INSTITUTO DE TECNOLOGIA E LIDERANÇA

INFORMATION SYSTEMS

**CAMILA FERNANDA DE LIMA ANACLETO**

# OPTIMIZING ROADSHOW SCHEDULING: ENHANCING EFFICIENCY IN CORPORATE EVENT PLANNING

São Paulo

2025

CAMILA FERNANDA DE LIMA ANACLETO

# OPTIMIZING ROADSHOW SCHEDULING: ENHANCING EFFICIENCY IN CORPORATE EVENT PLANNING

Work presented as the first project of the year on the Automation of Roadshow Schedules, conducted for the Instituto de Tecnologia e Liderança (INTELI), as part of the requirements for professional development and grade acquisition.

Advisor: Prof. Dr. Rafael Will M. de Araujo

São Paulo

2025

## ABSTRACT

This paper presents the development of an automation project using Visual Basic for Applications (VBA) and Python to create automated roadshow schedules for a financial institution. The project is carried out as part of the academic requirements of the Institute of Technology and Leadership (INTELI), where it will be evaluated. The main objective is to optimize the organization and planning of these corporate events, enhancing efficiency through technology and comparing the effectiveness of two distinct programming approaches. The identified problem is the manual management of schedules, which is both laborious and prone to errors. VBA and Python scripts were developed to integrate data from Microsoft Outlook and automatically generate Microsoft Word documents with detailed event information. Research procedures involved requirements analysis, programming in VBA and Python, and iterative testing to ensure system functionality and robustness. The results indicated a significant reduction in the time needed for schedule preparation and management, as well as a decrease in errors. The comparison between the two approaches will provide insights into which is more efficient and adaptable in corporate contexts. This study suggests that process automation is an effective strategy for event organization and can be expanded to other areas of corporate planning.

**Keywords:** Schedule automation. VBA. Python. Technology comparison. Operational efficiency. Event technology. Financial institution.

## TABLE OF CONTENTS

1. INTRODUCTION ........ 10
  1.1 CURRENT SITUATION ........ 10
  1.2 PROPOSED AUTOMATED WORKFLOW ........ 11
  1.3 LIBRARIES ........ 12
  1.3.1 PyWin32 (win32com.client) ........ 12
  1.3.2 python-docx ........ 13
  1.3.3 pytz ........ 13
  1.3.4 datetime (Standard Library) ........ 14
  1.3.5 Flask ........ 15
2. KEY DIFFERENCES BETWEEN USING A PYTHON-BASED APPROACH AND AN VBA-BASED APPROACH ........ 15
3. IMPLEMENTATION IN PYTHON ........ 16
  3.1 Speed and Efficiency ........ 16
  3.2 Scope and Scalability ........ 16
  3.3 Compatibility and Compliance ........ 16
  3.4 Considerations for Multi-Department Collaboration ........ 17
4. IMPLEMENTATION IN EXCEL ........ 17
  4.1 General Description ........ 17
  4.2 Speed and Efficiency ........ 17
  4.3 Scope and Scalability ........ 18
  4.4 Compatibility and Compliance ........ 18
5. COMPARATIVE DISCUSSION ........ 18
  5.1 Security and Governance in Financial Institutions ........ 19
6 DEVELOPMENT CHAPTERS ........ 20
  6.1 Project Initiation ........ 20
  6.2 Development of the VBA Script ........ 20
  6.3 Development of the Python Script ........ 20
  6.4 Testing and Feedback ........ 21
  6.5 Challenges Encountered ........ 21
7 SCHEDULE CREATION PROCESS ........ 22
REFERENCES ........ 24

# 1. INTRODUCTION

This project focuses on the automation of roadshow scheduling using Visual Basic for Applications (VBA) and Python to enhance operational efficiency at a financial institution. The necessity for this research arises from the current demand for improved efficiency and precision in corporate event planning, where manual scheduling methods are both time-consuming and prone to errors.

The theme of this study centers on evaluating and comparing the effectiveness of two distinct programming tools in automating scheduling processes. The choice of this theme reflects its timely relevance and the substantial benefits it promises to bring to the management of corporate events, particularly within financial sectors that are heavily reliant on meticulous organizational capabilities.

