# Système d'Information d'Entreprise
# Integration Project
# Printemps 2024
# Automating order processing via Odoo and FakeSMTP

Zeynep Caysar

April 2024

## Contents

# 1 Introduction

In a time when being fast and efficient is very important, a medium-sized company that makes custom diecast parts saw a big need to change. They were stuck in a lot of manual, or hand-done, work for handling orders, which was slowing them down and making customers unhappy. This project planned to make order processing automatic and more connected by using Odoo's ERP system, with the help of a special Python program to sort through emails.

# 2 Analysis

The company is a mid-sized manufacturer known for making custom diecast parts. They produce smooth and textured-surface metal parts for service areas like automotive, electronic communication, work machines, hand tools, pumps and valves.

Sales and inventory departments are primarily concerned.

The company uses a hybrid IT infrastructure, with critical systems hosted on-site (for control and security reasons) and non-critical systems in the cloud. The environment is primarily Linux-based for its reliability and security features.

Figure 1: die-casting products, source

Currently, the company uses a mix of spreadsheets for order tracking and customer management. They are open to adopting new software that can integrate with their existing tools and support their growth.

It's in a competitive market where doing things quickly and accurately really matters, and keeping customers happy is key to staying ahead. However, the company has been struggling with slow and error-prone manual processes for handling orders. This has led to mistakes, delays, and customers not being as satisfied as they could be. To keep growing and maintain a good relationship with their customers, the company recognized the urgent need to improve how it processes orders. They wanted to move away from doing things by hand to

using automated systems that are faster, more accurate, and can help make their customers happier.

At the beginning of the project, the information is given about how the company currently manages its orders, checks inventory, and communicates with customers—all of which were done manually. The order management involved employees manually checking emails for new orders and then entering these orders into spreadsheets. Inventory checks were also done by hand, with staff looking through spreadsheets to see if enough items were available to fulfill the orders. For customer communication, employees would manually send updates and respond to inquiries via email or phone, which was not only time-consuming but also increased the chance of missing or delaying important updates. This manual system was clearly slowing things down and leading to errors that affected customer satisfaction.

To move from the manual processes to an automated system, the first crucial step was to fully understand and document the current way of doing things, known as the AS-IS process. This meant closely looking at each step involved in order management, inventory checks, and customer communication to see exactly where improvements could be made.

## 2.1 AS-IS process model

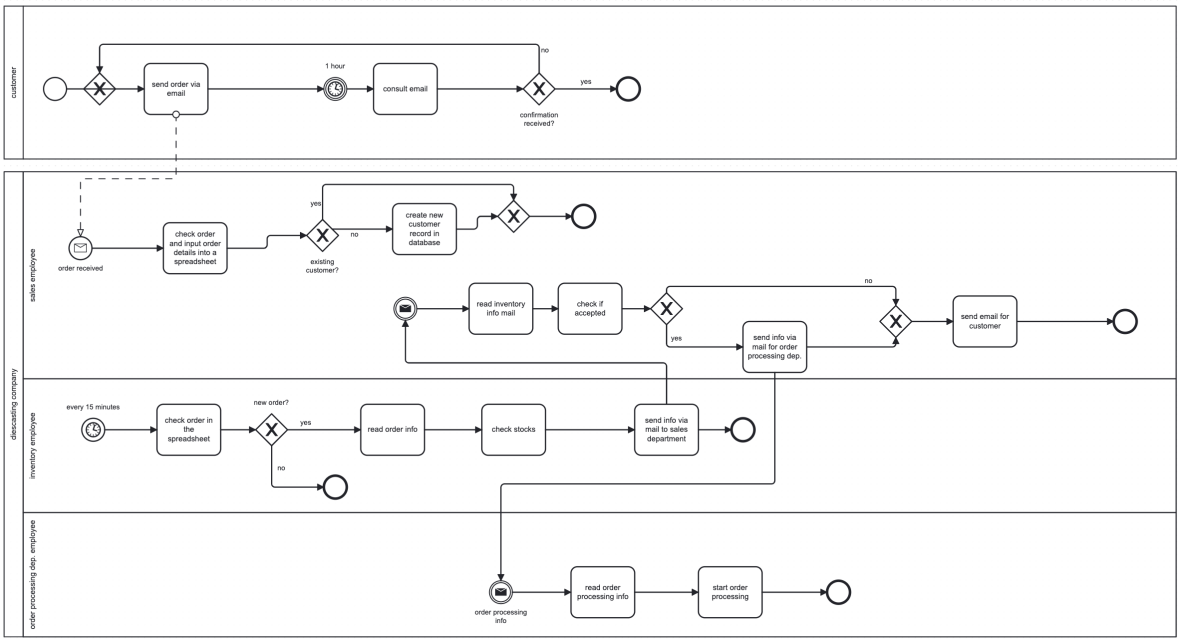



Figure 2: AS-IS BPMN

Flow of Customer process:

First, they send an order via email directly to the company. There is no automated system in place for capturing or processing these orders. They wait 1 hour to get confirmation. They consult if there are any email. If 1 hour is passed and they got no confirmation, they resend a new email about their order. If the confirmation is received, the process finishes.

