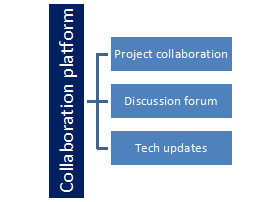
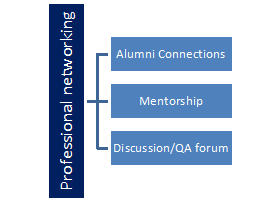
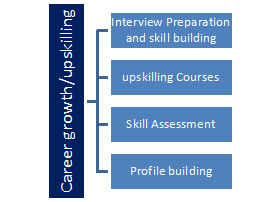
USC is a career development platform that delivers **personalized executive coaching** in a more affordable, scalable and measurable way.



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

<https://www.upskillcampus.com/>

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



## 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.

## 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.

# Problem Statement

Proposed a design or pseudo code for the solution with all needed libraries , components ,etc.

**URL Shortener Project:**

**Problem Statement:** The URL shortener project aims to create a service that converts long URLs into shorter, more manageable links. This is particularly useful for sharing links in a concise manner, especially in scenarios where character count is limited, such as social media posts or SMS messages.

**Design and Components:**

**Frontend Interface**: The project includes a user-friendly frontend interface where users can input a long URL and receive a shortened version.

**Backend Service:** The backend service is responsible for generating unique short URLs for each input, storing the mapping between the short and long URLs, and redirecting users to the original URL when they access the shortened link.

**Database:** To store the mapping between short and long URLs, a database is utilized. Common choices include relational databases like MySQL or NoSQL databases like MongoDB.

**Shortening Algorithm:** A robust algorithm is employed to generate short URLs that are unique and do not collide. This algorithm is designed to be efficient and produce short codes.

**APIs**: The project may expose APIs to allow integration with other services, enabling the automated generation of short URLs.

**Analytics:** Optionally, the system can incorporate analytics to track the usage of short URLs, providing insights into user behavior.

**Libraries and Technologies:**

**Backend Framework:** Flask , Express (Node.js), or similar frameworks.

**Database: M**ySQL, MongoDB, or another database as per the project requirements.

**Shortening Algorithm:** Custom algorithm or using libraries like Hashids.

**File Organizer Project:**

**Problem Statement:** The file organizer project is designed to automate the process of organizing files within a directory. This is particularly useful for maintaining a clean and structured file system, making it easier to locate and manage files.

**Design and Components**

**File Watcher**: The project includes a file watcher component that monitors a specified directory for any changes, such as the addition of new files.

**Organizer Module:** Upon detecting changes, an organizer module processes the new files and organizes them based on predefined rules. For example, files could be categorized by type (documents, images, videos) or date.

**Configuration:** Users can configure the rules for file organization according to their preferences. This could be done through a configuration file or a user-friendly interface.

**Libraries and Technologies:**

**File Watcher:** Watchdog library (Python), chokidar (Node.js).

**Organizer Logic:** Custom logic or libraries like shutil (Python) for file operations.

**Logging:** Built-in logging libraries or frameworks.

**Configuration:** YAML, JSON files or a configuration library.

# Existing and Proposed solution

What is your proposed solution?

**For url shortener:**

The code starts by importing the pyshorteners library. This library provides an easy-to-use interface for interacting with various URL shortening services.

**Function Definition (shorten\_url\_with\_bitly):**

This function takes two parameters: original\_url (the URL to be shortened) and bitly\_token (your Bitly access token).

**Documentation String (""" ... """):**

The triple-quoted string provides a docstring, which serves as documentation for the function. It explains the purpose of the function, its parameters, and the return value.

**Creating a Bitly Object:**

s = pyshorteners.Shortener(api\_key=bitly\_token): This line creates an instance of the Shortener class from pyshorteners and initializes it with your Bitly access token.

**Shortening the URL:**

shortened\_url = s.bitly.short(original\_url): This line uses the bitly attribute of the Shortener object to shorten the provided original\_url using Bitly.