The objectives of this research are twofold: to demonstrate the potential reductions in time and errors offered by automation, and to provide a comparative analysis of the programming tools involved. These objectives were chosen to highlight the practical applications of technological advancements in business processes and their impact on operational productivity.

The importance of this research extends beyond simple process improvement, offering significant contributions to the fields of business management and technology by showcasing how automated systems can transform traditional business operations. My engagement with this topic is driven by a deep-seated interest in leveraging technology to solve practical problems, combined with academic pursuits in technology and business process optimization.

The document is structured to guide the reader through the methodology employed in developing the automation scripts, the results of their implementation, and a comprehensive analysis that compares their effectiveness. Concluding remarks will summarize the findings and provide recommendations for future applications of such technologies in business practices.

## 1.1 CURRENT SITUATION

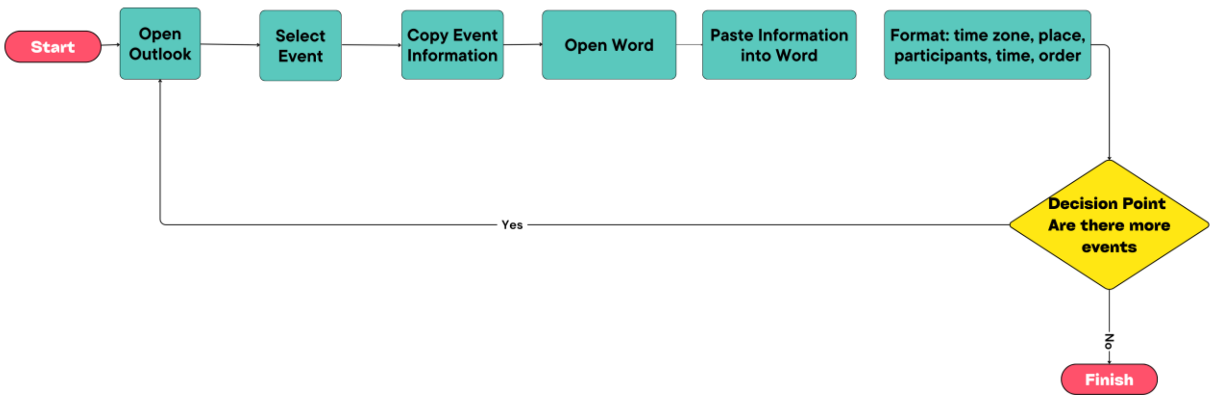
In the current workflow for preparing a roadshow agenda at a financial institution, a significant amount of time is consumed by repetitive manual tasks. First, the user accesses the Outlook calendar to locate each event to be included in the agenda.

Then, relevant information—such as title, participants, schedules, and locations—is copied and subsequently pasted into a Microsoft Word document. This initial transfer of data is followed by a manual formatting process, which entails adjusting time zones (often critical for international engagements), verifying attendees, organizing event sequences chronologically, and ensuring all details are accurate and consistent.

The above-mentioned sequence of steps must be repeated for every single event, compounding the total effort and increasing the likelihood of human error. If updates are required—such as modifications to venues, schedules, or participants, the process must be repeated once again for each affected event. Consequently, this manual approach becomes both time-intensive and prone to inconsistencies, potentially affecting the quality and timeliness of the final agenda.

Below is Figure 1, which illustrates the current manual procedure, highlighting the iterative nature of copying, pasting, and formatting data for each event.

Figure 1 – Current Manual Workflow


Start
Open Outlook
Select Event
Copy Event Information
Open Word
Paste Information into Word
Format: time zone, place, participants, time, order
Decision Point Are there more events
Yes
No
Finish

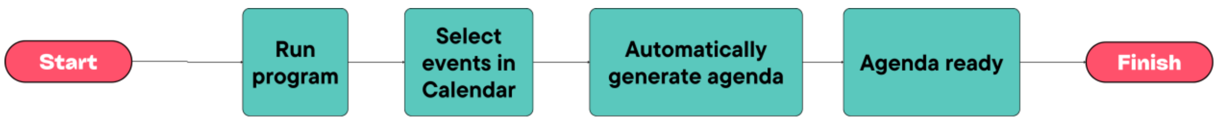
## 1.2 PROPOSED AUTOMATED WORKFLOW

To address these inefficiencies, a software-based solution is proposed. In this improved scenario, once the user runs the program, they only need to select the events intended for the roadshow agenda. The application then automatically retrieves all pertinent information from the Outlook calendar, including event titles, dates, locations, participant lists, and time zones. By consolidating these details, the software standardizes and formats the content, thereby eliminating the repetitive manual steps previously required.