Flow of the die-casting company:

Sales staff manually check the company's email to find and read new orders. They then manually input order details into a spreadsheet for tracking purposes. They check if the customer is new or not. If it's a new customer, the sales employee create a new record in the customer database. This process is prone to human error and is time-consuming.

When the inventory department's message come, they read the message. They check if the order is accepted, if accepted, they send a mail to order processing department and send a mail to customer. If not accepted, they send

their decision to customer via email.

Every 15 minutes, an Inventory employee manually checks if there are any new orders in the order spreadsheet. If there is a new order, they read the order info, check the stocks and send the information mail to the sales department.

This process can lead to delays in order processing, especially if inventory levels are not up to date or if communication between departments is slow.

When the order processing department gets a mail about a new order, they read the order info and start order processing.

Customers are informed about their order status through manual follow-ups. This could include confirmation of the order, updates on processing, and shipping details. Because this process is manual, it's not only time-consuming but can also lead to inconsistencies in customer communication.

Based on manual checks for inventory and customer records, the Sales department initiates order processing. This could involve further internal communications, manual creation of shipping documents, and finally, updating the order tracking spreadsheet.

## 2.2 Challenges in the AS-IS Model

Efficiency: The manual handling of orders, from email retrieval to entry into spreadsheets, is time-consuming and prone to error.

Scalability: As the company grows, the manual processes will become increasingly unsustainable.

Customer Satisfaction: Delays in order processing and potential errors can negatively impact customer satisfaction.

Data silos: With information spread across emails, spreadsheets, there's a high risk of data being out-of-date or inconsistent across departments.

This AS-IS business model demonstrates the need for an automated solution to streamline operations, reduce errors, and improve customer satisfaction.

The TO-BE process aims to address these challenges by introducing automation and integration across the departments through the implementation of Odoo ERP and custom Python scripting for email parsing and automation.

# 3 Proposition

My approach to automating order processing integrates a Python script that interfaces with FakeSMTP to capture order emails, parses them for order details, send automatic emails to customer and supplier, and utilizes Odoo's API to interact with its Inventory and Sales modules. This automated system ensures that orders are processed efficiently and accurately, minimizing human error and improving customer service by providing timely order confirmations.

## 3.1 TO-BE process model

Every 1 minute, Python script checks the email folder if there are any new emails.

Mails are received via email, captured by FakeSMTP during testing phases to simulate real-world email receipt.

It parses emails to extract order details. It checks if it's a new order or a reply mail to insufficient stock mail of the company.

If it's a new order mail, it reads the order details, it checks if the format is correct. If it's not correct, it informs the customer. If it's correct, it checks if the product available. If not, the case where they don't produce this product, it informs the customer. If its available, it checks the stocks. If the stocks are not sufficient, it informs the customer. If the stocks are sufficient, it creates an order record in odoo and it informs the customer and checks if the customer is an existing customer, if not, it creates a record in customer database.

If it's a reply mail to insufficent stocks, the script checks the mail, there could be 3 answers, if the answer is 1,which is a demand for a new order with available products, it reads the order details, if it's 2, which is a demand for waiting until product is replenished, it informs the customer and supplier with mail about Request for Product Replenishment, if it's 3, which is a demand for cancellation order, it informs the customer about order cancellation.

