UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI ECONOMIA "GIORGIO FUÀ"

Bachelor's degree in

Digital Economics and Business (Inter-class: L-33, L-18)

DIGITAL TRANSFORMATIONS IN THE HO.RE.CA. SECTOR: DESIGN AND IMPLEMENTATION OF AN AGENT-BASED PLATFORM FOR INVENTORY AND PROCUREMENT MANAGEMENT

Thesis Supervisor:
Prof. Alex Mircoli

Bachelor's Thesis:
Davide Boria

Academic Year 2025/2026

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

The restaurant industry often operates on tight margins, with very long hours of work and lots of responsibility. The cost and the freshness of ingredients directly determine profitability, food safety and, most importantly, customer satisfaction. But there is another factor that is often underestimated. This factor is the pressure and the stress that the Ho.Re.Ca sector's personnel is commonly subjected to.

The idea to develop this experimental thesis is born from an intuition of a necessity that I understood in my last year of work as a professional cook. Restaurateurs, chefs, and waiters are often burnt out because they have too many activities to accomplish, working in an almost always chaotic rhythm. Inventory management is one of the activities that consume a considerable amount of time. Chefs must always be aware of what is in stock, what is expiring, and what must be reordered. Yet, in many cases this process is still handled manually. Spreadsheets, phone calls, and the possibly fleeting memory of the staff make this system slow, error-prone and difficult to scale.

In recent years, business process automation has become increasingly accessible, expanding from large enterprises with dedicated IT departments, to small and medium businesses, helping these niche realities to operate more efficiently and get larger.

Procurement processes are the functions that have gathered the most benefits from this shift, as they are based on repetitive and rule-based tasks such as monitoring stock levels, placing orders and replenishing the inventory. All functions that are well suited to automation.

Automating these processes in a restaurant offers a wide range of potential advantages. Starting from faster reaction to shortages, reduction in human error, reduced staff stress, and support for just-in-time replenishment, where stock is ordered only when needed. This aligns with the lean management principles, an approach to management which seeks to minimize waste.

Traceability is another very important advantage that automation could bring. Every product that enters the kitchen is fully traced and catalogued, reducing

the room for error.The objective of this thesis is to design and implement an automated inventory management system for the Ho.Re.Ca sector using n8n, an open-source and low-code automation workflow platform, integrated with a relational database. The system aims to automate two core processes: the monitoring and consumption of stock, and the handling of supplier replies with consequent inventory replenishment.

The first workflow monitors stock by deducting, through a FEFO (first expired, first out) logic, the ingredients consumed by dishes sold, and automatically drafts orders to suppliers for products that fall below their minimum threshold. The second workflow closes the loop by interpreting suppliers' free-text email replies through an AI agent, updating the inventory accordingly and managing deliveries through automated fallbacks and, where appropriate, human approval.

The rest of this thesis is organized as follows. Chapter two, which is divided into three sections, describes the system's design. The first part covers the database structure, and the last two sections treat the automation workflows in detail. Finally, chapter three concludes with a discussion of limitations and possible future developments.

