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

Zeynep Caysar

20 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, inventory and customer service 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 a basic CRM for 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

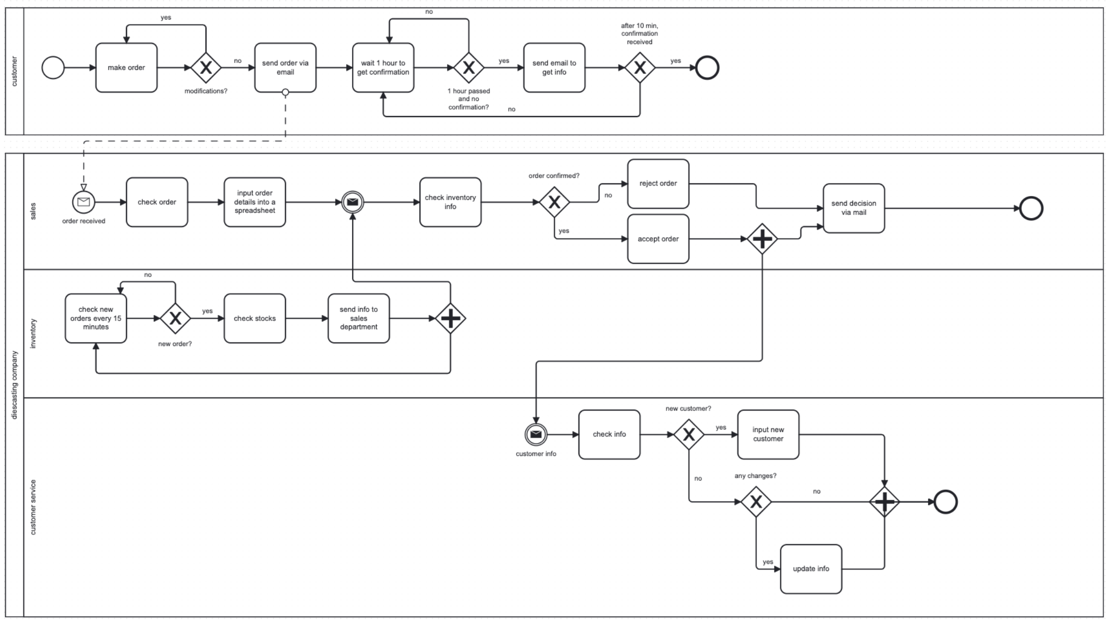


Figure 2: AS-IS BPMN

Flow of Customer Lane:
First, they make an order, if there are any modifications, they make; if there aren't, they send their orders 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. If 1 hour is passed and they got no confirmation, they resend email to get information about their order. After 10 minutes, if they got the confirmation, then the process finishes; if not, they wait again for another 1 hour and the loop continues until they get a confirmation about their order.

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. This process is prone to human error and is time-consuming. They wait for the inventory department's message. After they check the inventory message. If the order is confirmed, they accept the order and send their decision to customer by email. If not, they reject the order and again they send their decision via email.

The Inventory team manually checks if there are any new orders every 15 minutes. If there is a new order, they check the stocks and send the information 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.

Customer representatives manually check a basic CRM system to identify if the customer has an existing relationship with the company. This step is crucial for personalized service and maintaining customer relationships but is currently inefficient.

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, and a basic CRM, 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, and utilizes Odoo's API to interact with its CRM, 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

- Check every 1 minute: The email folder is checked every minute to see if there any mail.
- Email Capture: Orders are received via email, captured by FakeSMTP during testing phases to simulate real-world email receipt.
- Email Parsing: A Python script reads the email, extracting key details like order date, delivery date, customer information, and product details.
- Inventory Check: The script verifies product availability in the Inventory module. If stock levels are insufficient, it notifies the customer with an automatic refusal email.
- Order Processing: If inventory is sufficient, the script creates a sales order in the Odoo Sales module, linking all necessary details and ensuring that the inventory is updated.
- Customer Verification/Create: The script checks if the customer exists in the Odoo CRM module via API; if not, it creates a new customer record.
- Confirmation: Upon successful order creation, an automatic confirmation email is sent to the customer.