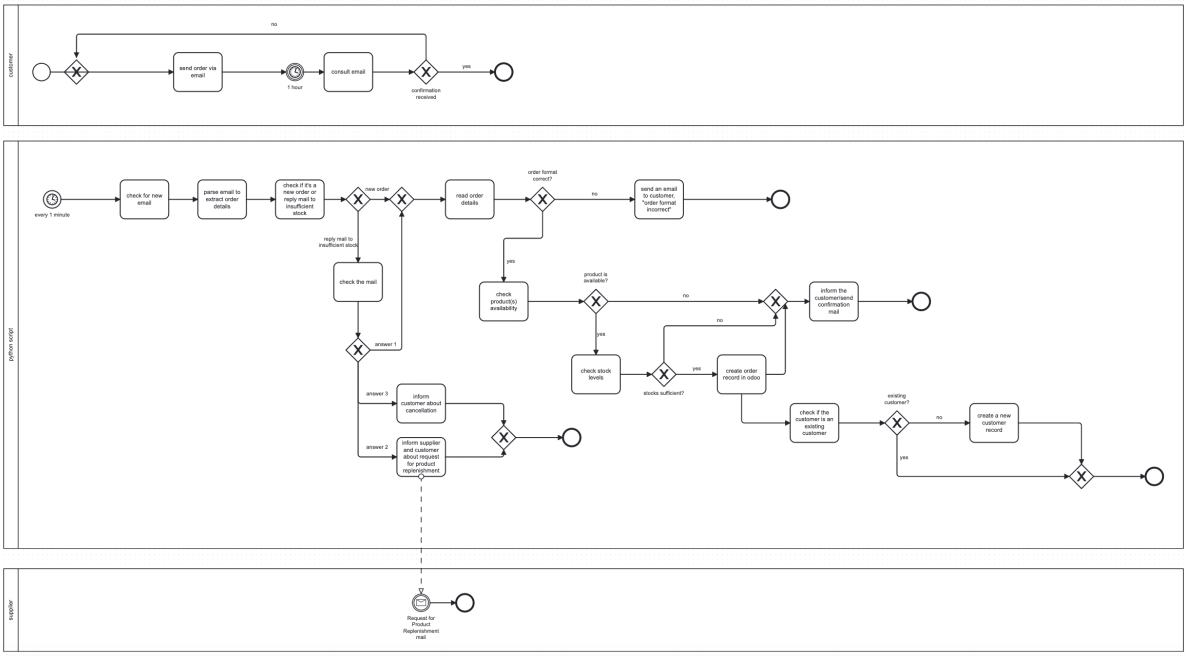



Figure 3: TO-BE BPMN

This approach leverages automation to reduce workload and human error in the order processing workflow. By integrating directly with existing systems like Odoo and using email as the primary order input, the solution is both scalable and adaptable to various business sizes and types. Moreover, it enhances customer satisfaction through faster processing and accurate communication.

## 3.2 Implementation

These are the softwares and technologies I used to build this new process:

Python: For scripting the automation process.

FakeSMTP: Used during development to test email capturing without needing real email traffic.

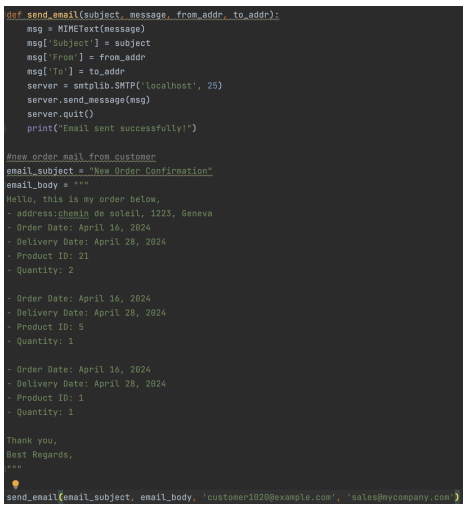
SMTP Protocol: Used for sending confirmation emails through the Python smtplib.

Odoo ERP: Central management software that handles sales, and inventory.

Odoo API: RESTful API for accessing and manipulating Odoo's database, particularly for creating sales orders, creating and updating customer records and checking inventory.

## 3.3 Code Review and Results

To test the project, first we need to send email(s) to the company like they receive it from the customer(s).
We use FakeSMTP to send emails to imitate real world email sending or receiving.



```
def send_email(subject, message, from_addr, to_addr):
    msg = MIMEText(message)
    msg['Subject'] = subject
    msg['From'] = from_addr
    msg['To'] = to_addr
    server = smtplib.SMTP('localhost', 25)
    server.send_message(msg)
    server.quit()
    print("Email sent successfully!")

#new order mail from customer
email_subject = "New Order Confirmation"
email_body = """
Hello, this is my order below,
- address:chemin de soleil, 1223, Geneva
- Order Date: April 16, 2024
- Delivery Date: April 28, 2024
- Product ID: 21
- Quantity: 2

- Order Date: April 16, 2024
- Delivery Date: April 28, 2024
- Product ID: 5
- Quantity: 1

- Order Date: April 16, 2024
- Delivery Date: April 28, 2024
- Product ID: 1
- Quantity: 1

Thank you,
Best Regards,
"""

send_email(email_subject, email_body, 'customer102@example.com', 'sales@mycompany.com')
```

Figure 4: send_emails function

send_emails function takes an email subject, an email body, a from and a to address and it sends it via fakesmtp server on port 25.