2. DATABASE AND AUTOMATION WORKFLOWS

2.1 – DATABASE DESIGN

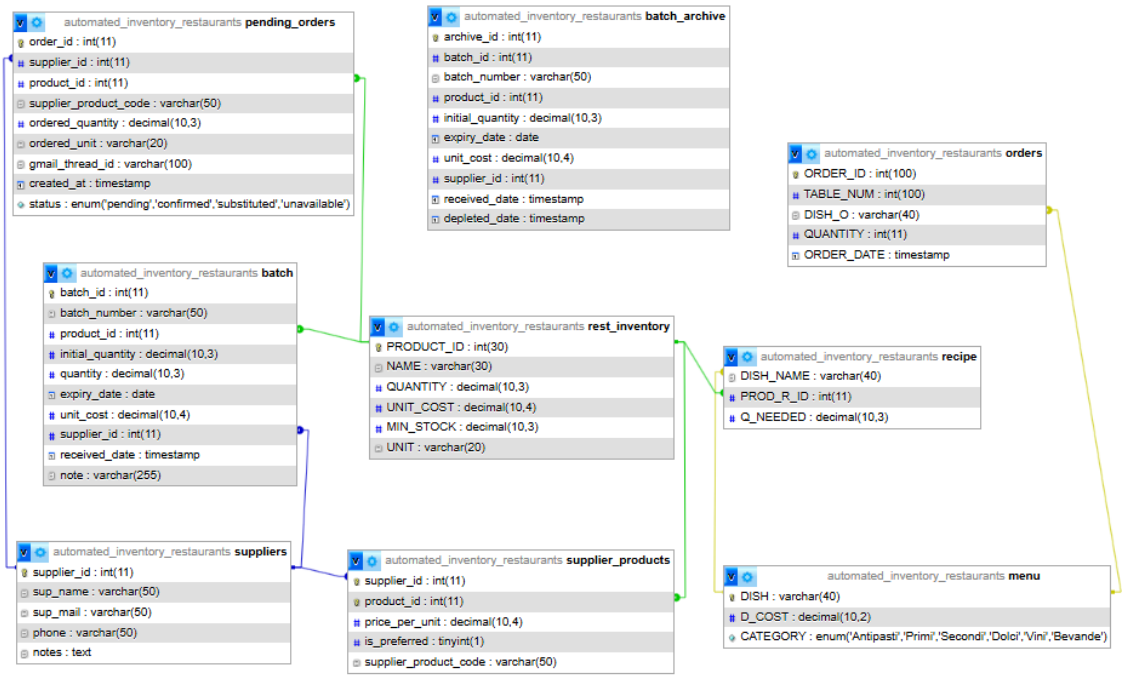


Figure 2.1-1: Database designer.

The system is built upon a relational database, implemented as the foundational component. phpMyAdmin was used, hosted on Hostinger, an online server that also hosts n8n, so that the two systems can communicate. The database keeps track of several attributes and of the events produced by the n8n automations. The schema comprises nine different entities. The first one is the table regarding the inventory of the restaurant. This table maintains a record of the product id, an autoincrement primary key, which increments by one for each new row inserted. Every primary key apart from the recipe table is built like this. Still concerning the restaurant inventory table, it includes the description of the product's name and the quantity available, which is the sum of the quantities of the active batches remaining of that specific product. The unit cost represents the active batches weighted average cost of the products, so that it is possible

to calculate weighted food costs on each dish served. Both the former columns are denormalized, maintained coherently via a resync query executed both after stock consumption and after every batch intake. The column for minimum stock acts as a threshold from where the automatic order draft will be created. Lastly, it contains the column regarding the unit with which the products are stocked.
The second table regards the batches. This table is strictly linked with the inventory, since here can be found the different stacks of the products that are located in the inventory, due to the foreign key applied. The batch table helps to monitor the FEFO (first expiry, first out) inventory mechanism. The contents of this table are populated by the second automation, based on the data extracted by its AI agent. The dimensions described by this table are batch number, expiry date, unit cost, date of receival, supplier and possible notes. Here two different quantities can be found. The INITIAL QUANTITY, that is the quantity with which the batch enters our inventory. The second is the current quantity in stock, from which the first workflow subtracts the quantity used in the dishes sold. The batch archive retains batches once they are depleted, preserving their history for traceability purposes, helping to conform to HACCP rules. The order table keeps track of the dishes ordered from clients. Each order has its ID, the number of the restaurant's table, the dishes ordered, the quantity and lastly, the order date and time.
The recipe table has a foreign key to the dishes ordered, so that the quantity of each ingredient put in dishes offered can be known. Furthermore, the menu table is observed. It delineates each dish with its cost and the category to which it pertains. Categories are entrées, pasta dishes, second courses, desserts, wines and drinks.
The last part of the database description is dedicated to the suppliers. The suppliers' table records the name, email, phone number and any notes, which may concern shipping, possible discounts or other details.
Products and their respective suppliers are registered in a separate table, called supplier products. Here the table also keeps track of the price per unit, the supplier product code and a Boolean variable assessing whether the supplier is the preferred one with respect to that product. This attribute is used later in the

