ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

Automated Workflow Optimization in SharePoint: A Power Automate Simulation Study

Mr. Bakraoui Ismail

1st Year Computer and Network Engineering
(3IIR)

28 June, 2024

EMSI Supervisor :
Dr. Azroumahli Chaimae

Jury Members :
Dr. Ahmed Rabhi
Dr. Azroumahli Chaimae
Dr. Jdidou Youssef
Dr. Wafae El Kaysouni

ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

Abstract

This report outlines the development and implementation of a workflow automation project using SharePoint and Power Automate. The primary objective of this project was to streamline and enhance the efficiency of our document approval processes, incorporating automatic updates to document status. By leveraging SharePoint for document management and Power Automate for workflow automation, we aimed to reduce manual intervention, minimize errors, and expedite approval times.

The project commenced with a thorough analysis of the existing manual processes, identifying key pain points and areas for improvement. A document library was established in SharePoint to serve as the central repository for all documents requiring approval. Subsequently, detailed workflows were designed in Power Automate, featuring triggers for new document uploads, conditional logic to determine approval requirements, and automated approval requests sent to designated approvers. Upon approval or rejection, the system automatically updates the document status in SharePoint.

The implementation phase involved configuring the SharePoint document library and building the Power Automate flows, followed by rigorous testing to ensure accuracy and reliability. Post-implementation, the automated workflows were monitored and fine-tuned based on feedback and performance metrics.

The results demonstrated significant improvements in process efficiency, including faster approval times, reduced manual errors, and consistent updates to document status. The report also discusses the challenges encountered during the project, such as permissions management and conditional logic complexities, and the solutions employed to address them.

Overall, the automation project successfully achieved its objectives, providing a robust and scalable solution for document management, approval, and status updates within the organization.

Acknowledgments

Before beginning the writing of my report, I would like to express my gratitude to the Almighty, for granting me the courage and patience necessary to complete this work. I also extend my thanks to all those who have contributed to the success of this project and supported me in the drafting of this report.

I extend my sincere thanks to my parents for providing me with the opportunity to pursue my studies and deepen my knowledge in this field.

I wish to express my deep gratitude to my supervisor, Dr. Chaimae AZROUMAHLI, and to Mr. Bazza Houssam and Mr. Khadir Mohamed for their competent assistance, patience, and encouragement. Their critical insights have been invaluable in structuring my work and enhancing the quality of various sections of the project.

I warmly thank the entire faculty of the Moroccan school of engineering sciences, including both the teaching staff and the administration, as well as the professors of the computer science department, for their efforts in my education and their passion for this field.

I express my sincere appreciation to the members of the jury who have agreed to dedicate their time to reading and evaluating this work.

Finally, I would like to thank all those who, directly or indirectly, have contributed to the completion of this work. Your efforts have been greatly appreciated, and I extend my warmest thanks to you all.

Table of Contents

Table of figures

Table of Listings

Overview

In the dynamic landscape of today's business world, efficiency and automation have emerged as indispensable assets. Manual workflows, once the cornerstone of operations, are now recognized as sluggish, error-prone, and resource-intensive. Consequently, organizations are actively pursuing automation solutions to streamline repetitive tasks, enhance productivity, and curtail operational expenses.

Through my humble experiences, gained from internships and keen observations of professionals spanning from project managers to HR teams, the imperative of automated software engineering has become abundantly clear. It is no longer merely advantageous but rather a prerequisite for staying competitive and adaptive in the modern business arena

Chapter 1

Literature Review

The literature review aims to provide an overview of the existing research and publications relevant to our study. By examining the current state of knowledge in the field, we can identify gaps, draw insights, and build a solid foundation for our research. This chapter will cover key texts and studies that are central to our investigation, starting with a comprehensive exploration of workflow automation using Power Automate.

1 Workflow Automation with Power Automate

The book *Workflow Automation with Power Automate* ([1]) provides an in-depth exploration of Microsoft's Power Automate platform, a tool designed to automate repetitive tasks and streamline business processes. This book is particularly relevant to our study as it covers various aspects of workflow automation, which is a central theme of our research.

1.1 Overview of Power Automate

Power Automate, formerly known as Microsoft Flow, is a service that helps users create automated workflows between their favorite apps and services to synchronize files, get notifications, collect data, and more. The book delves into the fundamentals of creating these workflows, providing readers with practical examples and use cases.

1.2 Key Topics Covered

The book covers a range of topics crucial for understanding and implementing workflow automation:

- **Introduction to Power Automate:** Basics of the platform, including its user interface and core features.

- **Creating Flows:** Step-by-step guides on creating different types of flows, such as automated workflows, button flows, and scheduled flows.
- **Integration with Other Services:** How Power Automate integrates with various Microsoft and third-party services, enabling seamless data flow across platforms.
- **Advanced Techniques:** Tips and techniques for optimizing and troubleshooting workflows, including error handling and performance tuning.

2 Relevance to Current Research

The insights provided in *Workflow Automation with Power Automate* are instrumental for our research on workflow automation in databases. By understanding the capabilities and limitations of Power Automate, we can better assess its applicability in automating complex database operations. Furthermore, the book's practical examples serve as a foundation for developing our own automated workflows, which will be tested and evaluated in our study.

Chapter 2

Research Framework

This chapter presents the research framework for investigating the effectiveness of automated workflow optimization in SharePoint using Power Automate. The framework encompasses the background and context of the study, the research objectives, and an overview of the methodology.

1 Background and Context

Through my internships and interactions with various multinational companies and startups, I observed a common challenge: maintaining and updating shared databases, whether for HR or project management, becomes difficult as the employee count grows. This issue often leads to inefficiencies and increased workloads.

My perspective broadened when I discovered Microsoft Power Platforms, particularly Power Automate and SharePoint. Through hands-on experience, I implemented Power Automate to create workflows that reduced manual data entry, minimized errors, and expedited processes. Utilizing SharePoint as a centralized repository enhanced organization, access control, and real-time collaboration.

These tools significantly improved data management practices, demonstrating their potential to transform operations within organizations. My experience with Power Automate and SharePoint has shown me how effective these platforms can be in addressing and overcoming common data management challenges.

2 Research Objectives

The primary objective of this research is to investigate the effectiveness of automated workflow optimization in SharePoint using Power Automate, then to apply the same logic using Python scripts in order to compare effectiveness. Specifically, the research aims to achieve the following objectives:

- Design and implement automated workflows using Power Automate to address identified challenges.
- Evaluate the performance and impact of automated workflows compared to manual processes.
- Identify key challenges and inefficiencies in manual workflow processes within SharePoint.
- duplicate said workflows using python scripts to compare effectiveness using metrics.
- Provide recommendations for optimizing and scaling Databases depending on the user background.

3 Methodology Overview

1. **Database Creation and Population:**
   - **Creation of Test Database:** The research begins with the creation of a test database from scratch to serve as the foundation for the automated workflows.
   - **Database Population:** Data is populated into the test database to simulate real-world scenarios and ensure that the workflows operate effectively with realistic data.
2. **Workflow Creation and Optimization:**
   - **Creation of Flows:** Automated workflows are designed and implemented to serve multiple purposes, such as email approvals for creating or modifying items within the database.
   - **Automatic Update of Items:** Recurrent workflows are developed to automatically update items within the database based on predefined criteria or triggers.
   - **Time Optimization:** Strategies are employed to optimize the timing and recurrence of these workflows, ensuring efficient and timely execution.
3. **Integration with External Systems:**
   - **Python Code Conversion:** As a final step, the workflows are converted into Python code to test in the same Database to ensure leveled ground for comparison.
   - **External Database Integration:** The Python code is set to work with external databases, allowing for seamless data exchange .

Gantt Digram

A Gantt chart is an essential tool for illustrating the key steps and their timelines in project management, from the initial conception to actual implementation. The diagram below outlines each phase of the project, detailing the tasks and their durations, providing a clear visual representation of the project's progression from start to finish.