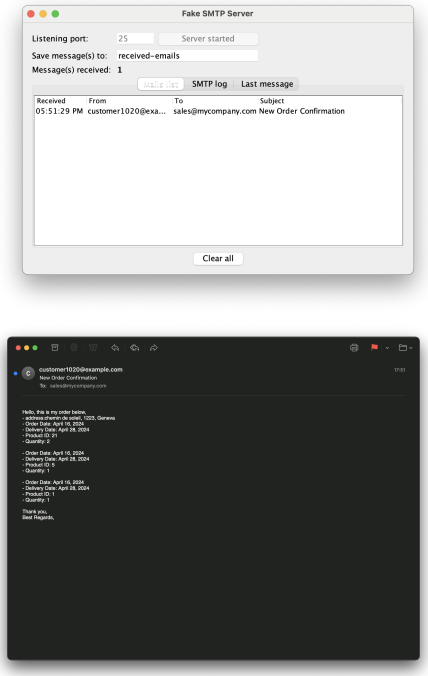



Figure 5: customer sends an email

Multiple new mails by a customer or different customers can be received.

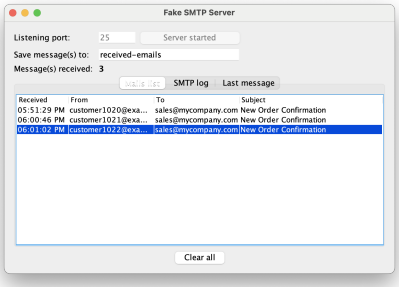



Figure 6: multiple emails in the email folder

These are the available products. We can access by running the odoo_api.py file.

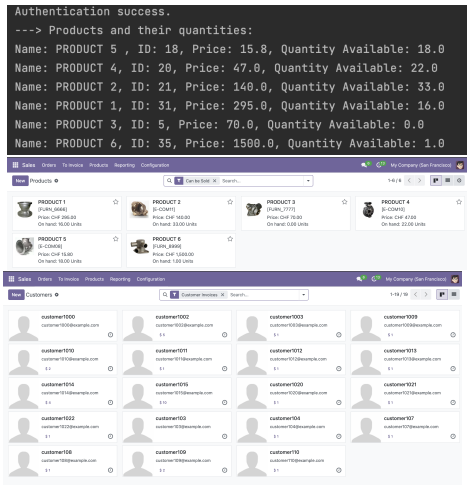



Figure 7: available products and customers in odoo

The main code is located in the parse.py file. When we run this file, we start by the process_emails function who triggers the execution.

```
def process_emails(directory, last_run_file):
    last_run_time = get_last_run_time(last_run_file)
    current_time = datetime.now()

    for file_path in Path(directory).rglob('*.eml'):
        file_mtime = datetime.fromtimestamp(os.path.getmtime(file_path))
        if file_mtime > last_run_time:
            with open(file_path, 'rb') as file:
                msg = BytesParser(policy=policy.default).parse(file)
                parse_email(msg, file_path)
                #print(f"Processed {file_path}")

    update_last_run_time(last_run_file, current_time)
```

Figure 8: process_emails function

It scans a directory for email files, checks if they have been modified since the last run, and processes each relevant email to manage orders or handle stock issues.

It retrieves the last time the script was run using get_last_run_time who fetches the last timestamp when the script was run from a file, which helps determine which emails are new or changed since the last script execution.

It then iterates over all email files in a specified directory, opening and parsing those modified after the last run time.

For each email, it calls parse_email to extract and process the order details or manage stock replies.

Finally, it updates the last run timestamp using update_last_run_time to log the current processing completion.

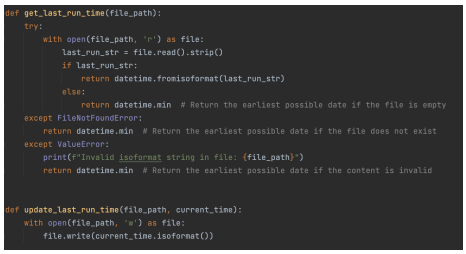



Figure 9: get_last_run_time and update_last_run_time functions

parse_email function analyzes individual email files, extracts meaningful content based on sender and subject, and decides the subsequent action (e.g., order processing or handling stock issues). It extracts the email body using extract_body who retrieves the text content from the email, handling both plain text and multipart messages. After the orders details are received, process_order function is called.


Figure 10: parse_email function


Figure 11: extract_body function

Depending on the subject or sender, it might call handle_stock_reply for handling stock responses or loop through potential order details within the email body, calling extract_order_details for each segment.