This automated process not only reduces the time spent on mechanical tasks but also minimizes the risk of transcription or formatting errors. Moreover, it enables quick updates: should any event detail change, the user can simply rerun the program or refresh the selection to generate a revised agenda without repeating each manual step. As a result, the institution can allocate resources more effectively, ensuring that staff can focus on higher-level objectives rather than routine data transcription.

Below is **Figure 2**, which demonstrates the proposed automated approach, emphasizing the streamlined workflow for generating a complete and accurate agenda with minimal manual intervention.

Figure 2 – Proposed Automated Workflow



## 1.3 LIBRARIES

This project demonstrates how modern technological tools can streamline complex administrative processes. By integrating web-based interfaces, dynamic document generation, and seamless data retrieval from existing systems, the solution significantly enhances efficiency and user experience. It automates the task of aggregating scheduling information and converting it into professionally formatted documents, representing a vital advancement in reducing manual workload and ensuring consistency in business operations.

### 1.3.1 PyWin32 (win32com.client)

One of the primary challenges in integrating Python with Windows-based applications lies in interfacing with the Component Object Model (COM). The **PyWin32** library, particularly its win32com.client module, presents a robust solution for bridging these environments. In the context of the roadshow agenda automation, PyWin32 enables seamless communication with Microsoft Outlook, thus facilitating direct access to calendar data—such as event titles, participant lists, and scheduling details—without requiring manual intervention.

Through the win32com.client interface, it becomes feasible to automate complex workflows, including querying Outlook folders, iterating through specific calendar items, and extracting field-level information pertinent to each event. This functionality not only decreases the time and effort expended on repetitive tasks but also minimizes the incidence of transcription errors. In an industry setting—particularly in financial institutions handling large volumes of roadshow events, this capability can markedly enhance operational efficiency and data reliability. By using PyWin32 to bridge Outlook and Python, developers can create scripts that operate in near real-time, updating or generating agendas with a level of consistency in achievable by manual processes.

### 1.3.2 python-docx

In an automated pipeline that involves producing human-readable documents, the **python-docx** library emerges as an indispensable asset. While raw data extraction and analysis can be performed via numerous Python modules, the final output in many corporate environments still relies heavily on Word documents (.docx). Python-docx allows developers to programmatically create, manipulate, and format these files, alleviating the burden of manually copying and pasting large segments of text or table entries.

For the proposed roadshow agenda generator, python-docx handles critical tasks such as inserting event details, adjusting text styles, and formatting tables to align with the institution’s official templates. This consistency in layout and design is especially relevant for institutions that demand professional and uniform reports, thereby conferring an added layer of credibility to their output. Moreover, python-docx abstracts away much of the complexity inherent in the .docx file structure, offering a straightforward application programming interface (API) that saves time and reduces the need for specialized document editing software.

### 1.3.3 pytz

Given the global nature of many financial roadshows, pytz plays a crucial role in ensuring accurate time zone conversions across multiple regions. Mistakes in time zone handling can lead to scheduling conflicts, missed meetings, or erroneous event listings—problems that bear significant cost in high-stakes financial settings. By embedding pytz within the automated workflow, the system can dynamically adjust event times according to the appropriate local time zone, thus enhancing the precision and trustworthiness of the final agenda.

One of the most significant advantages of this automated scheduling solution is precisely its built-in time zone management, which becomes especially critical when roadshow events span multiple countries or continents. By automatically converting each event's start and end times to the relevant local time zone, the system eliminates the risk of human error that often arises from manual calculations. Additionally, the automation script detects and accounts for Daylight Saving Time (DST) transitions, ensuring that all participants—regardless of location—receive accurate and timely updates. This level of precision not only streamlines the coordination process but also reinforces the institution's professional image by delivering well-organized agendas that seamlessly adapt to global scheduling demands.