Name	ID	Start Date	End Date	Duration	Color
identifying and solving the probl…	1	May 20, 2024	Jun 26, 2024	28 days	
Superficial research	2	May 20, 2024	May 24, 2024	5 days	
Report writing	3	May 20, 2024	Jun 26, 2024	28 days	
Getting used to software	4	May 20, 2024	May 30, 2024	9 days	
Power Platforms implementation	5	May 28, 2024	Jun 10, 2024	10 days	
Database Creation	8	May 28, 2024	May 29, 2024	2 days	
Creation of Workflows	6	May 28, 2024	Jun 06, 2024	8 days	
Enhancing workflows Durability	7	Jun 06, 2024	Jun 10, 2024	3 days	
Python Migration	9	Jun 10, 2024	Jun 24, 2024	11 days	
Granting Permissions for Azur…	10	Jun 10, 2024	Jun 14, 2024	5 days	
Code creation and test	11	Jun 10, 2024	Jun 20, 2024	9 days	
Code Optimization	12	Jun 20, 2024	Jun 24, 2024	3 days	

Figure 2.1: Gantt Diagram table

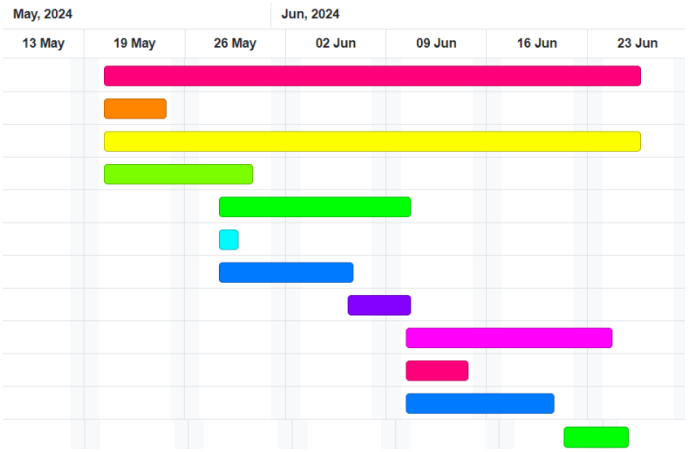


Figure 2.2: Gantt Diagram Bar chart

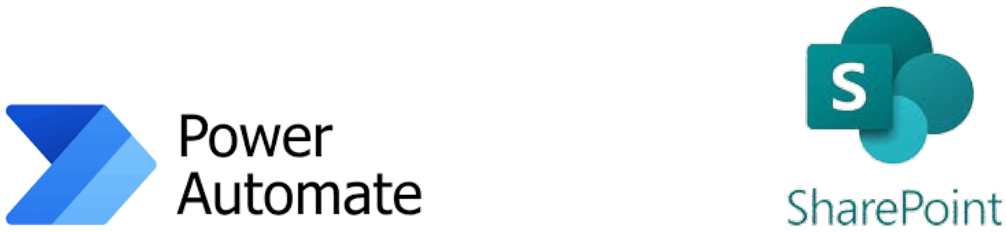
Chapter 3

Theoretical Framework

This chapter presents the theoretical framework for understanding and implementing automated workflow optimization in SharePoint using Power Automate. It includes discussions on the principles of automated process simulation, the integration of Power Automate with SharePoint, workflow design and optimization strategies, and metrics for evaluating automation impact.

1 Principles of Automated Process Simulation

Automated process simulation plays a crucial role in optimizing workflow efficiency and effectiveness within organizations. By simulating and automating repetitive tasks, businesses can streamline their operations, reduce manual errors, and enhance productivity. The principles of automated process simulation encompass various concepts such as process modeling, workflow design, and performance evaluation.

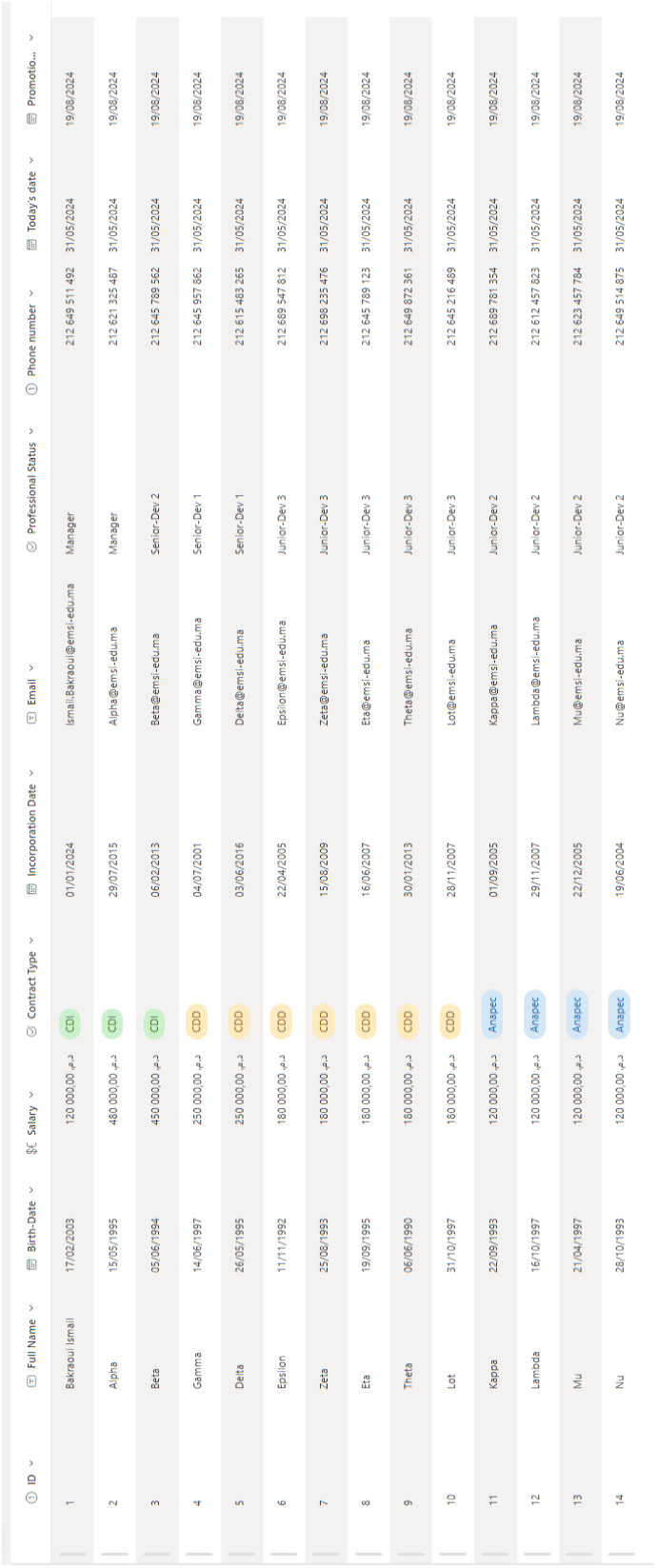


(a) Power Automate (b) SharePoint

Figure 3.1: Power Platforms

2 Selection of Case Study

The test database selected for this study replicates a typical HR department database, encompassing columns such as names, incorporation dates, and contract types, which are essential elements in the workflow processes discussed in the third chapter. Although hidden during the aesthetic process, the "today's date" column remains a focal point due to its pivotal role in triggering time-sensitive operations. This column ensures timely actions such as sending notifications or updating records, thereby enhancing the efficiency and responsiveness of the HR system. The use of the Greek alphabet in this database aims to simplify the testing phase for developers and clearly distinguish various approval stages. This approach not only facilitates easy identification and management of different workflow components but also creates a structured and unambiguous test scenario. By focusing on a SharePoint database, widely utilized in corporate settings for managing employee information and HR-related data, this case study will demonstrate the practical benefits and scalability of automating workflows. It highlights how Python can be leveraged to automate and optimize both locally stored and cloud-based databases, leading to improved data management, streamlined processes, and increased operational efficiency within HR departments.