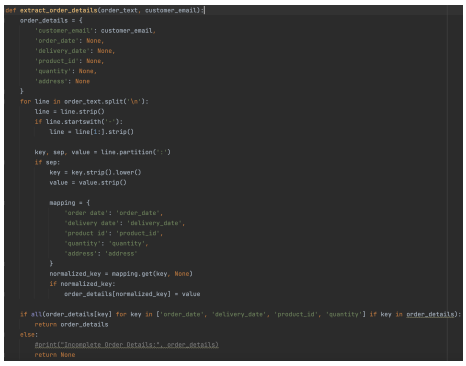


Figure 12: extract_order_details function

process_order processes an order by formatting dates and creating the order in Odoo if all conditions (like date format and product availability) are met.

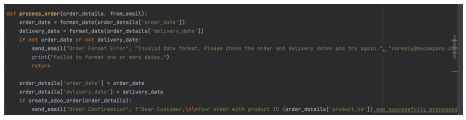


Figure 13: process_order function

Figure 14: format_date function

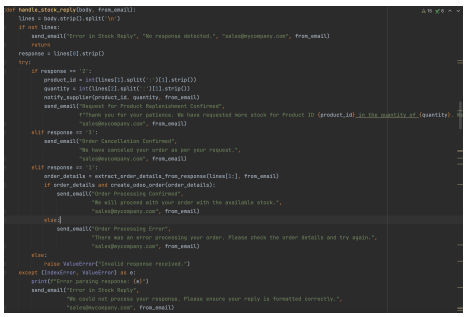


Figure 15: handle_stock_reply function

If the subject of the email is "reply: insufficient stocks", handle_stock_reply is called. It processes replies specific to stock inquiries, such as confirming product

replenishment, canceling orders, or adjusting orders based on stock availability. Depending on the response type (coded as numbers in the email), it may call notify_supplier to inform about stock needs (case 2), send_email to communicate with the customer, or extracting order details with extract_order_details_from_response function and processing an order with create_odoo_order if the decision involves proceeding with an available order (case 1).

```
def extract_order_details_from_response(lines, customer_email):
    order_details = {'customer_email': customer_email}
    for line in lines:
        clean_line = line.strip().lstrip('-').strip()
        key, _, value = clean_line.partition(':')
        normalized_key = key.strip().lower().replace(' ', '_')
        if normalized_key in ['address', 'order_date', 'delivery_date', 'product_id', 'quantity']:
            order_details[normalized_key] = value.strip()

    required_fields = ['order_date', 'delivery_date', 'product_id', 'quantity', 'address']
    if all(order_details.get(field) for field in required_fields):
        try:
            order_details['order_date'] = format_date(order_details['order_date'])
            order_details['delivery_date'] = format_date(order_details['delivery_date'])
            if order_details['order_date'] and order_details['delivery_date']:
                return order_details
        except ValueError as e:
            print(f"Date formatting error: {e}")

    return None
```

Figure 16: extract_order_details_from_response function

notify_supplier function manage communication via email, to notify suppliers about stock needs.

```
def notify_supplier(product_id, quantity, from_email):
    message = f"We need more stock for Product ID {product_id}, Quantity Required: {quantity}. Please confirm availability."
    send_email("Request for Product Replenishment", message, "sales@mycompany.com", "supplier@example.com")
```

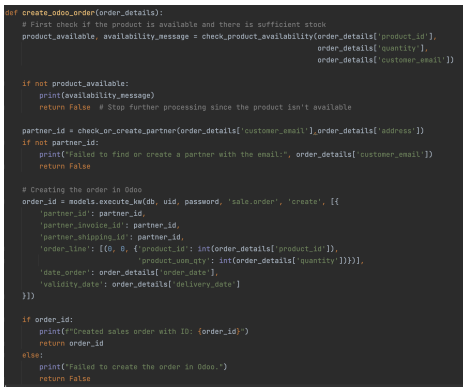
Figure 17: notify_supplier function

send_email function does responding to customers about various issues or confirmations regarding their orders.

```
def send_email(subject, message, from_addr, to_addr):
    msg = MIMEMultipart()
    msg['From'] = from_addr
    msg['To'] = to_addr
    msg['Subject'] = subject
    msg.attach(MIMEText(message, 'plain'))
    server = smtplib.SMTP('localhost', 25)
    server.send_message(msg)
    server.quit()
```