**Return Statement:**

return shortened\_url: The function returns the shortened URL.

**Function Definition (shorten\_url\_with\_tinyurl):**

Similar to the Bitly version, this function takes one parameter: original\_url.

**Creating a TinyURL Object:**

s = pyshorteners.Shortener(): This line creates an instance of the Shortener class without any specific service specified, making it default to TinyURL.

**Shortening the URL:**

shortened\_url = s.tinyurl.short(original\_url): This line uses the tinyurl attribute of the Shortener object to shorten the provided original\_url using TinyURL.

***Return Statement:***

return shortened\_url: The function returns the shortened URL

**Example Bitly URL Shortening:**

original\_url\_bitly = "https://www.example.com/long-url": An example long URL is defined.

shortened\_url\_bitly = shorten\_url\_with\_bitly(original\_url\_bitly, bitly\_token): The function is called to get the Bitly shortened URL using the provided example URL and Bitly access token.

**Example TinyURL Shortening:**

original\_url\_tinyurl = "https://www.example.com/long-url": Another example long URL is defined.

shortened\_url\_tinyurl = shorten\_url\_with\_tinyurl(original\_url\_tinyurl): The function is called to get the TinyURL shortened URL using the provided example URL.

**Displaying Results:**

print("Bitly Shortened URL:", shortened\_url\_bitly): The Bitly shortened URL is printed.

print("TinyURL Shortened URL:", shortened\_url\_tinyurl): The TinyURL shortened URL is printed.

**For file organizer:**

**1. Importing Modules:**

os: This module provides a way of interacting with the operating system, allowing the script to perform various file and directory operations.

shutil: Short for shell utility, this module provides a higher-level interface for file operations and is used here for file movement (shutil.move).

**2. Organize Function:**

organize\_files: This is a function that takes a folder\_path as an argument and organizes files within that folder based on their file extensions.

file\_types: A dictionary that maps file types (Images, Documents, Videos, Music, Others) to lists of corresponding file extensions. Files with extensions not listed will be categorized as "Others."

**3. Creating Directories:**

This loop iterates through the keys of the file\_types dictionary and creates directories for each file type within the specified folder\_path if they don't already exist.

python

**4. Iterating Through Files:**

This loop iterates through each file in the specified folder (folder\_path) using os.listdir().

It checks if the item is a file (os.path.isfile()), and if so, it extracts the file extension.

**5. Determine Destination Directory:**

This section determines the appropriate destination directory (destination\_folder) based on the file extension. It checks the file\_types dictionary and assigns the appropriate folder name. If the extension is not found, it defaults to "Others."

**6. Moving Files:**

The script then uses shutil.move() to move the file from its current location (file\_path) to the calculated destination path (destination\_path).

A message is printed indicating the file has been moved.

**7. Main Execution:**

This block ensures that the organize\_files function is called only when the script is run directly (not imported as a module).

Replace "/path/to/your/folder" with the actual path of the folder you want to organize. This is the entry point of the script.

What value addition are you planning?

**URL shortener:**

To enhance the URL shortener code, I consider adding the following value-added features:

1. **User Authentication:** Implement user authentication to allow registered users to track and manage their shortened URLs. This can be achieved using Flask-Login or a similar authentication library.
2. ***URL Analytics:*** Extend the code to collect and display analytics for shortened URLs, such as the number of clicks, geographic location of users, and referral sources. This can provide valuable insights into the popularity and effectiveness of shared links.
3. ***Custom Short URLs:*** Allow users to customize their short URLs. This feature could involve providing users with the ability to choose a custom alias for their shortened link.
4. ***URL Expiry and Deactivation:*** Add functionality to set expiration dates for shortened URLs. Additionally, allow users to deactivate or delete their shortened URLs when they are no longer needed.
5. ***QR Code Generation:*** Integrate QR code generation for shortened URLs. Users can easily share URLs by scanning QR codes, enhancing usability in offline contexts.

**File organizer:**

Instead of just printing messages to the console, you can use the logging module to log information to a file. This makes it easier to track and review the history of file organization.