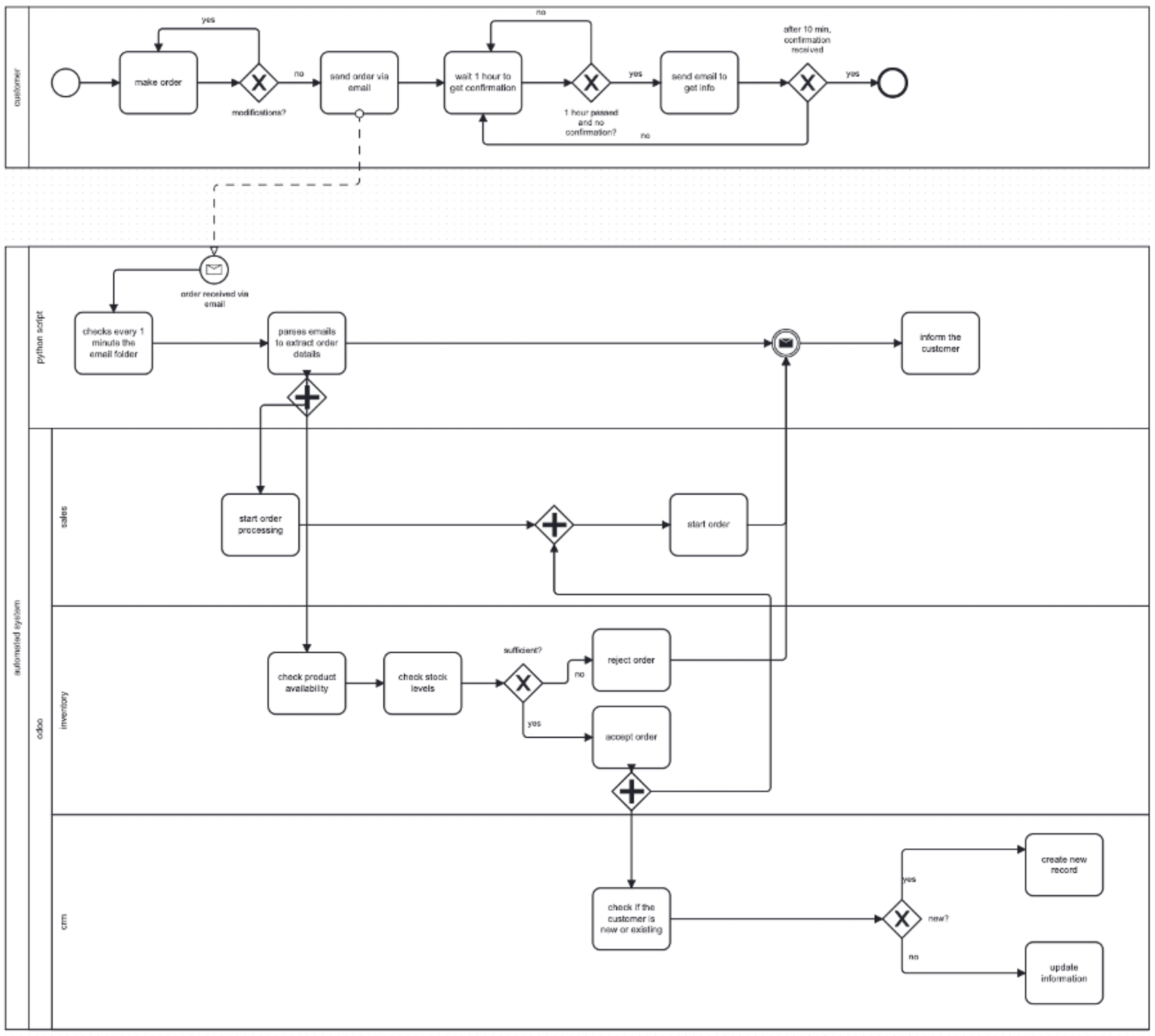


Figure 3: TO-BE BPMN

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.
Odoo ERP: Central management software that handles CRM, sales, and inventory.
SpaCy: Natural Language Processing library to parse email content effectively.
Odoo API: RESTful API for accessing and manipulating Odoo's database, particularly for creating and updating CRM records, checking inventory, and processing sales orders.
SMTP Protocol: Used for sending confirmation emails through the Python smtplib.

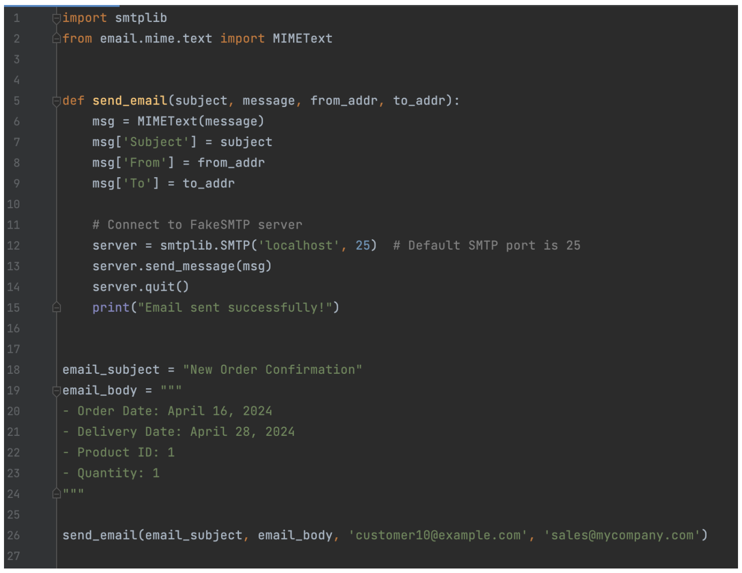
3.2.1 Code review

The Python script includes functions for parsing emails, checking product availability, interacting with Odoo's API to create or update records, and sending emails.

Key functions are:

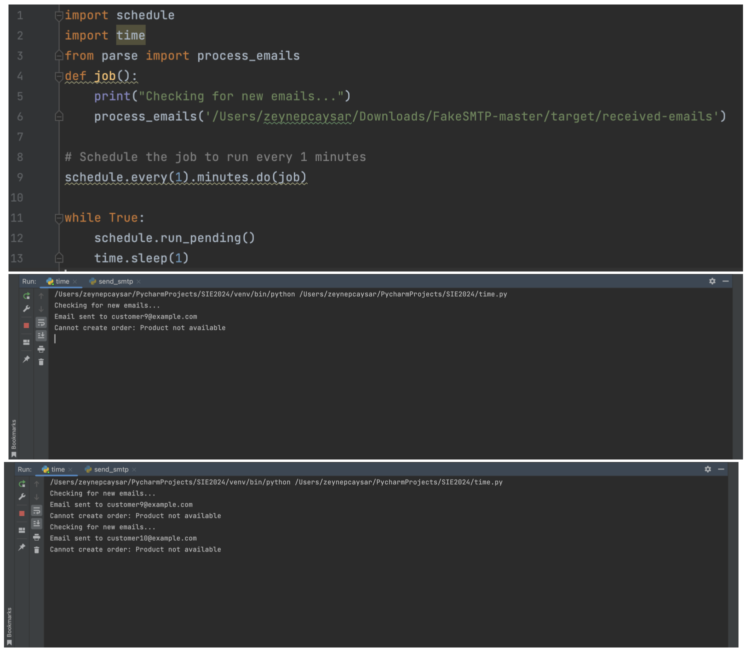
def send_email(): This function sends emails using Python's smtplib. It constructs an email with a subject, message body, sender, and recipient details, then connects to a local SMTP server to send the email.