Figure 18: send_email function

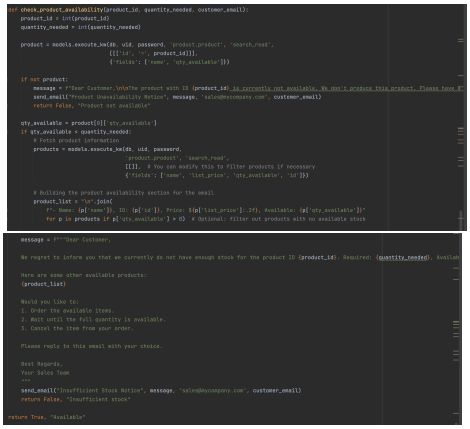
create_odoo_order function checks product availability and partner details (customer), and creates a sales order in Odoo if all conditions are satisfied.



```
def create_odoo_order(order_details):
    # First check if the product is available and there is sufficient stock
    product_available, availability_message = check_product_availability(order_details['product_id'],
                                                                         order_details['quantity'],
                                                                         order_details['customer_email'])

    if not product_available:
        print(availability_message)
        return False  # Stop further processing since the product isn't available

    partner_id = check_or_create_partner(order_details['customer_email'], order_details['address'])
    if not partner_id:
        print("Failed to find or create a partner with the email:", order_details['customer_email'])
        return False

    # Creating the order in Odoo
    order_id = models.execute_kw(db, uid, password, 'sale.order', 'create', [{
        'partner_id': partner_id,
        'partner_invoice_id': partner_id,
        'partner_shipping_id': partner_id,
        'order_line': [(0, 0, {'product_id': int(order_details['product_id']),
                               'product_uom_qty': int(order_details['quantity'])})],
        'date_order': order_details['order_date'],
        'validity_date': order_details['delivery_date']
    }])

    if order_id:
        print(f"Created sales order with ID: {order_id}")
        return order_id
    else:
        print("Failed to create the order in Odoo.")
        return False
```

Figure 19: create_odoo_order function

check_product_availability function checks if a product is available and in sufficient quantity in Odoo. If not, it sends an email to customer with send_email function suggesting ordering alternative products or wait for product replenishment or cancelling order.



```
def check_product_availability(product_id, quantity_needed, customer_email):
    product_id = int(product_id)
    quantity_needed = int(quantity_needed)

    product = models.execute_kw(db, uid, password, 'product.product', 'search_read',
                                [[['id', '=', product_id]]],
                                {'fields': ['name', 'qty_available']})

    if not product:
        message = f"Dear Customer,\n\nThe product with ID {product_id} is currently not available. We don't produce this product. Please have a
        send_email("Product Unavailability Notice", message, 'sales@mycompany.com', customer_email)
        return False, "Product not available"

    qty_available = product[0]['qty_available']
    if qty_available < quantity_needed:
        # Fetch product information
        products = models.execute_kw(db, uid, password,
                                     'product.product', 'search_read',
                                     [[]],  # You can modify this to filter products if necessary
                                     {'fields': ['name', 'list_price', 'qty_available', 'id']})

        # Building the product availability section for the email
        product_list = "\n".join(
            f"- Name: {p['name']}, ID: {p['id']}, Price: ${p['list_price']:.2f}, Available: {p['qty_available']}"
            for p in products if p['qty_available'] > 0  # Optional: filter out products with no available stock
        )
```

```
        message = f"""Dear Customer,

        We regret to inform you that we currently do not have enough stock for the product ID {product_id}. Required: {quantity_needed}, Availab

        Here are some other available products:
        {product_list}

        Would you like to:
        1. Order the available items.
        2. Wait until the full quantity is available.
        3. Cancel the item from your order.

        Please reply to this email with your choice.

        Best Regards,
        Your Sales Team
        """
        send_email('Insufficient Stock Notice', message, 'sales@mycompany.com', customer_email)
        return False, "Insufficient stock"

    return True, "Available"
```

Figure 20: check_product_availability function

If the product is available, check_or_create_partner is called. It ensures that the customer details are registered in Odoo as a partner (client) record, creating a new one if necessary.

```
def check_or_create_partner(email, address=None):
    try:
        if not email:
            raise ValueError("Email address is required for creating or finding a partner.")

        email_str = str(email)
        partner_search = models.execute_kw(db, uid, password, 'res.partner', 'search', [[['email', '=', email_str]]])
        if partner_search:
            partner_id = partner_search[0]
            # Update address if provided and not None
            if address:
                models.execute_kw(db, uid, password, 'res.partner', 'write', [partner_id, {'street': address}])
            return partner_id
        else:
            partner_data = {
                'name': email_str.split('@')[0],
                'email': email_str,
                'is_company': False,
                'customer_rank': 1
            }
            # Ensure address is not None when creating a new partner
            if address:
                partner_data['street'] = address
            return models.execute_kw(db, uid, password, 'res.partner', 'create', [partner_data])
    except Exception as e:
        print(f"Error in partner creation or retrieval: {e}")
        return None
```

Figure 21: check_or_create_partner function

If the quantity is available for the product, create_odoo_order integrates with Odoo to create an order entry based on the extracted and validated order details and creates/updates a customer record.