Furthermore, pytz integrates seamlessly with Python's standard datetime module, granting developers fine-grained control over the creation, conversion, and manipulation of time objects. This robustness goes beyond simply reconciling disparate time zones, as it also supports more nuanced logic—such as automatically applying local holidays or adjusting for exceptional time shifts if needed. As a result, **pytz** is particularly valuable for an institution whose events might occur in multiple global financial centers, each subject to distinct local time regulations.

### 1.3.4 datetime (Standard Library)

Although part of Python's standard library, **datetime** should not be overlooked in discussions of time-related functionality. In synergy with pytz, the datetime module underpins the program's ability to store, manipulate, and calculate date and time data with reliability. It allows for arithmetic operations on timestamps, validation of chronological order for a sequence of events, and straightforward formatting to meet specific display requirements (e.g., “YYYY-MM-DD HH:MM”).

For the financial roadshow agenda, datetime facilitates the creation of precise timestamps reflecting each event's scheduled start and end times. Additionally, it simplifies interval calculations, such as determining the total duration of a particular session or computing the overlap between consecutive events. In doing so, it mitigates the need for extensive custom parsing or time arithmetic, thereby enhancing code clarity and maintainability.

#### 1.3.5 Flask

Flask is a lightweight web framework written in Python that plays a pivotal role in the development of this automated scheduling system. As a micro-framework, Flask offers a minimal yet powerful set of features, allowing for the rapid development of web applications without the overhead typical of more comprehensive frameworks. Its simplicity enables seamless integration with Python, facilitating direct access to Python’s extensive libraries and tools while maintaining a clear and concise application architecture. A key advantage of Flask is its cross-platform compatibility, which ensures that the server can operate on any machine regardless of the underlying operating system. This characteristic not only aids in the prototyping and testing phases but also guarantees that the final application can be deployed in diverse computing environments. Moreover, Flask's flexibility and modular design allow developers to extend the application's functionality through well-integrated extensions, thereby supporting iterative and agile development practices. In the context of this project, Flask serves as the backbone for the web-based interface that enables users to interact with the system. Its lightweight nature reduces system resource consumption while providing the necessary routing and template rendering capabilities required for dynamically generating schedules. By streamlining the communication between the user interface and the backend processes—such as data extraction from Outlook and document generation—Flask significantly enhances both the usability and efficiency of the system. Overall, the adoption of Flask has not only simplified the architectural design of the application but has also contributed to a more agile development cycle, ensuring that the solution remains adaptable and scalable to meet evolving user requirements.

## 2. KEY DIFFERENCES BETWEEN USING A PYTHON-BASED APPROACH AND AN VBA-BASED APPROACH

The creation of agendas for roadshows in financial institutions often involves repetitive and error-prone processes, especially when executed manually. Seeking to optimize the workflow, two approaches were developed: one based on **Python**, and the other utilizing **Excel**. Both solutions aim to reduce manual effort and enhance the reliability of the resulting information, yet they differ in terms of execution speed, flexibility, scalability, and adherence to specific requirements within the banking environment.

## 3. IMPLEMENTATION IN PYTHON

The **Python**-based version employs libraries such as **PyWin32**, **python-docx**, and **pytz**, enabling automated data extraction from Microsoft Outlook, time-zone conversions, and the generation of properly formatted Word documents. This ecosystem of tools facilitates fast and reliable processing of large data volumes, minimizing human intervention and reducing transcription errors. For scenarios involving multiple events or participants, Python's capabilities stand out due to its wide array of packages that streamline data manipulation and automation.

### 3.1 Speed and Efficiency

One of Python's key advantages lies in its **efficiency for data handling**. The language allows users to read, process, and output reports quickly, particularly if data structures such as DataFrames (through **pandas**) are utilized for large-scale data manipulation. Integration with **APIs** and potential support for asynchronous or parallel tasks (when applicable) contribute to significant performance improvements, especially in complex scenarios or when dealing with a high volume of events.

### 3.2 Scope and Scalability

Thanks to Python's extensive library ecosystem and its ease of integration with databases, web services, and other corporate systems, the project in Python tends to be **more scalable and comprehensive**. For instance, it can be expanded to accommodate various data sources, generate additional reports, or even incorporate predictive analytics. This flexibility is particularly relevant in large-scale environments where demands can change rapidly and may involve multiple departments or units within the financial institution.