def job(): This Python script is set up to periodically check for new emails every minute using the schedule library. The job function, when called, prints a message indicating it's checking for new emails and then runs the process_emails function with a specified directory path. This function looks for the latest email file in the specified directory and processes it to extract and handle order details. The infinite loop at the end of the script ensures that this scheduling continues to check and process emails as long as the script is running. This is useful for automating tasks like email processing without manual intervention, keeping the system responsive to new information continuously.



```python
import smtplib
from email.mime.text import MIMEText


def send_email(subject, message, from_addr, to_addr):
    msg = MIMEText(message)
    msg['Subject'] = subject
    msg['From'] = from_addr
    msg['To'] = to_addr

    # Connect to FakeSMTP server
    server = smtplib.SMTP('localhost', 25)  # Default SMTP port is 25
    server.send_message(msg)
    server.quit()
    print("Email sent successfully!")


email_subject = "New Order Confirmation"
email_body = """
- Order Date: April 16, 2024
- Delivery Date: April 28, 2024
- Product ID: 1
- Quantity: 1
"""

send_email(email_subject, email_body, 'customer10@example.com', 'sales@mycompany.com')
```

Figure 4: send_email() function



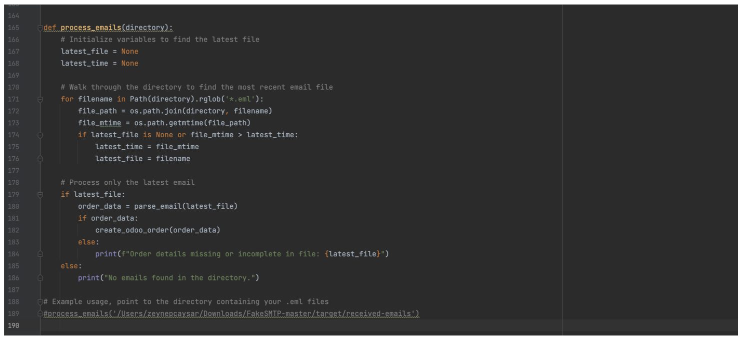
```python
import schedule
import time
from parse import process_emails
def job():
    print("Checking for new emails...")
    process_emails('/Users/zeynepcaysar/Downloads/FakeSMTP-master/target/received-emails')

# Schedule the job to run every 1 minutes
schedule.every(1).minutes.do(job)

while True:
    schedule.run_pending()
    time.sleep(1)
```

```
/Users/zeynepcaysar/PycharmProjects/SIE2024/venv/bin/python /Users/zeynepcaysar/PycharmProjects/SIE2024/time.py
Checking for new emails...
Email sent to customer9@example.com
Cannot create order: Product not available
```

```
/Users/zeynepcaysar/PycharmProjects/SIE2024/venv/bin/python /Users/zeynepcaysar/PycharmProjects/SIE2024/time.py
Checking for new emails...
Email sent to customer9@example.com
Cannot create order: Product not available
Checking for new emails...
Email sent to customer10@example.com
Cannot create order: Product not available
```

Figure 5: Every 1 minute, the scripts looks for a new email.

def process_emails(): The process_emails function scans a specified directory for the most recent email file (with a .eml extension), determines which file is the latest by comparing their modification times, and then processes only this latest email. If it finds a file, it attempts to extract order details from it using the parse_email function. If order details are successfully extracted, it proceeds to create an order in an Odoo system using the create_odoo_order function. If no valid order details are found, it prints a message indicating that details are missing or incomplete. If no email files are found in the directory at all, it informs the user accordingly.

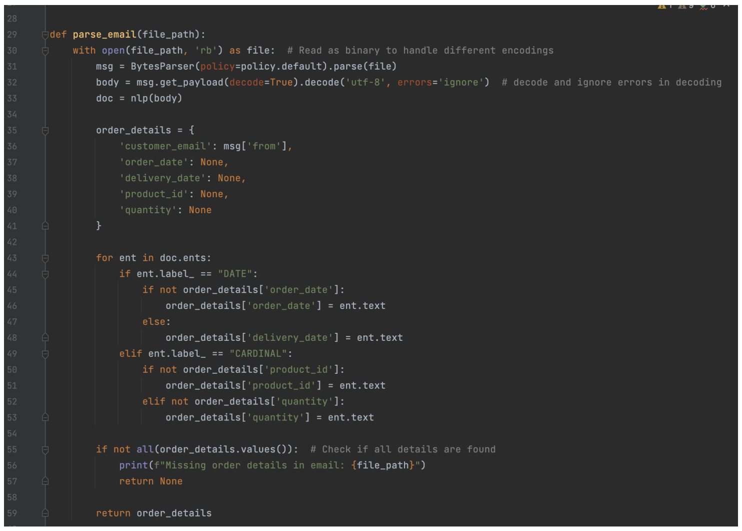


```python

def process_emails(directory):
    # Initialize variables to find the latest file
    latest_file = None
    latest_time = None

    # Walk through the directory to find the most recent email file
    for filename in Path(directory).rglob('*.eml'):
        file_path = os.path.join(directory, filename)
        file_mtime = os.path.getmtime(file_path)
        if latest_file is None or file_mtime > latest_time:
            latest_time = file_mtime
            latest_file = filename

    # Process only the latest email
    if latest_file:
        order_data = parse_email(latest_file)
        if order_data:
            create_odoo_order(order_data)
        else:
            print(f"Order details missing or incomplete in file: {latest_file}")
    else:
        print("No emails found in the directory.")

# Example usage, point to the directory containing your .eml files
#process_emails('/Users/zeynepcaysar/Downloads/FakeSMTP-master/target/received-emails')

```

Figure 6: process_email() function

def parse_email(): The parse_email function reads an email file in binary format to handle different encodings properly. It decodes the email's content to extract text, ignoring any errors in decoding, and utilizes a natural language processing model (nlp) to analyze this text. It aims to gather specific details about an order from the email: the customer's email address, order and delivery dates, product ID, and quantity.