To see the **results**, we run parse.py file. The automatic response mails are sent from the company.

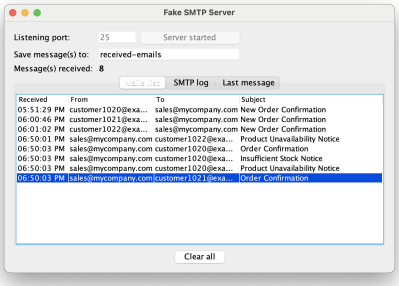



Figure 22: mail folder

When the order is successful with the product id 21, company's message:

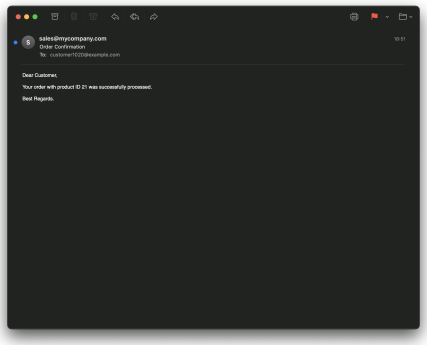



Figure 23: order successful message

When the product id 1 is not produced by the company, company's message:

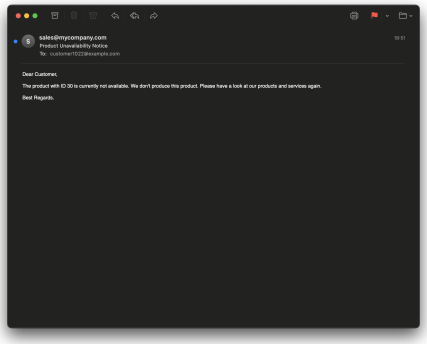



Figure 24: product unavailability message

When the product has insufficient stocks, company's message:

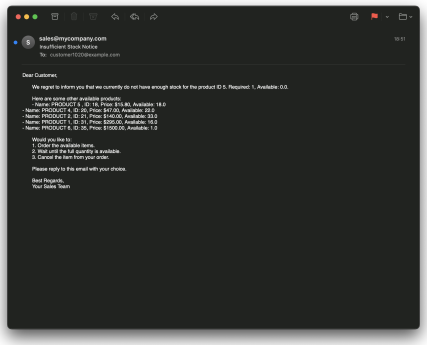



Figure 25: insufficient stock message

If the customer responds to the insufficient stock mail above with a message like this below:

Case 1, when the customer choose to order other available products:

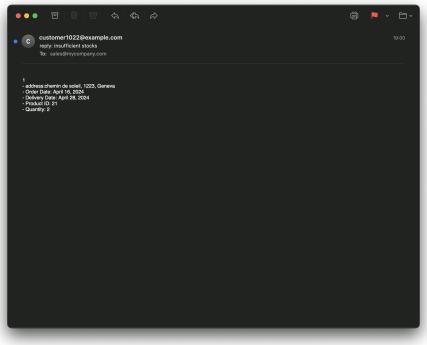



Figure 26: case 1 : reply to insufficient stock message

The company's response:

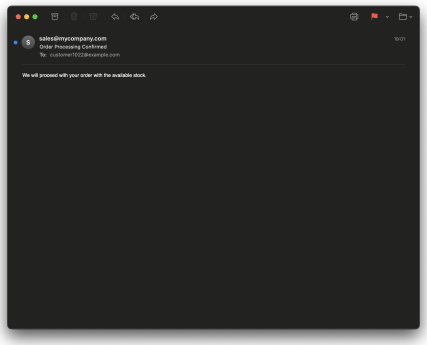



Figure 27: case 1 company response

Case 2, when the customer wants to wait for the product replenishment:

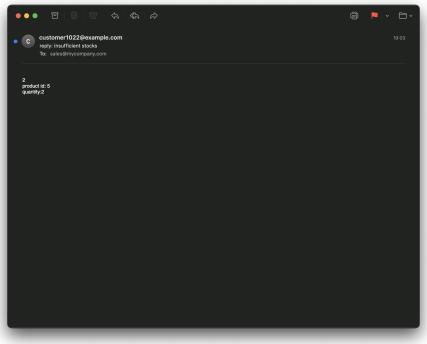



Figure 28: case 2 customer response

Automatic mail to supplier about product replenishment:

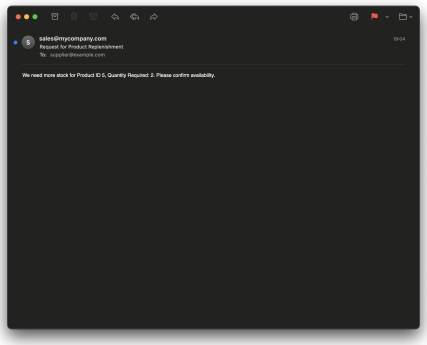



Figure 29: company's mail to supplier

Automatic mail to customer to inform about product replenishment:

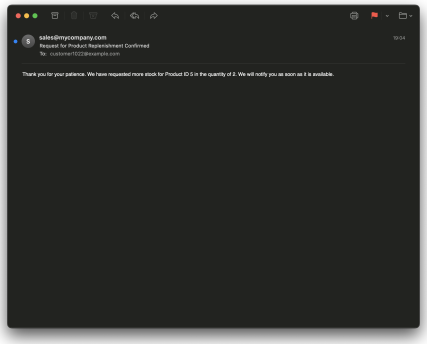



Figure 30: company's mail to customer

Case 3, when the customer chooses to cancel their order with this mail:

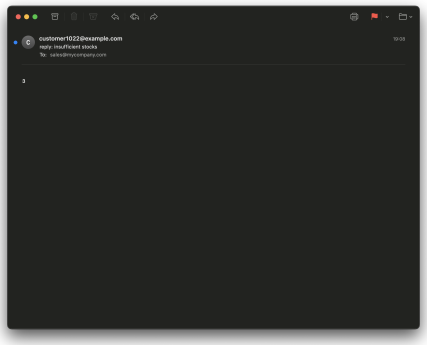



Figure 31: customer's mail to cancel order

Company's automatic response to customer:

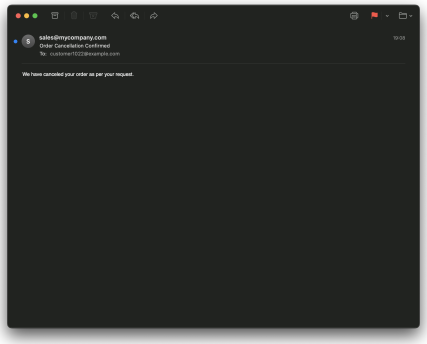



Figure 32: company's mail to customer

### 3.4 Deployment

To deploy the developed solution effectively, IT departments should follow a systematic approach to ensure all necessary components are correctly installed and configured.

Software Installation:

Python: Ensure Python is installed on the server, along with pip for managing packages.

Odoo: The solution integrates with Odoo, so ensure that Odoo is installed and configured correctly on server.

Source Code: Use Git to manage the source code. Ensure the latest version of the project is pulled from the repository.

Configure SMTP settings for sending emails, including server address, port, and authentication details.

Set up environmental variables or configuration files for database connections and other sensitive information to avoid hard-coding credentials.

Conduct thorough testing to ensure that the integration with Odoo and the email functionality works as expected.

## 4 Conclusion

This automation enhances efficiency by reducing manual data entry and speeding up order processing.

Workability and Challenges:

Workability: The solution is highly workable within environments where email is a standard mode of receiving orders.

Challenges: The current system's reliance on identifying specific keywords such as "order date" limits its ability to fully comprehend and extract information from the entire text in a nuanced manner. This method, while functional for structured data, struggles with the complexities and variations found in real-world email communications. Moving forward, enhancing the system's capabilities with more advanced machine learning models could address these limitations, enabling a more intelligent and adaptive approach to information extraction. This would not only improve the system's accuracy but also its efficiency in handling a broader spectrum of email formats, thus enhancing overall user experience and system reliability.

Benefits:

Efficiency: Automates the order entry process, reducing time and errors associated with manual input.

Scalability: Easily handles increased volume of orders without additional human resources.

Project Retrospect:

Why This Project?: I chose this project subject, because automating order processing through email integration was driven by the need to streamline operations, a common requirement in many businesses.

Successes: Successful integration with Odoo and the ability to automate order parsing and creation and to automate email communication.

Areas for Improvement: Handling varied and complex email formats and improving real-time inventory updates could enhance the system further.

# 5 Annexes

git link: `https://gitlab.unige.ch/Zeynep.Caysar/sie_integration_project`