ID	Full Name	Birth-Date	Salary	Contract Type	Incorporation Date	Email	Professional Status	Phone number	Today's date	Promotio...
1	Bakraoui Ismail	17/02/2003	120 000,00 د.م.	CDI	01/01/2024	Ismail.Bakraoui@emsi-edu.ma	Manager	212 649 511 492	31/05/2024	19/08/2024
2	Alpha	15/05/1995	480 000,00 د.م.	CDI	29/07/2015	Alpha@emsi-edu.ma	Manager	212 621 325 487	31/05/2024	19/08/2024
3	Beta	05/06/1994	450 000,00 د.م.	CDI	06/02/2013	Beta@emsi-edu.ma	Senior-Dev 2	212 645 789 562	31/05/2024	19/08/2024
4	Gamma	14/06/1997	250 000,00 د.م.	CDD	04/07/2001	Gamma@emsi-edu.ma	Senior-Dev 1	212 645 957 862	31/05/2024	19/08/2024
5	Delta	26/05/1995	250 000,00 د.م.	CDD	03/06/2016	Delta@emsi-edu.ma	Senior-Dev 1	212 615 483 265	31/05/2024	19/08/2024
6	Epsilon	11/11/1992	180 000,00 د.م.	CDD	22/04/2005	Epsilon@emsi-edu.ma	Junior-Dev 3	212 689 547 812	31/05/2024	19/08/2024
7	Zeta	25/08/1993	180 000,00 د.م.	CDD	15/08/2009	Zeta@emsi-edu.ma	Junior-Dev 3	212 698 235 476	31/05/2024	19/08/2024
8	Eta	19/09/1995	180 000,00 د.م.	CDD	16/06/2007	Eta@emsi-edu.ma	Junior-Dev 3	212 645 789 123	31/05/2024	19/08/2024
9	Theta	06/06/1990	180 000,00 د.م.	CDD	30/01/2013	Theta@emsi-edu.ma	Junior-Dev 3	212 649 872 361	31/05/2024	19/08/2024
10	Lot	31/10/1997	180 000,00 د.م.	CDD	28/11/2007	Lot@emsi-edu.ma	Junior-Dev 3	212 645 216 489	31/05/2024	19/08/2024
11	Kappa	22/09/1993	120 000,00 د.م.	Anapec	01/09/2005	Kappa@emsi-edu.ma	Junior-Dev 2	212 689 781 354	31/05/2024	19/08/2024
12	Lambda	16/10/1997	120 000,00 د.م.	Anapec	29/11/2007	Lambda@emsi-edu.ma	Junior-Dev 2	212 612 457 823	31/05/2024	19/08/2024
13	Mu	21/04/1997	120 000,00 د.م.	Anapec	22/12/2005	Mu@emsi-edu.ma	Junior-Dev 2	212 623 457 784	31/05/2024	19/08/2024
14	Nu	28/10/1993	120 000,00 د.م.	Anapec	19/06/2004	Nu@emsi-edu.ma	Junior-Dev 2	212 649 514 875	31/05/2024	19/08/2024

Figure 3.2: The Test Database

3 Integration of Power Automate with Sharepoint

The integration between Power Automate and SharePoint commences through formal access to both platforms using an authorized account, notably the EMSI@ma credentials. The ensuing figure elegantly illustrates this pivotal process.

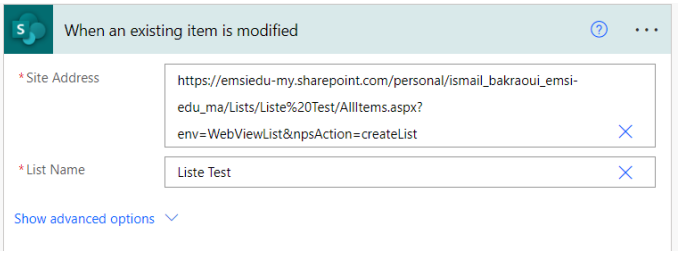


Figure 3.3: Illustration of the integration process

4 Workflow Designs and Optimization Strategies

The first automatic workflow demonstrates the approval process, which is one of the simplest workflows offered by Power Automate. It is primarily designed to serve as a trigger for managers to review changes proposed by their teams. Once these changes are reviewed, managers can either approve or reject the updates.

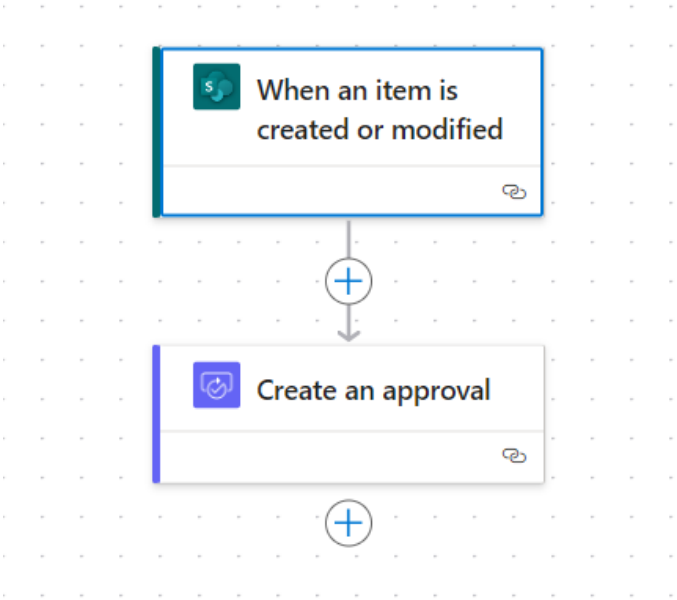


Figure 3.4: Approval Workflow

The second workflow is more complex. Unlike the automatic approval workflow, this is a recurring workflow that updates the current date in SharePoint, which SharePoint alone cannot do. This workflow involves several steps: setting the recurrence, retrieving the items, and finally applying the update to the current date every day at a specified time.

The purpose of this flow is to insert an automated "Today's Date" column in SharePoint. This enables the creation of other calculated columns based on the current date, such as the promotion date in our case.

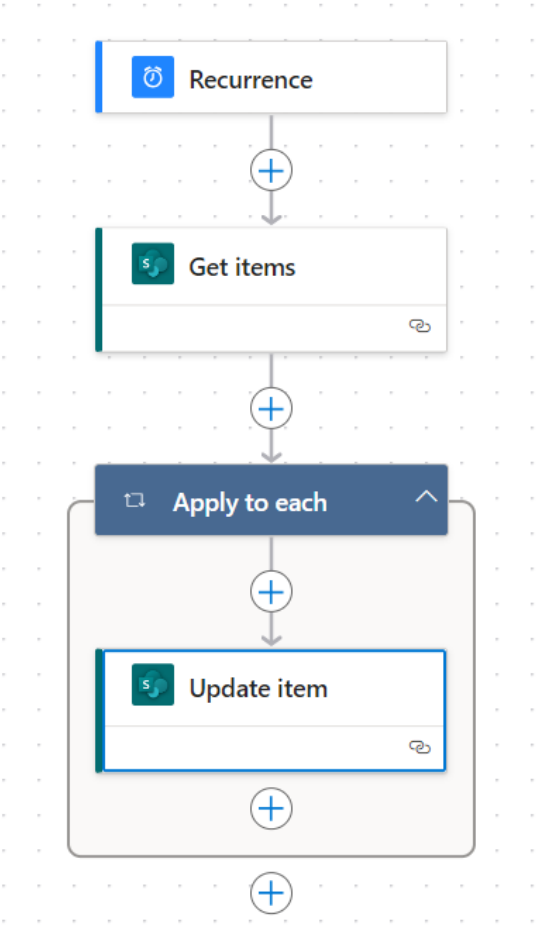


Figure 3.5: Today's date Recurring flow

The third workflow is designed to update employee positions and contract types, taking into account several key date columns, such as promotion dates. The workflow's primary function is to ensure that these updates are accurately and consistently applied based on the current date.

One of the critical aspects of this workflow is its reliance on today's date to determine the necessary actions. Specifically, as illustrated in Figure 3.5, the workflow compares today's date with the incorporation date of each employee. This comparison is crucial for identifying employees who are eligible for promotions or changes in their contract types.

The workflow starts by identifying the relevant date columns to be monitored and continuously checks them against today's date. If the conditions in the company's promotion policy are met, the workflow triggers the necessary updates.