It uses named entity recognition (NER) to identify dates and cardinal numbers within the text. Dates are used to set the order and delivery dates. Cardinal numbers are first interpreted as the product ID, and if a product ID has already been set, the next cardinal number is taken as the quantity. If any of these details are missing after parsing the email, it reports that some order details are missing and returns None; otherwise, it returns a dictionary of the order details. This function is typically used to automate the extraction of order information from incoming emails, facilitating further processing such as order creation in a system like Odoo.



```python

def parse_email(file_path):
    with open(file_path, 'rb') as file:  # Read as binary to handle different encodings
        msg = BytesParser(policy=policy.default).parse(file)
        body = msg.get_payload(decode=True).decode('utf-8', errors='ignore')  # decode and ignore errors in decoding
        doc = nlp(body)

        order_details = {
            'customer_email': msg['from'],
            'order_date': None,
            'delivery_date': None,
            'product_id': None,
            'quantity': None
        }

        for ent in doc.ents:
            if ent.label_ == "DATE":
                if not order_details['order_date']:
                    order_details['order_date'] = ent.text
                else:
                    order_details['delivery_date'] = ent.text
            elif ent.label_ == "CARDINAL":
                if not order_details['product_id']:
                    order_details['product_id'] = ent.text
                elif not order_details['quantity']:
                    order_details['quantity'] = ent.text

        if not all(order_details.values()):  # Check if all details are found
            print(f"Missing order details in email: {file_path}")
            return None

        return order_details
```

Figure 7: parse_email() function

create_odoo_order(): This function, create_odoo_order create a sales order in an Odoo system using provided order details. It performs several key operations:

Check for Null Input: It immediately returns if no order details are provided, indicating that there's nothing to process.

Check Product Availability: It verifies if the product is available in sufficient quantity. If not, it prints an error message and returns without creating an order.

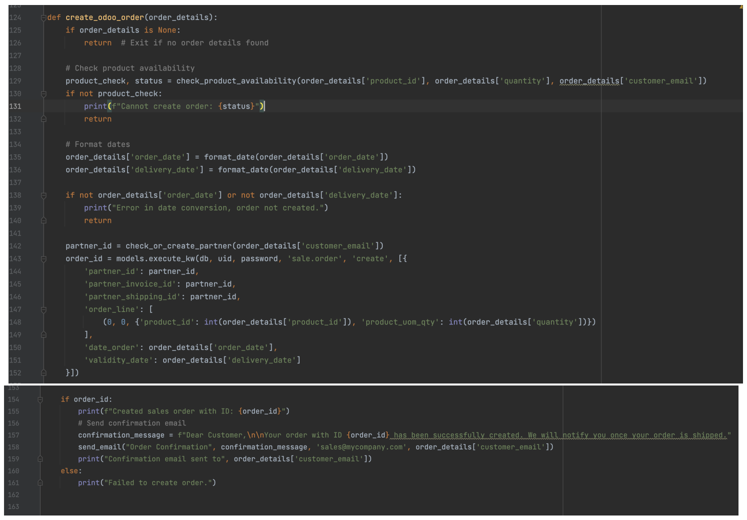
Format Dates: It formats the order and delivery dates. If there's an error in date conversion, it prints an error message and aborts the operation.

Partner Verification: It checks if the customer exists in the Odoo database or creates a new record if the customer doesn't exist.

Order Creation: It attempts to create a sales order with all the given details in Odoo. If successful, it prints a confirmation with the order ID.

Send Confirmation Email: If the order is successfully created, it sends a confirmation email to the customer.

This function is integral for automating order processing from emails, ensuring that all orders are logged into the system and that the inventory is updated appropriately. It also enhances customer communication by confirming orders through email.



```python
def create_odoo_order(order_details):
    if order_details is None:
        return  # Exit if no order details found

    # Check product availability
    product_check, status = check_product_availability(order_details['product_id'], order_details['quantity'], order_details['customer_email'])
    if not product_check:
        print(f"Cannot create order: {status}")
        return

    # Format dates
    order_details['order_date'] = format_date(order_details['order_date'])
    order_details['delivery_date'] = format_date(order_details['delivery_date'])

    if not order_details['order_date'] or not order_details['delivery_date']:
        print("Error in date conversion, order not created.")
        return

    partner_id = check_or_create_partner(order_details['customer_email'])
    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}")
        # Send confirmation email
        confirmation_message = f"Dear Customer,\n\nYour order with ID {order_id} has been successfully created. We will notify you once your order is shipped."
        send_email("Order Confirmation", confirmation_message, 'sales@mycompany.com', order_details['customer_email'])
        print("Confirmation email sent to", order_details['customer_email'])
    else:
        print("Failed to create order.")
```

Figure 8: create_odoo_order() function

check_or_create_partner(): The function check_or_create_partner manages customer data within an Odoo ERP system. Here's what the function does:

Convert to String: It ensures the email variable is a string, accommodating for different data types that might be passed to the function.