1. **Error Handling:** Implement error handling to manage exceptions that may occur during file operations. For example, handle cases where a file cannot be moved.
2. **User Input :** Allow the user to input the folder path interactively instead of hardcoding it in the script. You can use the input() function for this.
3. **File Type Extension Validation:** Add validation to ensure that file extensions in the file\_types dictionary are prefixed with a dot (e.g., ".jpg"). This can prevent potential bugs.
4. **Configurability:** Make the file type dictionary and folder paths configurable, either through command-line arguments, a configuration file, or environment variables.
5. **File Count Display:bb**After organizing files, display the count of files moved to each category. This gives the user a summary of the organization process.

## Code submission (Github link)

https://github.com/Vishvam248/Python\_report.git

## Report submission (Github link) : first make placeholder, copy the link.

# Performance Test

URL shortener:

**Constraints:**

**Lack of Error Handling:**

The code lacks comprehensive error handling. For instance, if the Bitly access token is incorrect or the URL shortening service is unavailable, the code may not handle these situations gracefully.

**Security Risks:**

The code does not implement extensive security measures, leaving it vulnerable to various threats, including input validation issues and potential attacks.

**No Expiration or Deactivation:**

There is no mechanism for setting an expiration date for shortened URLs or allowing users to deactivate or delete URLs when they are no longer needed.

**Limited User Interaction:**

The code lacks user interaction beyond the basic URL shortening process. Implementing user authentication and user-specific functionalities could enhance the user experience.

**No Analytics or Tracking:**

The code does not include analytics or tracking features, preventing users from obtaining insights into the performance of their shortened URLs.

**Solutions and Test Results:**

**Error Handling:**

Solution: Implement comprehensive error handling by using try-except blocks to catch exceptions. Provide meaningful error messages to users.

Test Result: Manually test the code with incorrect Bitly tokens or unavailable services to ensure that the code handles errors gracefully and informs users appropriately.

**Security Measures:**

Solution: Implement input validation for user-provided URLs to prevent malicious input. Additionally, consider using secure practices, such as sanitizing input, validating user permissions, and protecting against common web vulnerabilities.

Test Result: Conduct security testing, including penetration testing, to identify and mitigate potential security vulnerabilities. Verify that the system can withstand common attack vectors.

**Expiration and Deactivation:**

Solution: Introduce a mechanism for setting an expiration date for shortened URLs. Allow users to deactivate or delete URLs when needed.

Test Result: Test the expiration and deactivation features by creating URLs with different expiration dates and verifying that they are no longer accessible after expiration or deactivation.

**User Interaction:**

Solution: Implement user authentication to enable user-specific functionalities. This could include a user dashboard to view and manage their shortened URLs.

Test Result: Verify that user authentication works as intended, allowing users to access their personalized dashboard and perform actions like editing, deactivating, or deleting their URLs.

File organizer:

**File Overwrite:**

Constraint: If there are files with the same name in the destination folders, the shutil.move() operation might overwrite existing files.

Handling: Before moving a file, you can check if a file with the same name already exists in the destination folder. If it does, you can rename the file or handle the situation in a way that avoids overwriting.

**Symbolic Links:**

Constraint: The code does not handle symbolic links explicitly.

Handling: Depending on your use case, you might want to add additional logic to follow or not follow symbolic links. The shutil.move() function has a follow\_symlinks parameter that you can use for this purpose.

**Input Validation:**

Constraint: The code assumes that the user enters a valid folder path when prompted for input.

Handling: Implement input validation to ensure that the provided path exists and is a directory before attempting to organize files.

**File Type Extension Matching:**

Constraint: The code matches file extensions in a case-sensitive manner. For example, ".JPG" would not match the ".jpg" category.

Handling: Normalize file extensions to lowercase before comparing them. You can use

**Cross-Platform Compatibility:**

Constraint: The code assumes a Unix-like file path separator ("/"). On Windows, paths use backslashes ("").

Handling: Use os.path.join() for creating paths instead of hardcoding separators. This ensures cross-platform compatibility.