This adaptable workflow can be duplicated to meet various departmental needs, ensuring that each department's unique promotion criteria and timelines are respected.

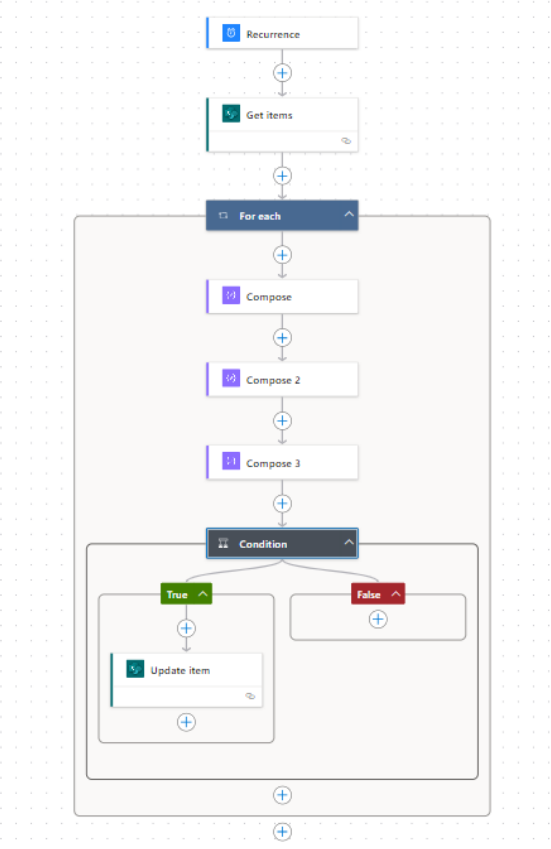


Figure 3.6: Promotion Update Workflow

In this case, the changes will occur in the Professional Status column, updating the value from 'Junior-Dev 1' to 'Junior-Dev 2'. Additionally, the type of contract will be changed from 'Anapec' to 'CDD'. All of these updates have been implemented using the "Update Item" clause offered by Power Automate in SharePoint. This workflow ensures that updates are made systematically and in accordance with the company's policies.

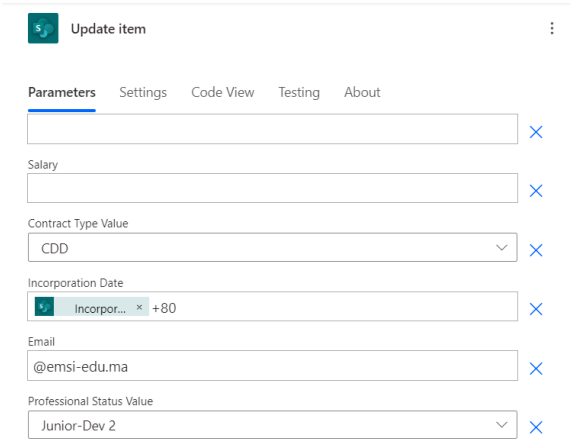


Figure 3.7: Update item Configuration JD1 - JD2

To update the remaining positions in alignment with the company's policy, we have two main options. First, we can duplicate the flow for each desired position, such as creating a specific flow for transitioning from Junior Dev 2 to Junior Dev 3, and another flow for different position changes. Alternatively, we can use the OR clause in the update item option within a single flow to handle multiple position updates. This approach allows us to streamline the process by consolidating all position updates into one flow, simplifying management and maintenance.

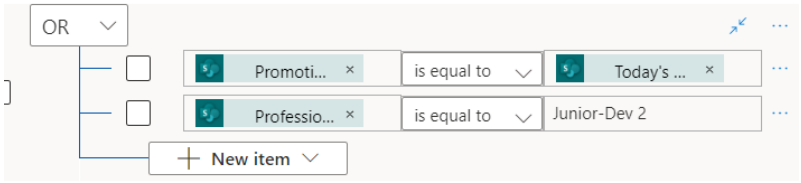


Figure 3.8: OR clause

5 Code breakdown

5.1 Integrity workflow code breakdown

The upcoming JSON code breakdown exemplifies the integrity workflow's progression throughout this process. It encompasses the inception of API definitions and extends seamlessly to the incorporation of parameters. This detailed exploration serves to clarify the intricate workings of the integrity workflow in its entirety.

```
1
2  {
3    "type": "OpenApiConnection",
4    "inputs": {
5      "parameters": {
6        "approvalType": "BasicAwaitAll",
7        "ApprovalCreationInput/title": "changes approval to @{
     triggerBody()?['Nom']}",
8        "ApprovalCreationInput/assignedTo": "Ismail.Bakraoui@emsi-edu
     .ma;",
9        "ApprovalCreationInput/details": "@triggerBody()?['
     ContractType']?['Value']",
10       "ApprovalCreationInput/itemLink": "@triggerBody()?['{Link
     }']",
11       "ApprovalCreationInput/enableNotifications": true,
12       "ApprovalCreationInput/enableReassignment": true
13     },
14     "host": {
15       "apiId": "/providers/Microsoft.PowerApps/apis/
     shared_approvals",
16       "connection": "shared_approvals",
17       "operationId": "CreateAnApproval"
18     }
19   },
20   "runAfter": {},
21   "metadata": {
22     "operationMetadataId": "f483c9f5-b048-4214-abee-c07c56d22c55"
23   }
24 }
```

Listing 3.1: Approval workflow

JSON block: This block also defines an OpenAPI connection, but this time it's related to approval processes. It includes parameters for approval titl.1 – Your JSON code snipe, assigned user, details, item link, and settings for notifications and reassignment. Metadata is provided for operation identification as well.

5.2 Today's date code breakdown

The provided JSON code snippet illustrates a workflow designed to interact with a SharePoint database. The workflow involves retrieving items from the specified dataset using the "GetItems" operation and subsequently updating each item's title and today's date using the "PatchItem" operation. This automation process ensures timely and accurate updates to the SharePoint database, enhancing its efficiency and data integrity.

```
1
2 {
3 "type": "Foreach",
4 "foreach": "@outputs('Get_items')?['body/value']",
5 "actions": {
6 "Update_item": {
7 "type": "OpenApiConnection",
8 "inputs": {
9 "parameters": {
10 "dataset": "https://emsiedu-my.sharepoint.com/personal/
     ismail_bakraoui_emsi-edu_ma",
11 "table": "c0980682-f2b0-44e5-9ca5-f9fc4700dfd5",
12 "id": "@items('Apply_to_each')?['ID']",
13 "item/Title": "@items('Apply_to_each')?['Nom']",
14 "item/Todaysdate": "@{utcNow()}"
15 },
16 "host": {
17 "apiId": "/providers/Microsoft.PowerApps/apis/
     shared_sharepointonline",
18 "connection": "shared_sharepointonline",
19 "operationId": "PatchItem"
20 }
21 }
22 }
23 },
24 "runAfter": {
25 "Get_items": ["Succeeded"]
26 }
27 }
```

Listing 3.2: Today's date workflow JSON code

Update of Items: Following the retrieval of items, a "Foreach" loop is employed to iterate through each item retrieved. Within the loop, the "PatchItem" operation is utilized to update each item's title and today's date. This operation also defines an OpenAPI connection with the required parameters, such as the dataset URL, table ID, item ID, title, and today's date.