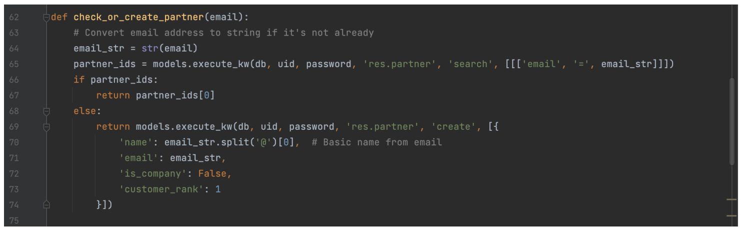
Search for Existing Partner: It checks if there's an existing partner (customer) in the Odoo database with the given email address.

Return Existing Partner ID: If a partner with the given email exists, it returns the ID of the first partner found, ensuring that duplicate records are not created.

Create New Partner: If no existing partner is found, it creates a new partner record in the database with the provided email and a name derived from the email prefix (before the @ symbol). The new partner is marked as an individual (not a company) with a customer rank set to 1, indicating they are a customer.

Return New Partner ID: After creating a new partner, it returns the ID of the newly created partner.

This function helps maintain an accurate and up-to-date customer database by ensuring that each customer has a unique record in the system. It automates the process of customer data entry and retrieval, which is critical for managing sales, customer relationships, and communications within Odoo.



```python
def check_or_create_partner(email):
    # Convert email address to string if it's not already
    email_str = str(email)
    partner_ids = models.execute_kw(db, uid, password, 'res.partner', 'search', [[['email', '=', email_str]]])
    if partner_ids:
        return partner_ids[0]
    else:
        return models.execute_kw(db, uid, password, 'res.partner', 'create', [{
            'name': email_str.split('@')[0],  # Basic name from email
            'email': email_str,
            'is_company': False,
            'customer_rank': 1
        }])

```

Figure 9: check_or_create_partner() function

check_product_availability(): The function check_product_availability() manages inventory within an Odoo ERP system, specifically to check if a product is available in sufficient quantity for customer orders. Here's a breakdown of its operation:

Convert Data Types: It converts the product_id and quantity_needed to integers to ensure compatibility with database operations.

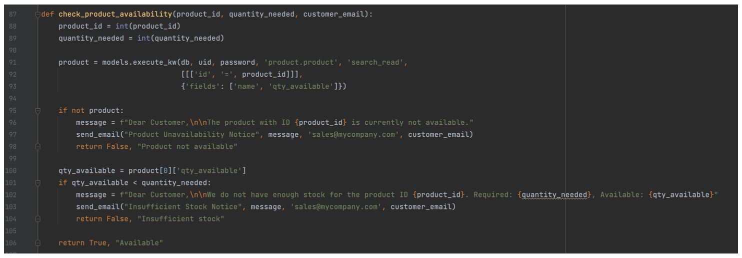
Retrieve Product Data: It uses the Odoo XML-RPC API to query the product.product model for the specified product ID, retrieving data about the product's name and available quantity.

Check Product Existence: If the product is not found in the database (i.e., the query returns an empty list), it sends an email to the customer informing them of the product's unavailability and returns False with a message indicating that the product is not available.

Check Stock Levels: If the product exists, it compares the required quantity (quantity_needed) to the quantity available (qty_available). If the available quantity is less than the required quantity, it sends an email warning of insufficient stock and also returns False with a message about insufficient stock.

Return Availability: If the product exists and there is sufficient stock, it returns True with a message stating "Available", indicating that the order can proceed.

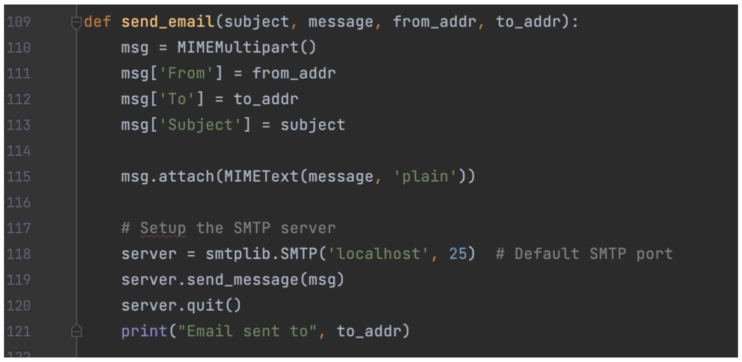
This function effectively automates inventory checks for customer orders, ensuring that orders are only processed if sufficient stock is available, thus preventing overselling and managing customer expectations effectively.



```python
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."
        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:
        message = f"Dear Customer,\n\nWe do not have enough stock for the product ID {product_id}. Required: {quantity_needed}, Available: {qty_available}"
        send_email("Insufficient Stock Notice", message, 'sales@mycompany.com', customer_email)
        return False, "Insufficient stock"

    return True, "Available"
```

Figure 10: check_product_availability() function

send_email(): This function sends confirmation or refusal email to the customer. The send_email function constructs and sends an email using SMTP. It sets up the email's headers and body, connects to an SMTP server on the local host, sends the email, and then closes the connection, confirming the action with a printed message.

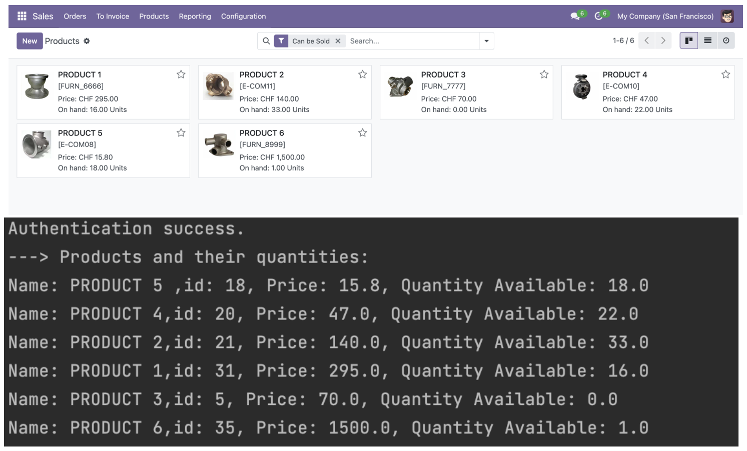


```python
def send_email(subject, message, from_addr, to_addr):
    msg = MIMEMultipart()
    msg['From'] = from_addr
    msg['To'] = to_addr
    msg['Subject'] = subject

    msg.attach(MIMEText(message, 'plain'))

    # Setup the SMTP server
    server = smtplib.SMTP('localhost', 25)  # Default SMTP port
    server.send_message(msg)
    server.quit()
    print("Email sent to", to_addr)
```

Figure 11: send_email() function

3.2.2 Results

These are the available products.