**Test Results:**

Testing the code involves running it with various scenarios to ensure it behaves correctly. Some test cases to consider:

**Basic Test:**

Organize a folder with a mix of different file types and check if files are moved to the correct folders.

**Overwrite Test:**

Create a scenario where files with the same name exist in different categories. Ensure that the code handles this situation without overwriting files.

**Symbolic Links Test:**

Test the code with folders containing symbolic links and verify if the follow\_symlinks parameter is working as expected.

**Invalid Path Test:**

Input an invalid folder path and check if the code handles it gracefully, providing a clear error message.

**Case-Insensitive Extension Test:**

Create files with extensions in mixed case (e.g., ".JPG", ".PdF") and ensure that the code correctly categorizes them.

# My learnings

**1. URL Shortener Project:**

Overview:

A URL shortener is a service that takes a long URL and generates a shorter alias, making the link easier to share or remember. The project involves creating a web-based URL shortening service.

***Components:***

**Frontend**:

HTML/CSS/JS: To create a simple user interface for entering long URLs and displaying the shortened ones.

Web Framework (optional): Use a framework like Flask or Django for a more structured web application.

**Backend:**

Flask/Django (Web Framework): For handling HTTP requests, routing, and managing the application logic.

Database (SQLite, PostgreSQL, etc.): To store the mapping between short and long URLs.

URL Shortening Algorithm: Generate unique short codes for each URL. Common algorithms include Base62 encoding or Hash functions.

**Libraries:**

Flask/Django: Web frameworks for Python.

SQLAlchemy/Django ORM: For interacting with the database.

Requests: To interact with external APIs (if you want to fetch metadata for the URLs).

pyshorteners: A library for generating short URLs using various URL shortening services.

**Workflow:**

**User Inputs URL:**

User enters a long URL into the frontend.

The frontend sends a request to the backend.

**Backend Processing:**

The backend generates a unique short code for the URL.

The URL and its short code are stored in the database.

**Short URL Display:**

The frontend displays the shortened URL to the user.

**Redirection:**

When someone accesses the short URL, the backend looks up the short code in the database and redirects the user to the corresponding long URL.

**Test Cases:**

Test shortening various long URLs.

Check if the short URL redirects to the correct long URL.

Ensure uniqueness of short codes.

Test for edge cases, such as empty input or invalid URLs.

**2. File Organizer Project:**

Overview:

A file organizer is a tool that automatically categorizes and moves files into specific folders based on their types or other criteria.

**Components:**

**Organizing Logic:**

Determine criteria for file categorization (e.g., file extension, creation date).

Define destination folders for each category.

**Backend Script:**

Write a Python script to automate the file organization process.

Utilize the os and shutil modules for file operations.

Implement error handling and logging for robustness.

**Command-Line Interface (CLI) or GUI:**

Create a user interface for interacting with the file organizer.

For a CLI, use libraries like argparse for parsing command-line arguments.

For a GUI, use tkinter or another GUI library.

***Configurability:***

Allow users to customize the organization criteria, destination folders, etc.

Options can be provided through command-line arguments, a configuration file, or an interactive setup.

**Libraries:**

os: For interacting with the operating system, navigating file paths, and checking file properties.

shutil: For moving and organizing files.

**Workflow:**

User specifies the folder to be organized and any customization options.

The script analyzes the files in the folder based on the defined criteria.

Files are moved to their respective destination folders.

The script provides feedback, such as the number of files moved.

**Test Cases:**

Organize a folder with various file types.

Verify files are moved to the correct folders.

Test with an empty folder or a folder with a large number of files.

Check if the script handles errors gracefully.

Test configurability options.

**Conclusion:**

Both projects offer practical experience in web development (for the URL shortener) and file system manipulation (for the file organizer). They cover a range of concepts, including web frameworks, databases, algorithms, and user interfaces, making them valuable projects for a Python internship. Throughout the internship, you'll gain skills in problem-solving, testing, and improving the efficiency and usability of your projects.