5.3 Update Promotion code breakdown

the following JSON code block illustrates the process of the update promotion workflow, from defining the API connection to updating the position of employees, it also checks the conditions of the update as shown in figure 3.8.

```
{
  // Define the type of connection
  "type": "OpenApiConnection",

  // Specify the inputs for the connection
  "inputs": {

    // Parameters for the dataset, table, and item to be updated
    "parameters": {
      // URL of the dataset
      "dataset": "https://emsiedu-my.sharepoint.com/personal/
   ismail_bakraoui_emsi-edu_ma",
      // Identifier for the specific table within the dataset
      "table": "c0980682-f2b0-44e5-9ca5-f9fc4700dfd5",
      // ID of the item to be updated, retrieved dynamically
      "id": "@items('For_each')?['ID']",
      // Set the ContractType value to 'CDI'
      "item/ContractType/Value": "CDI",
      // Set the Level value to 'Junior-Dev 2'
      "item/Level/Value": "Junior-Dev 2"
    },

    // Host details for the connection
    "host": {
      // API ID for the connection to SharePoint Online
      "apiId": "/providers/Microsoft.PowerApps/apis/
   shared_sharepointonline",
      // Connection name
      "connection": "shared_sharepointonline",
      // Operation ID for the PatchItem operation
      "operationId": "PatchItem"
    }
  }
```

Listing 3.3: Promotion Promotion workflow JSON code

Json block: This configuration defines an OpenAPI connection for updating an item in a SharePoint list through Microsoft Power Automate. It specifies parameters such as the dataset URL, table identifier, and dynamically retrieved item ID. It updates the ContractType to "CDI" and the Level to "Junior-Dev 2" after it checks the inputted condition true. The connection details include the API ID for SharePoint Online, the connection name, and the operation ID for the PatchItem operation. Metadata is also provided to uniquely identify this specific operation.

6 Scalability

The scalability of this project is boundless, underscoring the versatility of automations. These automations can stem from minor adjustments, whether they involve obtaining approvals, sending email notifications, or shifting to automated workflows. The forthcoming chapter will delve into the migration of these workflows from Power Platforms to various programming languages. This transition will showcase the vast potential for automating both locally stored and cloud-based databases, leveraging the techniques and principles provided by languages such as Python.

By embracing these advanced automation strategies, organizations can unlock new levels of efficiency and innovation. The scalability of Python-based solutions ensures that they can grow alongside the business, adapting to increased data volumes and more complex processes without compromising performance. This chapter will not only explore the technical aspects of workflow migration but also highlight the broader implications for business operations, demonstrating how Python's powerful capabilities can drive transformative changes across various sectors, accommodating the evolving needs of a dynamic market.

7 Metrics for Evaluating Automation Impact

The metrics relied upon to measure the impact are derived from Power BI within the Power Platform. These metrics unveil both successful and failed attempts during the testing phase of each workflow. Additionally, they calculate the average time taken for every instance of the workflow, highlighting reductions achieved during the optimization phase.

Figure 3.9: Power BI

For the approval workflow, commonly referred to in the company as the integrity workflow—a recurrent flow that ensures nothing has been changed or modified in a dedicated database, the results were as follows:

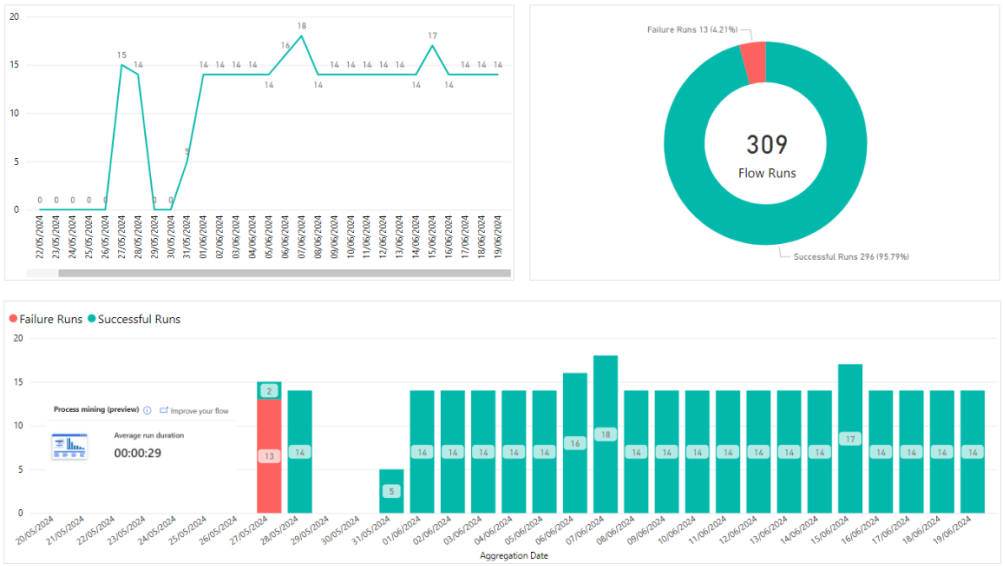


Figure 3.10: Approval workflow Metrics

- **Initial Inactivity**: The workflow was inactive from 5/4/2024 to 5/26/2024, the creation and implementation phase started on 5/23/2024.
- **Spike in Activity and Issues**: There was a large number of runs on 5/27/2024, with a high failure rate, highlighting the testing phase.
- **Resolution and Stability**: After the high failure rate on 5/27/2024, subsequent days showed improved performance with all runs being successful.
- **Flow Health**: The overall health of the workflow is good, with 95.79% the runs being successful. The issue on 5/27/2024 was a failed attempt to improve on the workflow's primary function.

For today's date recurrent workflow, the metrics applied here are perceived differently because the recurrence is set to run only once a day. The testing phase was brief; once the workflow was completed, its function was achieved. The only remaining task at hand was to optimize the process by reducing the average time the workflow took to complete.

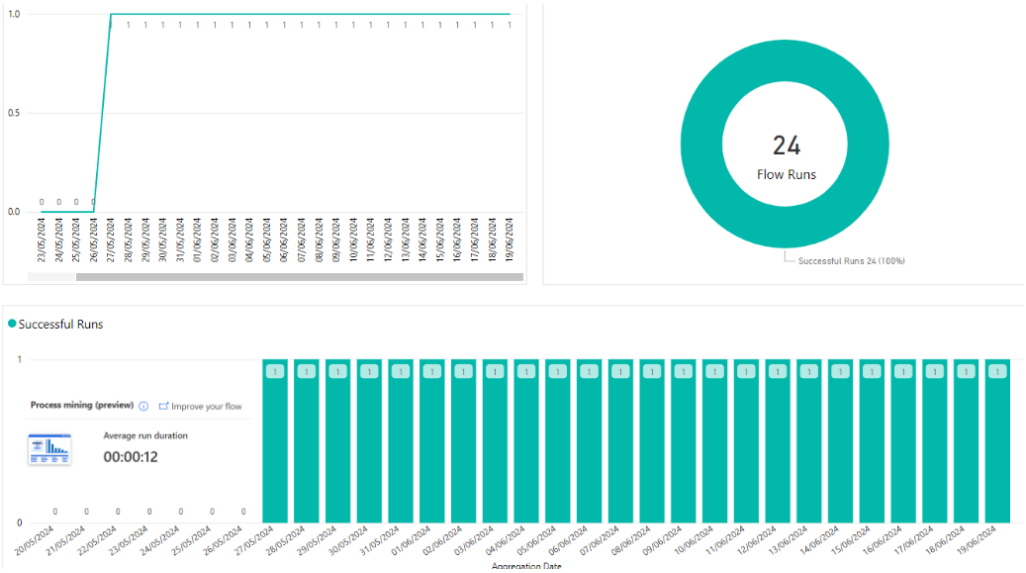


Figure 3.11: Today's date Workflow Metrics

- **Initial Inactivity**: The workflow was inactive from 5/20/2024 to 5/24/2024, the creation and implementation phase started on 5/23/2024.
- **Spike in Activity and Issues**: Activity began on 5/25/2024, but there were no recorded issues, indicating successful initial tests.
- **Resolution and Stability**: From 5/25/2024 onwards, the workflow showed consistent performance with all runs being successful.

- **Flow Health**: The overall health of the workflow is excellent, with 100% of the runs being successful, demonstrating a well-implemented and stable system.

For the Promotion update recurrent workflow, the metrics applied here are perceived similarly to the last because the recurrence is set to run only once a day. The testing phase was brief; once the workflow was completed, its function was achieved. The only remaining task at hand was to optimize the process by reducing the average time the workflow took to complete.