PRODUCT 1
[FURN_6666]
Price: CHF 295.00
On hand: 16.00 Units

PRODUCT 2
[E-COM11]
Price: CHF 140.00
On hand: 33.00 Units

PRODUCT 3
[FURN_7777]
Price: CHF 70.00
On hand: 0.00 Units

PRODUCT 4
[E-COM10]
Price: CHF 47.00
On hand: 22.00 Units

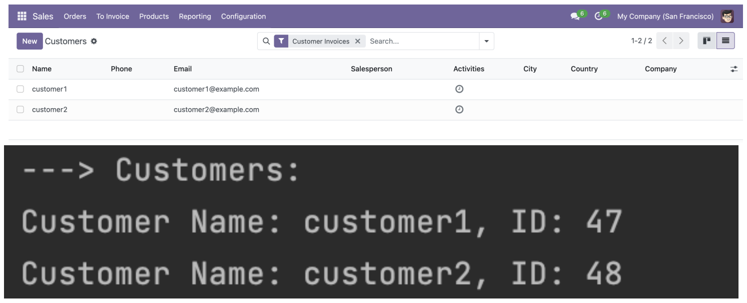
PRODUCT 5
[E-COM08]
Price: CHF 15.80
On hand: 18.00 Units

PRODUCT 6
[FURN_8999]
Price: CHF 1,500.00
On hand: 1.00 Units

```
Authentication success.
---> Products and their quantities:
Name: PRODUCT 5 ,id: 18, Price: 15.8, Quantity Available: 18.0
Name: PRODUCT 4,id: 20, Price: 47.0, Quantity Available: 22.0
Name: PRODUCT 2,id: 21, Price: 140.0, Quantity Available: 33.0
Name: PRODUCT 1,id: 31, Price: 295.0, Quantity Available: 16.0
Name: PRODUCT 3,id: 5, Price: 70.0, Quantity Available: 0.0
Name: PRODUCT 6,id: 35, Price: 1500.0, Quantity Available: 1.0
```

Figure 12: Products in odoo database

These are the customers.

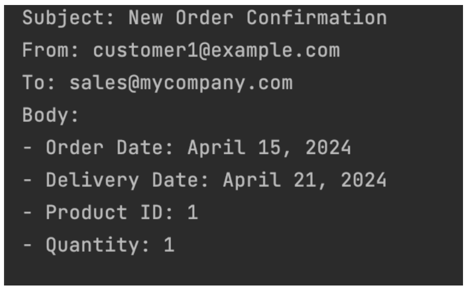


Name	Phone	Email	Salesperson	Activities	City	Country	Company
customer1		customer1@example.com					
customer2		customer2@example.com					

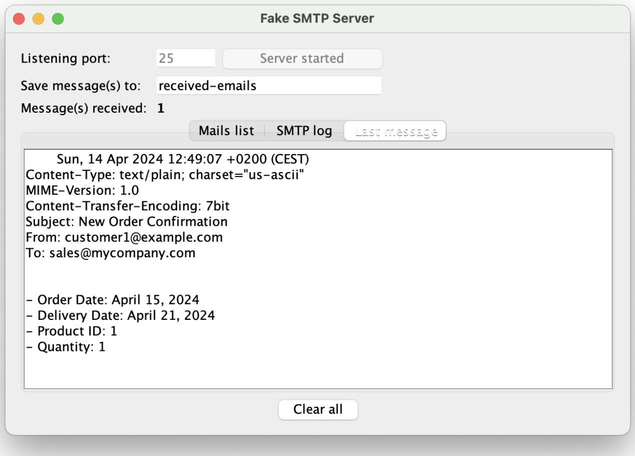
```
---> Customers:
Customer Name: customer1, ID: 47
Customer Name: customer2, ID: 48
```

Figure 13: Customers

When an existing customer makes an order:



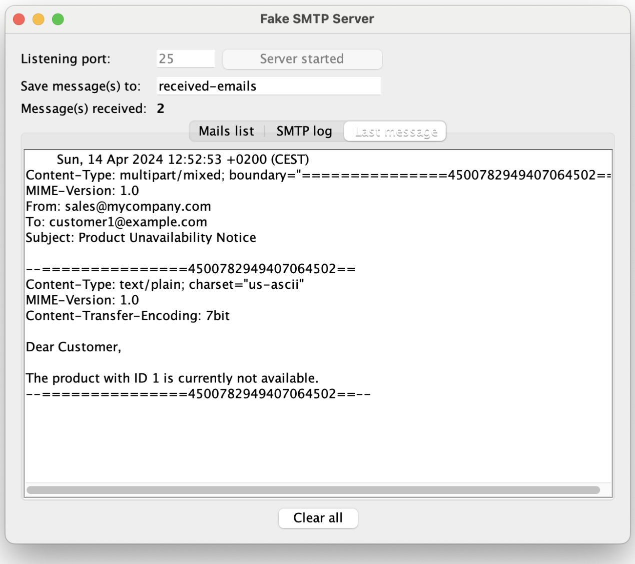
```
Subject: New Order Confirmation
From: customer1@example.com
To: sales@mycompany.com
Body:
- Order Date: April 15, 2024
- Delivery Date: April 21, 2024
- Product ID: 1
- Quantity: 1
```