### 3.3 Compatibility and Compliance

Regarding **compatibility with a bank's system**, Python can offer substantial advantages. Many financial institutions maintain internal servers where setting up Python environments and necessary libraries is feasible, in accordance with security requirements (e.g., segregating internal and external networks). Python also has **well-established libraries for security and encryption**, facilitating compliance with internal regulations and standards. Nevertheless, adopting this approach requires the institution to have personnel proficient in Python and knowledgeable about corporate security protocols.

### 3.4 Considerations for Multi-Department Collaboration

Implementing Python- or Excel-based solutions in a financial institution rarely occurs in isolation. Typically, multiple departments—ranging from Investor Relations to IT Security—are involved. In some cases, the Marketing or Corporate Communications team also collaborates to ensure that brand guidelines and communication standards are upheld in the final roadshow documents. These cross-functional interactions add layers of complexity to the project management aspect, requiring careful change management strategies. For instance, **Kotter's 8-Step Model** suggests that clear communication of benefits and potential impacts fosters broader acceptance of new technologies in traditionally conservative environments.

Furthermore, the need to reconcile different software policies can arise. While certain divisions may already use Python for data analytics, others may strictly rely on Microsoft Office applications due to licensing, training, or security constraints. Therefore, choosing between a Python-based or Excel-based solution might involve practical considerations, such as existing skill sets and the scope for future expansion.

## 4. IMPLEMENTATION IN EXCEL

### 4.1 General Description

The second version of the project uses **Microsoft Excel** as the foundation for automation, often through **VBA** (Visual Basic for Applications) macros or via **Power Query** and **Power Automate**. This choice is frequently driven by the **familiarity** most users already have with Excel and by its ease of deployment in environments where installing additional software may be restricted.

### 4.2 Speed and Efficiency

While Excel's automation features via VBA can streamline certain repetitive tasks, its **processing speed** is generally **lower** than that of solutions built in programming languages like Python. For smaller or medium-scale datasets, performance remains acceptable; however, as the number of events (and associated spreadsheets) increases, the tool may face **performance bottlenecks** and a higher likelihood of freezes or errors.

### 4.3 Scope and Scalability

In terms of **scope**, Excel-based solutions are typically more limited, since the platform was originally designed for tabular data manipulation and relatively static reporting. Although it can integrate with other platforms (such as internal databases and Outlook), the degree of automation via Excel is usually less flexible. Scalability also becomes challenging, as handling large volumes of data or more complex queries often require advanced VBA implementations, which can render the codebase **difficult to maintain** and more **susceptible to failures**.

### 4.4 Compatibility and Compliance

In a **financial institution** context, Excel initially presents certain benefits, such as widespread acceptance, longstanding presence in corporate environments, and **user-friendly** features. Most departments already have Excel installed, with macros enabled in a controlled manner. However, **from a compliance perspective**, macros can pose greater vulnerabilities if not rigorously managed. Moreover, tracking and governing the underlying code can become complex, particularly if various users create or modify macros locally without a structured version control process.

## 5. COMPARATIVE DISCUSSION

Comparing both solutions reveals that the **Python-based version** tends to offer:

1. **Higher speed** of data processing, especially in large-scale scenarios.
2. **Greater efficiency** in automation, thanks to specialized libraries that enable deeper integration with other systems, along with straightforward code maintenance in version control repositories (e.g., Git).
3. **Broader scope**, given Python’s ecosystem, which extends from machine learning to data visualization, allowing future project expansions.

4. **Significant compatibility** with banking infrastructure, as long as the institution supports the secure deployment of Python scripts in compliance with regulatory standards.

By contrast, the **Excel-based version** may be attractive because:

1. It offers **user-friendly adoption**: many individuals are already proficient in Excel, reducing the initial learning curve.
2. It ensures **immediate compatibility**: nearly all workstations in financial institutions have Excel installed, with macro functionality properly configured.
3. It requires **less infrastructure setup**: in some cases, there is no need to install Python or external libraries, simplifying initial deployment.

Nevertheless, the Excel solution exhibits limitations in **speed**, **scalability**, and **ease of maintenance**, particularly as the project grows in complexity or data volume. Moreover, macro usage demands heightened vigilance regarding security and governance.