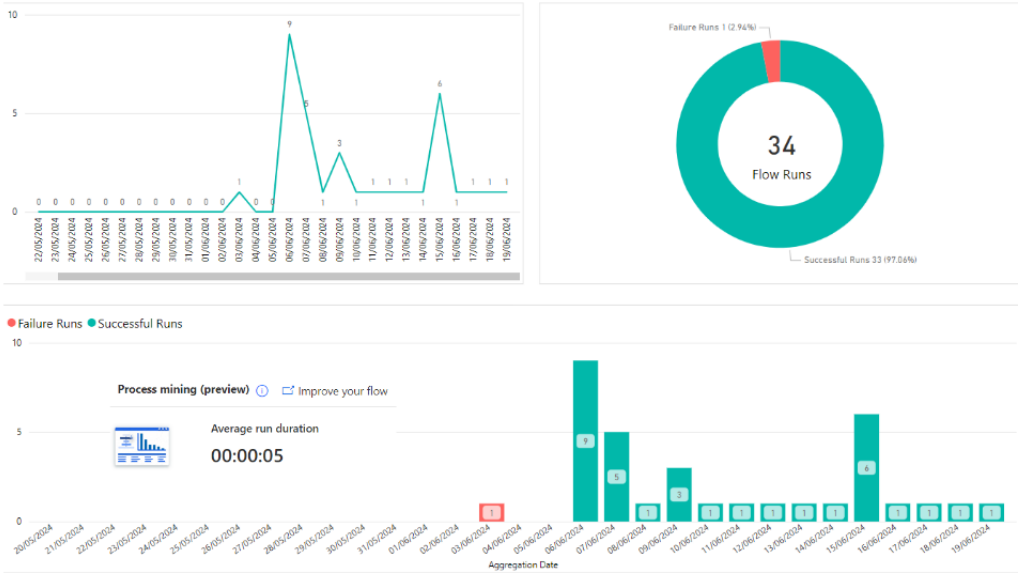


Figure 3.12: Promotion Update Workflow Metrics

- **Initial Inactivity**: The workflow was inactive from 20/05/2024 to 23/05/2024, with the creation and implementation phase starting on 23/05/2024.
- **Spike in Activity and Issues**: Activity began on 24/05/2024, with no recorded issues initially, indicating successful initial tests.
- **First Recorded Issues**: A spike in activity is noted on 06/06/2024 with 9 runs, followed by a period of multiple successful runs and occasional failures, highlighting the need for adjustments.
- **Resolution and Stability**: From 14/06/2024 onwards, the workflow showed consistent performance with only one failed run, indicating improved stability.
- **Flow Health**: The overall health of the workflow is excellent, with 97.06% of the runs being successful, demonstrating a well-implemented and stable system.

Chapter 4

Power Platforms to Python

Although Power Automate performs well in the realm of automated software engineering, my exploration into programming languages like Python stems from a desire to uncover the untapped potential that goes beyond the capabilities of packaged solutions like the Power Platform. Python, renowned for its prowess in data science, has been extensively highlighted in research as an ideal language for my thesis. This is supported by numerous studies, such as W. McKinney's book, "Python for Data Analysis" [2], which has been particularly useful during this study.

During a test phase involving a real-life database of more than 500 employees, I observed recurring deficiencies in Power Automate. These included limited performance with large datasets, inadequate support for complex data manipulation tasks, and insufficient error handling capabilities.

In a recent article by Marcel Broschk, a specialist in the Power Platform, titled *"10 Challenges and Frustrations with Microsoft Power Apps"* ([3]), various problems with Power Apps are highlighted from a user's perspective. These include slower load times, restricted design options, limited offline capabilities, and challenges related to external data sources. All of these issues resonate with the challenges encountered in our study.

This section will delve into an in-depth study of how Python represents one of the best alternatives for automated software engineering (ASE) from a developer's perspective. By comparing Python to Power Automate, the analysis will highlight Python's superior capabilities in handling large volumes of data, executing complex data manipulations, and providing robust error handling. The study will examine real-life scenarios, showcasing how Python's flexibility, efficiency, and extensive libraries make it a more powerful and versatile tool for developers. Through this examination, the research will demonstrate Python's potential to overcome the limitations of Power Automate, establishing it as an optimal choice for advanced ASE tasks.

1 Methodology

1. **Database Choice:** This part of the study will begin by using the same SharePoint list as shown in Figure 3.2. The reason for this choice is to create an equal environment for running automation scripts, regardless of the tool used. Additionally, the SharePoint list is hosted in the cloud, providing free access compared to other options.

2. **IDE Selection and Scripts Running:** I chose Visual Studio Code (VS Code) as the integrated development environment (IDE) for this study due to its versatility and extensive extension support, which facilitate efficient development and debugging of Python scripts. The scripts will connect to SharePoint via the SharePoint REST API to automate and schedule tasks such as creating, updating, and deleting list items, thereby mimicking the behavior of Power Automate workflows. This setup allows us to create an equal environment for automation, ensuring a fair comparison between the two tools. By running these scripts recurrently using task schedulers, we can evaluate the performance, reliability, and ease of use of Python-based automation versus Power Automate.

3. **Metrics:** To evaluate the performance metrics, Python scripts in Visual Studio Code will be timed and logged during execution. This will include measuring execution times for tasks such as data manipulation and workflow automation, allowing for comparison with equivalent tasks performed using Power Automate.

4. **User Guiding:** After the comparative study, guiding the user will be the final and most crucial step. Users with a background in coding will be directed towards the Python-based automation option, leveraging their technical skills to fully customize and control their workflows. Conversely, users without a background in coding or STEM will be guided towards Power Automate, benefiting from its user-friendly interface and ease of use. This tailored guidance ensures that each user can effectively implement automated workflows based on their technical proficiency and needs.

2 Implementation of Workflows in Python

2.1 Workflow Implementation

In creating and implementing workflows using Python, we utilized Visual Studio Code (VS Code) as our integrated development environment (IDE), paired with the latest version of Python, version 3.12. This combination of tools and technologies enabled us to develop robust automation scripts tailored to mimic and enhance the functionalities typically managed by Power Automate in SharePoint.

(a) Python 3.12

(b) VS Code IDE

Figure 4.1: Migration Tools

Python Migration Steps

1. To interact with SharePoint's REST API and automate tasks, we utilize several libraries. The Office365-REST-Python-Client library, in conjunction with the msal library, facilitates authentication and seamless interaction with SharePoint. The requests library simplifies sending HTTP requests, making it easy to perform CRUD operations. Additionally, the schedule library allows for the automation of recurrent tasks. Metrics will be used to compare the efficiency of these scripts with the workflows discussed in Chapter 3.

```
PS C:\Users\DELL> pip install msal
PS C:\Users\DELL> pip install requests
PS C:\Users\DELL> pip install schedule
PS C:\Users\DELL> pip install Office365-REST-Python
```

Figure 4.2: SharePoint Python package installation

2. Make sure all permissions are granted to handle all Microsoft connections, full control of the registrations in azure AD and lists in SharePoint is necessary

3. Use the following script to import the necessary packages.

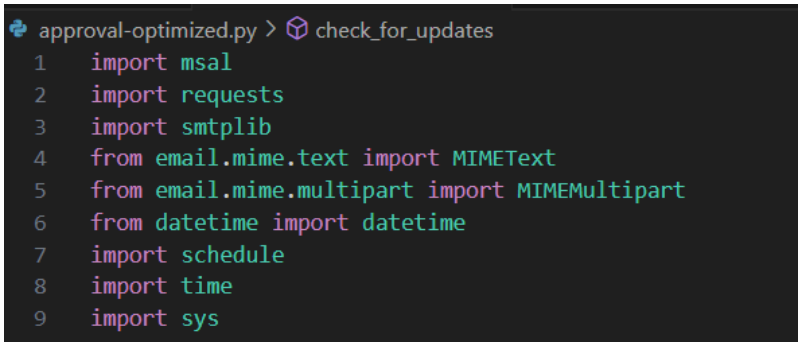


Figure 4.3: packages import

4. Accessing your SharePoint site, set up authentication by providing the appropriate SharePoint credentials, along with the AZURE AD registration details of the app. This step ensures secure access to the site and its resources.

```
    # Azure AD app registration details
    TENANT_ID = "********-****-****-****-************"
    CLIENT_ID = "********-****-****-****-************"
    CLIENT_SECRET = "****************************************"
    HOSTNAME = "*************.sharepoint.com"
    SITE_RELATIVE_PATH = "/personal/********_********_********"
    LIST_NAME = "Liste Test"

    # Email credentials
    SENDER_EMAIL = "***************@gmail.com"
    SENDER_PASSWORD = "****************"
    RECIPIENT_EMAIL = "*****************@emsi-edu.ma"

```

Listing 4.1: Azure AD and Sharepoint Credentials

5. Develop scripts that replicate the workflows discussed in Chapter 3. These scripts will automate the same tasks and processes while ensuring efficient operation and integration with your SharePoint environment.