Listening port: 25 Server started
Save message(s) to: received-emails
Message(s) received: 1

```
      Sun, 14 Apr 2024 12:49:07 +0200 (CEST)
Content-Type: text/plain; charset="us-ascii"
MIME-Version: 1.0
Content-Transfer-Encoding: 7bit
Subject: New Order Confirmation
From: customer1@example.com
To: sales@mycompany.com


- Order Date: April 15, 2024
- Delivery Date: April 21, 2024
- Product ID: 1
- Quantity: 1
```

Figure 14: customer's order message comes

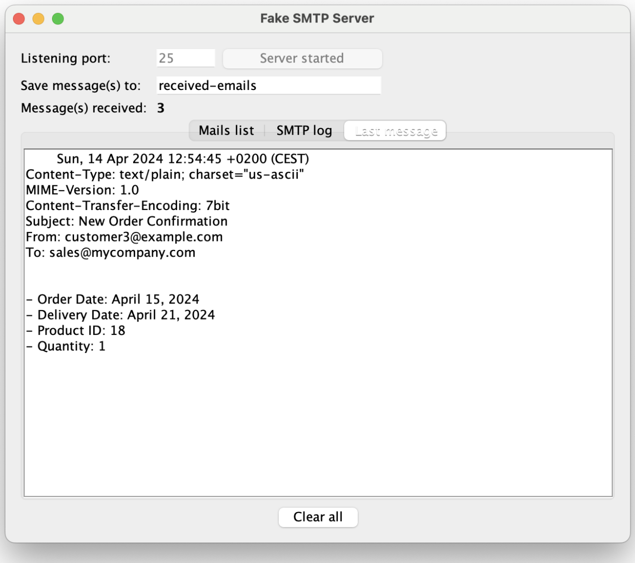


Listening port: 25 Server started
Save message(s) to: received-emails
Message(s) received: 2

```
      Sun, 14 Apr 2024 12:52:53 +0200 (CEST)
Content-Type: multipart/mixed; boundary="===============4500782949407064502==
MIME-Version: 1.0
From: sales@mycompany.com
To: customer1@example.com
Subject: Product Unavailability Notice

--===============4500782949407064502==
Content-Type: text/plain; charset="us-ascii"
MIME-Version: 1.0
Content-Transfer-Encoding: 7bit

Dear Customer,

The product with ID 1 is currently not available.
--===============4500782949407064502==--
```

Figure 15: company's automatic response to customer

When a new customer makes an order:



```
      Sun, 14 Apr 2024 12:54:45 +0200 (CEST)
Content-Type: text/plain; charset="us-ascii"
MIME-Version: 1.0
Content-Transfer-Encoding: 7bit
Subject: New Order Confirmation
From: customer3@example.com
To: sales@mycompany.com


- Order Date: April 15, 2024
- Delivery Date: April 21, 2024
- Product ID: 18
- Quantity: 1
```

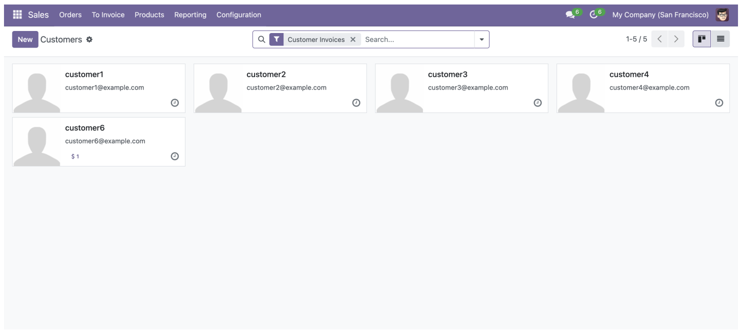
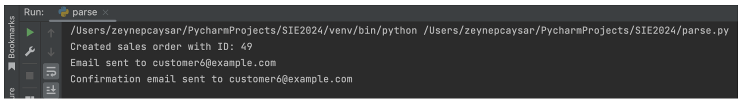


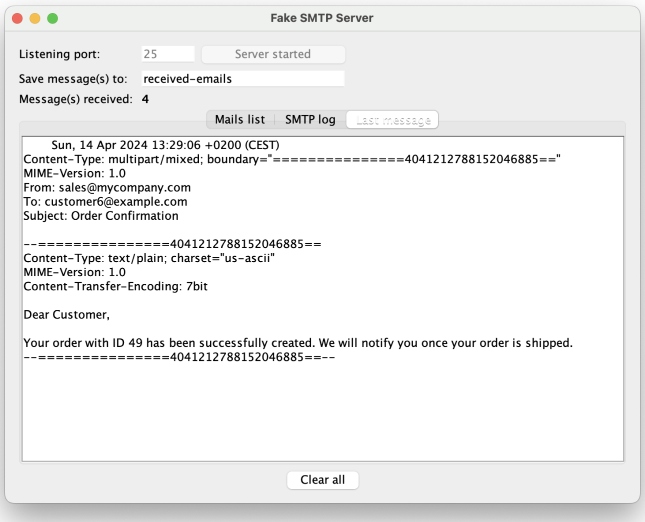
Figure 16: new customer added to the database

Order cases

CASE 1: Order success

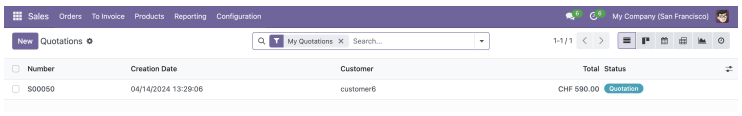


```
/Users/zeynepcaysar/PycharmProjects/SIE2024/venv/bin/python /Users/zeynepcaysar/PycharmProjects/SIE2024/parse.py
Created sales order with ID: 49
Email sent to customer6@example.com
Confirmation email sent to customer6@example.com
```