### 5.1 Security and Governance in Financial Institutions

Automation in a financial context must comply with stringent security protocols. ISO 27001 standards, for instance, guide information security management systems (ISMS) and are often mandated in regulated industries. Any tool that interacts with emails, calendars, or file systems—such as Python scripts using PyWin32 or macros embedded in Excel—must align with these security measures to protect sensitive client data and proprietary information.

Governance also plays a significant role. Many institutions enforce policies where macros or Python scripts must be digitally signed or approved by an internal review board. While this may slow deployment, it ensures that any code changes are vetted for potential vulnerabilities. In some cases, containerization (via Docker) or virtualization environments (like Citrix) can sandbox the automation, confining potential risks.

# 6 DEVELOPMENT CHAPTERS

This section details the initial stages of developing automation scripts utilizing Visual Basic for Applications (VBA) and Python. The primary accomplishment thus far includes the automation of extracting event information from Microsoft Outlook and its subsequent formatting into Microsoft Word documents. This foundational step is integral to a broader project aimed at enhancing the efficiency of scheduling roadshows for a financial institution.

## 6.1 Project Initiation

The inception of the project involved a clear definition of scope and objectives, focusing initially on the automation of data retrieval from Outlook calendars and the generation of structured Word documents. This task was identified as a critical bottleneck in the efficient management of roadshows, where manual data entry frequently led to operational inefficiencies and increased error rates.

## 6.2 Development of the VBA Script

In addressing the needs identified, a VBA script was meticulously developed to interface directly with Microsoft Outlook. Utilizing the capabilities inherent to VBA's integration with Outlook, the script facilitates the automated extraction of essential details such as the date, time, location, and participant information for each event. This data is then seamlessly populated into a Word document template, formatted to meet institutional standards for clarity and professionalism.

The development process for the VBA script encompassed several key activities: accessing Outlook calendar objects to retrieve upcoming events, applying logic to filter and sort pertinent event details, and automating the transfer of this information into a Word document to compile the extracted data effectively.

## 6.3 Development of the Python Script

Concurrently, a Python script was developed to mirror the functionalities offered by the VBA script, with enhancements to ensure greater flexibility and scalability. The script employs the exchangelib library for Outlook connectivity and python-docx for dynamic Word document management, establishing a robust framework for handling data.

Significant efforts in the Python script development included configuring the Outlook connection to fetch calendar entries reliably, utilizing python-docx to dynamically create and format Word documents based on the retrieved data, and embedding comprehensive error handling mechanisms to address potential issues such as missing data or connectivity interruptions.

### 6.4 Testing and Feedback

Both scripts were subjected to initial testing phases to validate basic functionalities, ensuring accurate data extraction and documentation. Feedback from these sessions has been instrumental in pinpointing areas for improvement, guiding subsequent refinements to enhance both scripts' functionality and user experience.

### 6.5 Challenges Encountered

During the initial stages of both the VBA and Python script implementations, several challenges were identified:

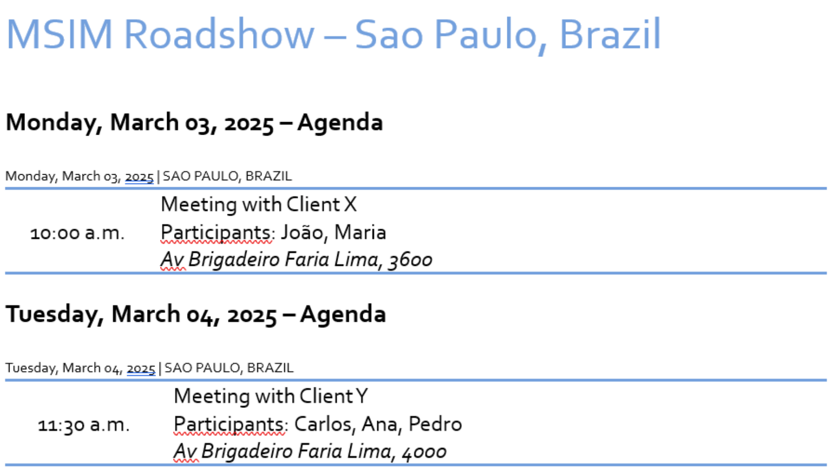
- Connectivity Issues: In environments with stringent firewall settings, establishing a stable connection to Outlook’s COM objects or Exchange servers may demand specialized configurations.
- Date/Time Formatting: Variations in regional settings can cause Python scripts or VBA macros to misinterpret date and time fields, resulting in scheduling inaccuracies.
- Document Standardization: Aligning the generated Word documents with corporate branding guidelines sometimes requires custom styling or formatting, which adds complexity to the code.
- User Adoption: Despite automation’s advantages, some employees may resist transitioning away from familiar manual processes, especially if they perceive the new system as complex or fear job displacement.