6. Apply metrics to be able to compare python scripts and power automate workflows in terms of time and efficiency.

2.2 Code breakdown

Connection to Microsoft Azure and Sharepoint list

In the first section of our code, we will demonstrate how to establish a connection using the specified library. To maintain a clean and organized code structure, it is better to store the credentials in a JSON format file. During this test phase, we will verify that the connection is both valid and functional. Additionally, we will work with environment variables to enhance security. It is essential to ensure that the necessary permissions are granted to the user for successful operation.

```
#import all libraries
#Enter all Azure AD app registration details
#Enter all Sharepoint credentials
def get_access_token():
    app = msal.ConfidentialClientApplication(
        CLIENT_ID,
    authority=f"https://login.microsoftonline.com/{TENANT_ID}",
        client_credential=CLIENT_SECRET
    )
    result = app.acquire_token_for_client(scopes=["https://graph.
   microsoft.com/.default"])
    if "access_token" not in result:
        raise Exception(f"Failed to obtain access token: {result}")
    print("Token obtained")
    return result["access_token"]
def get_json_response(url, headers):
    response = requests.get(url, headers=headers)
    response.raise_for_status()
    return response.json()
def find_sharepoint_list():
    try:
        token = get_access_token()
        headers = {"Authorization": f"Bearer {token}", "Accept": "
   application/json"}
        site_url = f"https://graph.microsoft.com/v1.0/sites/{
   HOSTNAME}:{SITE_RELATIVE_PATH}"
        site_id = get_json_response(site_url, headers)['id']
        lists_url = f"https://graph.microsoft.com/v1.0/sites/{
   site_id}/lists"
        lists = get_json_response(lists_url, headers)['value']
        list_found = any(lst['displayName'] == LIST_NAME for lst in
    lists)
        if list_found:
            print("List found")
        else:
            print("List not found")
    except Exception as e:
        print(f"Error: {e}")

```

```
find_sharepoint_list()
```

Listing 4.2: Connection to Microsoft Azure and Sharepoint list

Integrity Worflow

In the second section, we will replicate the integrity flow described in Listing 3.1. Our implementation will involve the use of the msal library and the requests library for interacting with the Microsoft Graph API to fetch SharePoint list data. Additionally, we will incorporate metrics using the schedule library to automate the update checks and measure the time taken for each operation. By doing so, we can compare the performance of our code with the original workflow, highlighting the enhancements and efficiency gains achieved. This approach not only ensures the integrity of the data but also demonstrates the effectiveness of our optimized script in detecting changes and sending email notifications promptly.

```
#import all libraries
#Enter all Azure AD app registration details
#Enter all Sharepoint credentials
#Enter Email credentials
#function to send email

def get_json_response(url, headers):
    response = requests.get(url, headers=headers)
    response.raise_for_status()
    return response.json()

def initialize_state(headers, site_id, list_id):
    for item in get_json_response(f"https://graph.microsoft.com/v1.0/sites/{site_id}/lists/{list_id}/items?expand=fields", headers)['value']:
        initial_state[item['id']] = item['fields']
    print("Initial state set successfully")

def check_for_updates():
    start_time = time.time()
    try:
        headers = {"Authorization": f"Bearer {get_access_token()}", "Accept": "application/json"}
        site_id = get_json_response(f"https://graph.microsoft.com/v1.0/sites/{CONFIG['HOSTNAME']}:{CONFIG['SITE_RELATIVE_PATH']}", headers)['id']
        list_id = next(lst['id'] for lst in get_json_response(f"https://graph.microsoft.com/v1.0/sites/{site_id}/lists", headers)['value'] if lst['displayName'] == CONFIG["LIST_NAME"])
        items = get_json_response(f"https://graph.microsoft.com/v1.0/sites/{site_id}/lists/{list_id}/items?expand=fields", headers)['value']

```

```
        changes_found = False
        for item in items:
            item_id, fields = item['id'], item['fields']
            if item_id in initial_state and datetime.strptime(fields["Modified"], "%Y-%m-%dT%H:%M:%SZ") > datetime.strptime(initial_state[item_id]["Modified"], "%Y-%m-%dT%H:%M:%SZ"):
                changes = [(k, initial_state[item_id][k], v) for k, v in fields.items() if k not in CONFIG["IGNORED_FIELDS"] and initial_state[item_id].get(k) != v]
                initial_state[item_id] = fields
                if changes:
                    send_email(f"Change detected in {fields.get('Title', 'No Title')}", f"Mr. Bakraoui Ismail,\n\nThe following changes were detected:\n\n" + "\n".join([f"{col}: {old} -> {new}" for col, old, new in changes]) + "\n\nBest regards,\nAutomated Notification System")
                    changes_found = True
            else:
                initial_state[item_id] = fields

        print("Changes detected and emails sent." if changes_found else "No changes detected.")

    except Exception as e:
        print(f"Error while checking for updates: {e}")
    finally:
        print(f"Time taken to check for updates and send emails: {time.time() - start_time:.2f} seconds")
        sys.exit(0)

schedule.every().day.at("12:00").do(check_for_updates)
print("Scheduled check_for_updates to run at noon")
while True:
    schedule.run_pending()
    time.sleep(1)
```

Listing 4.3: Approval workflow Python script

The "Today's date" workflow will not be duplicated in our study. Instead, we will leverage the schedule library to automate the timing of our tasks, thereby eliminating the need to manually retrieve and use today's date. The schedule library efficiently handles the timing of operations, providing a simpler and more reliable solution for scheduling periodic tasks.

Promotion Update

This third section of code will represent the update promotion recurrent workflow, as the precedent,code to determine metrics will be added to define efficiency.

```
#import all libraries
#Enter all Azure AD app registration details
#Enter all Sharepoint credentials
def get_access_token():
    result = msal.ConfidentialClientApplication(
        CONFIG["CLIENT_ID"],
        authority=f"https://login.microsoftonline.com/{CONFIG['TENANT_ID']}",
        client_credential=CONFIG["CLIENT_SECRET"]
    ).acquire_token_for_client(scopes=["https://graph.microsoft.com/.default"])
    if "access_token" not in result:
        raise Exception("Failed to obtain access token")
    return result["access_token"]

def get_json_response(url, headers):
    response = requests.get(url, headers=headers)
    response.raise_for_status()
    return response.json()

def update_item(headers, site_id, list_id, item_id, updates):
    url = f"https://graph.microsoft.com/v1.0/sites/{site_id}/lists/{list_id}/items/{item_id}/fields"
    response = requests.patch(url, headers=headers, json=updates)
    response.raise_for_status()
    return response.json()

def check_and_update_items():
    try:
        headers = {"Authorization": f"Bearer {get_access_token()}", "Accept": "application/json"}
        site_id = get_json_response(f"https://graph.microsoft.com/v1.0/sites/{CONFIG['HOSTNAME']}:{CONFIG['SITE_RELATIVE_PATH']}", headers)['id']
        list_id = next(lst['id'] for lst in get_json_response(f"https://graph.microsoft.com/v1.0/sites/{site_id}/lists", headers)['value'] if lst['displayName'] == CONFIG["LIST_NAME"])
        items = get_json_response(f"https://graph.microsoft.com/v1.0/sites/{site_id}/lists/{list_id}/items?expand=fields", headers)['value']

        today = datetime.utcnow().date().isoformat()
        for item in items:
            fields = item['fields']
            updates = {}