Listening port: 25 Server started
Save message(s) to: received-emails
Message(s) received: **4**

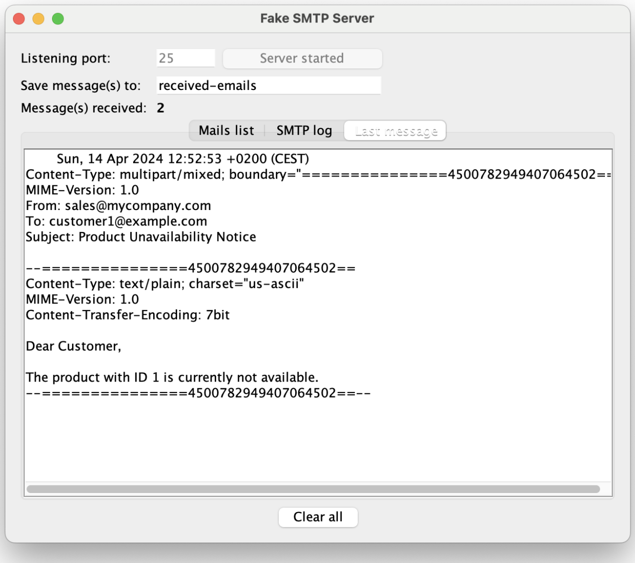
```
       Sun, 14 Apr 2024 13:29:06 +0200 (CEST)
Content-Type: multipart/mixed; boundary="===============4041212788152046885=="
MIME-Version: 1.0
From: sales@mycompany.com
To: customer6@example.com
Subject: Order Confirmation

--===============4041212788152046885==
Content-Type: text/plain; charset="us-ascii"
MIME-Version: 1.0
Content-Transfer-Encoding: 7bit

Dear Customer,

Your order with ID 49 has been successfully created. We will notify you once your order is shipped.
--===============4041212788152046885==--
```



Number	Creation Date	Customer	Total	Status
S00050	04/14/2024 13:29:06	customer6	CHF 590.00	Quotation

Figure 17: Order is confirmed

CASE 2: Product is unrecognisable

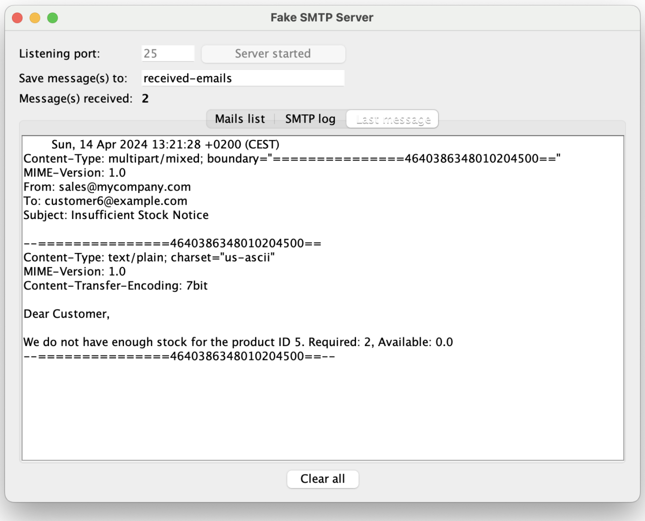


```
Listening port:          25
Save message(s) to:      received-emails
Message(s) received:     2
```

```
       Sun, 14 Apr 2024 12:52:53 +0200 (CEST)
Content-Type: multipart/mixed; boundary="===============4500782949407064502==
MIME-Version: 1.0
From: sales@mycompany.com
To: customer1@example.com
Subject: Product Unavailability Notice

--===============4500782949407064502==
Content-Type: text/plain; charset="us-ascii"
MIME-Version: 1.0
Content-Transfer-Encoding: 7bit

Dear Customer,

The product with ID 1 is currently not available.
--===============4500782949407064502==--
```

Figure 18: Product is unrecognisable

CASE 3: Stock levels are insufficient



```
Listening port:          25
Save message(s) to:      received-emails
Message(s) received:     2
```

```
      Sun, 14 Apr 2024 13:21:28 +0200 (CEST)
Content-Type: multipart/mixed; boundary="===============4640386348010204500=="
MIME-Version: 1.0
From: sales@mycompany.com
To: customer6@example.com
Subject: Insufficient Stock Notice

--===============4640386348010204500==
Content-Type: text/plain; charset="us-ascii"
MIME-Version: 1.0
Content-Transfer-Encoding: 7bit

Dear Customer,

We do not have enough stock for the product ID 5. Required: 2, Available: 0.0
--===============4640386348010204500==--
```

Figure 19: insufficient stocks message

3.3 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.

SpaCy: Install SpaCy, a powerful NLP library, and ensure the necessary language models are downloaded.

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

Source Code: Use Git to manage the source code. Ensure the latest version of the project is pulled from the repository. Use commands like git clone for initial setup and git pull to update the codebase.

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. Validate the NLP functionalities and ensure that the system accurately parses and processes the email data.

4 Conclusion

The proposed solution integrates NLP with business processes, specifically for handling orders via email and integrating with an ERP system like Odoo. 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: Ensuring accurate natural language processing (NLP) parsing across varied email formats is the main challenge. 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 NLP 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?: The choice to automate 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 were major wins.

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

Bugs: I couldn't make the parsing work with regex. That's why I used NLP but like mentioned in the "challenges", it can be developed to enable a more intelligent approach to information extraction.

5 Annexes

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