Addressing these challenges involves a combination of technical refinements (e.g., robust error handling, flexible configuration files) and organizational strategies (e.g., training sessions, clear documentation, and stakeholder engagement).

# 7 SCHEDULE CREATION PROCESS

The development of an automated scheduling system represents a significant innovation in the management of corporate events and meetings. This process is designed to transform traditional manual agenda creation into a highly efficient, automated procedure that ensures consistency and precision. Initially, the system allows the user to define a specific date and time interval. Based on this interval, the application accesses the integrated calendar system and retrieves all pertinent event data, including the event title, start and end times, location, and the list of participants. This automated extraction substantially reduces human error and minimizes the time spent on manual data entry, directly addressing common inefficiencies in conventional scheduling methods. Subsequently, the retrieved event data is organized and grouped according to predefined criteria. A key feature of the system is its use of a dynamic document template that contains placeholders for data substitution. These placeholders (e.g., for the event title, time, and location) are systematically replaced with the actual event information retrieved from the calendar system. This process ensures that the final document maintains a consistent and professional format, which is crucial for clear communication and record-keeping in a corporate environment. As illustrated in Figure 4, the initial template includes clearly marked placeholders indicating where specific event details are to be inserted. Upon execution of the data replacement process, the system generates the final schedule document, as exemplified in Figure 3. In this final document, every placeholder is accurately substituted with the corresponding data, resulting in a comprehensive and polished agenda. Furthermore, the project integrates a web-based interface developed using the Flask framework. This interface, depicted in Figure 5, enables users to easily input the desired date and time intervals, view the list of events retrieved from the calendar system, and select the events to be included in the final agenda. The user-friendly design of the Flask application further emphasizes the system’s commitment to operational efficiency and accessibility. In summary, the proposed system is a paradigm shift from manual agenda assembly to a fully automated framework that leverages modern technological advances to simplify routine administrative tasks. The resultant efficiency gains, increased reliability, and reduction in error rates underscore the practical impact of adopting such an automated solution in contemporary organizational settings.

Figure 3 – Final Agenda



MSIM Roadshow – Sao Paulo, Brazil

**Monday, March 03, 2025 – Agenda**

Monday, March 03, 2025 | SAO PAULO, BRAZIL

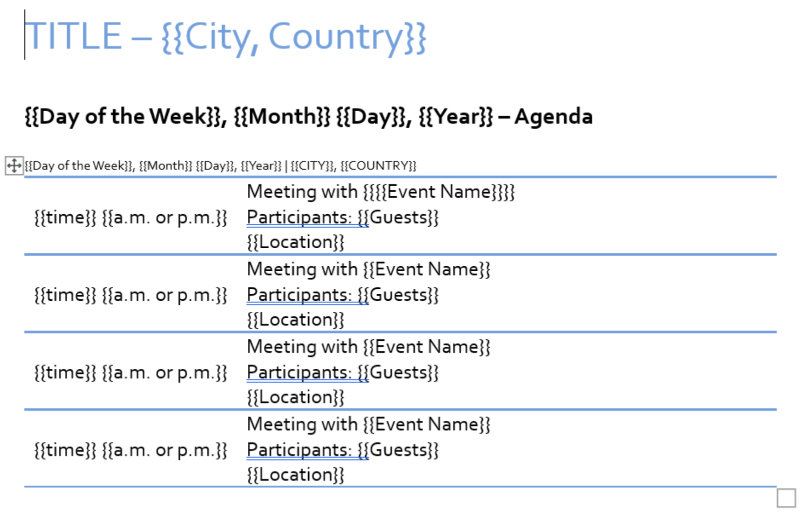
| | |
|---|---|
| 10:00 a.m. | Meeting with Client X<br>Participants: João, Maria<br>*Av Brigadeiro Faria Lima, 3600* |