```

```
            print(f"Checking item {item['id']} with fields: {fields}")

            if 'Promotion_x0020_date' in fields and fields['Promotion_x0020_date'][:10] == today:
                print(f"Item {item['id']} has today's promotion date.")

                if fields.get('ContractType') in ['Anapec', 'CDD']:
                    print(f"Item {item['id']} has contract type {fields.get('ContractType')} - updating to CDI.")
                    updates['ContractType'] = 'CDI'

                if fields.get('Level') == 'Junior-Dev 1':
                    print(f"Item {item['id']} has professional status {fields.get('Level')} - updating to Junior-Dev 2.")
                    updates['Level'] = 'Junior-Dev 2'

                if updates:
                    print(f"Updating item {item['id']} with: {updates}")
                    try:
                        response = update_item(headers, site_id, list_id, item['id'], updates)
                        print(f"Update response for item {item['id']}: {response}")
                    except Exception as e:
                        print(f"Failed to update item {item['id']}: {e}")

        print("Check and updates completed.")

    except Exception as e:
        print(f"Error while checking and updating items: {e}")

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

Listing 4.4: Update promotion python script

3 Comparison and Analysis

3.1 Integrity workflow:

In duplicating the approval workflow from Figure 3.4 using the Python code from Listing 4.3, we not only recreated the functionality but also surpassed the average processing time of the workflow. Despite the seamless integration capabilities of Microsoft Power Platform products, Python demonstrated superior efficiency. Unlike the streamlined connectivity of Power Automate, Python required us to handle APIs, credentials, and other technical details to ensure smooth operation.

As depicted in Figure 3.10, the average processing time for Power Automate is significantly higher compared to the Python script timing (shown in Listing 4.3). Specifically, Power Automate's average processing time is visibly longer, indicating less efficiency. In contrast, the Python script's average processing time is notably shorter, emphasizing Python's effectiveness as a programming language in optimizing workflow processes. This contrast highlights Python's superiority in handling complex workflows more efficiently despite the need for more intricate setup and technical configurations.

(a) Power automate timing

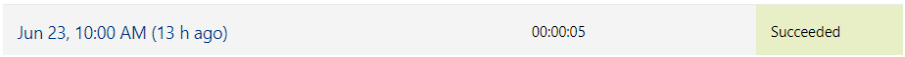
```
Time taken for the check and update process: 1.67 seconds
```

(b) Python script timing

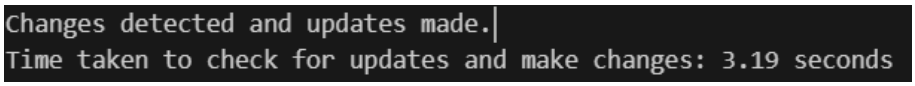
Figure 4.4: Time metrics Comparison

3.2 Promotion update workflow:

In replicating the promotion update workflow using the script from Listing 4.4, we once again achieved superior performance compared to the average processing time of Power Automate (Figure 3.12). Despite the integrated capabilities of Microsoft's Power Platform, which handles API integration, credential management, and other technical intricacies to ensure connectivity and efficiency, our Python script significantly outperformed it. The timing metrics reveal that our Python script took 3.19 seconds to detect and implement changes, our Workflow script executed the same tasks in 5 seconds. This comparison highlights Python's efficiency in optimizing workflow processes, demonstrating its superiority over Power Automate's automation tools.



(a) Power automate timing



(b) Python script timing

Figure 4.5: Time metrics Comparison 2

Conclusion

1 Summary of Findings

The migration process and comparative analysis revealed several significant insights. Firstly, Power Automate demonstrated its strength in providing a user-friendly and accessible platform for workflow automation, particularly for users with no prior coding experience. It successfully streamlined processes, reduced manual errors, and accelerated task completion.

However, when migrating these workflows to Python, notable improvements were observed. Python's robust libraries and advanced data manipulation capabilities enabled the development of more efficient and secure workflows. The comparative analysis showed that Python scripts outperformed Power Automate in terms of processing time and handling of complex tasks. Specifically, Python provided greater customization options, faster execution, and improved scalability, making it a superior choice for users with coding expertise.

Overall, the transition from Power Automate to Python not only retained the initial benefits of automation but also enhanced performance, security, and scalability, demonstrating Python's potential in optimizing and advancing automated workflow solutions. .

2 Scope and Limitations

This research focuses on the design, implementation, and evaluation of automated workflow optimization in SharePoint using Power Automate. The primary goal is to improve time efficiency, reduce errors, and streamline processes. However, several limitations must be acknowledged:

Time Efficiency:

- **Time Constraints:** The implementation and optimization of automated workflows require significant time investment. Initial setup, testing, and troubleshooting can be time-consuming, especially for complex workflows with multiple conditional branches.

Integration Challenges:

- **Connectivity Issues:** Integrating Power Platform tools with SharePoint can present challenges. Ensuring seamless communication between Power Automate and SharePoint often requires careful configuration and troubleshooting of connectivity issues.
- **Third-Party Systems:** Connecting Power Automate and SharePoint to other third-party systems such as the scripts may pose additional difficulties due to API limitations, data format inconsistencies, and security restrictions.

Technical Complexity:

- **Permissions Management:** Configuring permissions in SharePoint or python scripts to ensure appropriate access control while maintaining security is complex. Misconfigured permissions can lead to unauthorized access or hinder workflow functionality.
- **Conditional Logic:** Developing scripts and workflows with advanced conditional logic can be challenging. Handling multiple approval paths, exceptions, and dynamic conditions requires detailed planning and testing.

Scalability:

- **Volume of Data:** As the volume of data and the number of users increase, the automated workflows may require adjustments to maintain performance and efficiency. Scaling the workflows to handle larger datasets and more concurrent users can be challenging.
- **Resource Allocation:** Ensuring that the SharePoint environment has adequate resources (e.g., storage, processing power) to support the automated workflows is essential for maintaining performance.

Despite these limitations, the research aims to demonstrate the significant potential of Power Automate and Python to enhance efficiency, accuracy, and scalability in document management and workflow processes. By addressing these challenges, the project seeks to provide valuable insights and practical solutions for organizations looking to optimize their data management practices.

3 Future Works

For a broader perspective on this project, mastering Python would be ideal for developing a dedicated app for the human resources department. This approach surpasses the limitations of power platforms and incorporates all the advanced features that custom development can offer. By testing on various databases with large volumes and conducting recurrent tests, this study aims to provide a comprehensive understanding of automated software engineering. It equips developers with the knowledge necessary to build a sophisticated desktop application that encompasses detailed employee information, from basic elements to dynamic entities such as leave management. Addressing this gap represents a crucial opportunity to enhance the company's operational success. A robust, custom-tailored app for the department stands as an ideal future prospect.
Considering my personal preferences, I believe that this report, if published, could mark the beginning of a series of comparative studies. It delves deeply into the capabilities of power platforms and critically examines their strengths. While there may not be many articles supporting this claim due to low interest in such studies, I firmly believe, as a developer, that custom code has the potential to rival products from major firms like Microsoft in terms of efficiency, power, and time-saving features.

4 Recommendations

Based on the insights gathered from this study, I recommend the use of Power Platforms, particularly Power Automate, for developing workflows tailored to individuals with no coding background. Power Automate offers a user-friendly interface that simplifies the creation and management of automated workflows, making it accessible for users without technical expertise. This platform streamlines processes, reduces manual errors, and accelerates task completion, as evidenced by its successful implementation in various scenarios outlined in this report.

Additionally, for users with a background in coding, I advocate the use of Python to duplicate and enhance these workflows. Python's robust libraries and powerful data manipulation capabilities enable the development of more efficient and secure workflows. The migration of workflows from Power Automate to Python allows for greater customization, faster processing times, and improved handling of complex tasks. This approach not only maintains the initial benefits observed with Power Automate but also leverages Python's strengths to achieve higher performance and scalability, ensuring that workflows are both rapid and secure.

Bibliography

[1] A. Guilmette, *Workflow Automation with Microsoft Power Automate*. Packt Publishing, 2020.

[2] W. McKinney, *Python for Data Analysis*. O'Reilly Media, 2013.

[3] M. Broschk, "10 challenges and frustrations with microsoft power apps," *14 days - 14 articles*, 2024.