**Tuesday, March 04, 2025 – Agenda**

Tuesday, March 04, 2025 | SAO PAULO, BRAZIL

| | |
|---|---|
| 11:30 a.m. | Meeting with Client Y<br>Participants: Carlos, Ana, Pedro<br>*Av Brigadeiro Faria Lima, 4000* |

Figure 4 – Agenda with Placeholders



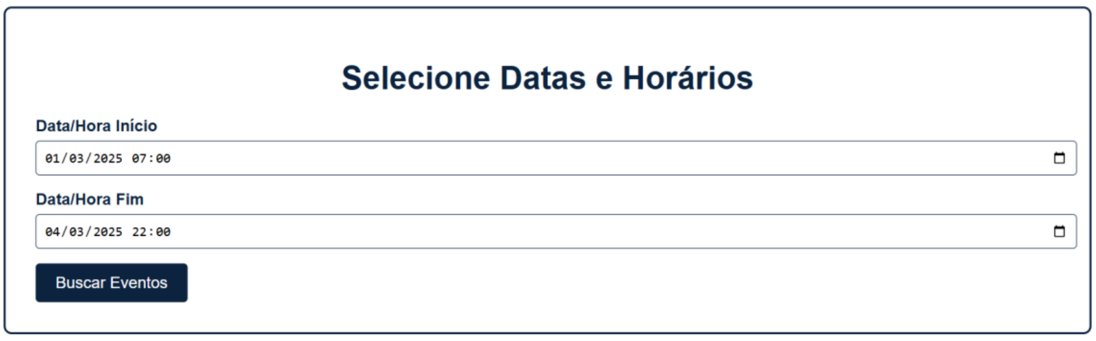
TITLE – {{City, Country}}

**{{Day of the Week}}, {{Month}} {{Day}}, {{Year}} – Agenda**

{{Day of the Week}}, {{Month}} {{Day}}, {{Year}} | {{CITY}}, {{COUNTRY}}

| | |
|---|---|
| {{time}} {{a.m. or p.m.}} | Meeting with {{{{Event Name}}}}<br>Participants: {{Guests}}<br>{{Location}} |
| {{time}} {{a.m. or p.m.}} | Meeting with {{Event Name}}<br>Participants: {{Guests}}<br>{{Location}} |
| {{time}} {{a.m. or p.m.}} | Meeting with {{Event Name}}<br>Participants: {{Guests}}<br>{{Location}} |
| {{time}} {{a.m. or p.m.}} | Meeting with {{Event Name}}<br>Participants: {{Guests}}<br>{{Location}} |

Figure 5 – Flask Application Interface



Selecione Datas e Horários

Data/Hora Início

01/03/2025 07:00

Data/Hora Fim

04/03/2025 22:00

# REFERENCES

BRAZILIAN ASSOCIATION OF TECHNICAL STANDARDS. ABNT NBR 6023: information and documentation: references: preparation. Rio de Janeiro: ABNT, 2018.

GRINBERG, M. Flask Web Development: Developing Web Applications with Python. 2nd ed. Sebastopol, CA: O'Reilly Media, 2018.

PYTHON SOFTWARE FOUNDATION. Python 3 Documentation. [S.l.]: Python Software Foundation, 2023. Disponível em: https://docs.python.org/3/. Acesso em: 9 abr. 2025.

BLANKENSHIP, J. python-docx: A Python Library for Creating and Updating Microsoft Word (.docx) Files. [S.l.], 2016. Disponível em: https://python-docx.readthedocs.io/en/latest/. Acesso em: 2 abr. 2025.

PYWIN32. Pywin32 Documentation. [S.l.], 2023. Disponível em: https://github.com/mhammond/pywin32/. Acesso em: 16 mar. 2025.

MICROSOFT. Office VBA Reference. Microsoft, 2023. Disponível em: https://docs.microsoft.com/en-us/office/vba/api/overview/. Acesso em: 18 mar. 2025.

PRESSMAN, R. S. Software Engineering: A Practitioner's Approach. 8th ed. New York: McGraw-Hill, 2014.

MARTIN, R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Upper Saddle River, NJ: Prentice Hall, 2017.