automations. Lastly, a pending order table can be observed. This table serves to create and update the status of supplier orders. It records its own primary key ID, then the supplier's ID, the ID of the ordered product and its code, the quantity ordered and the email thread id. Most importantly, it keeps track of the order's condition. Pending, for newborn orders that are yet to be confirmed by suppliers; Confirmed, for confirmed orders. This means the inventory has been updated; Substituted for orders which products have been substituted for a similar one; Unavailable, for orders that are denied by supplier.

2.2 - AUTOMATION 1 - INVENTORY CONSUMPTION AND ORDER MAKING

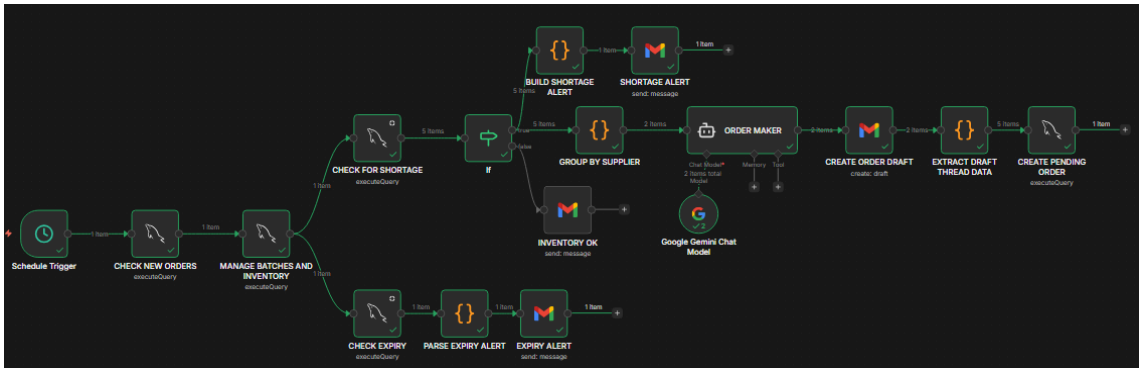


Figure 2.2-1: Automation 1 overview

With the database in place, the first automation can be described. As the subtitle anticipates, the purpose of this automation is to subtract from the batches in inventory the ingredients used in the dishes sold, checking for possible shortages or expiry and only then creating a draft for orders to suppliers.

Going into details, the automation is initiated from a scheduled trigger every 8 hours. It then checks for new orders via a MySQL node. The output of this node goes into "MANAGE BATCHES AND INVENTORY". The new node implements FEFO (First-Expired-First-Out) consumption: when dishes are sold, the ingredients required must be deducted from stock by selecting first the batches closest to their expiry date. This minimizes waste in a perishable goods environment. The entire operation is wrapped in a single SQL transaction to guarantee atomicity, so that either the whole consumption succeeds, or nothing is written.

The transaction begins with creating 3 temporary tables. The first one calculates how much of each product must be consumed, taking as input the result of the previous node, to finally calculate the total quantity to deduct.
The second temporary table lists all active batches and for each one computes the quantity contained in the batches of the same product that expires earlier.

```
COALESCE(SUM(b.quantity) OVER (
    PARTITION BY b.product_id
    ORDER BY b.expiry_date ASC, b.batch_id ASC
    ROWS BETWEEN UNBOUNDED PRECEDING AND 1 PRECEDING
  ), 0) AS qty_before
```

This is the window function which represents the core of FEFO.
The *PARTITION BY PRODUCT ID* groups batches by product, *ORDER BY EXPIRY DATE AND BATCH ID BOTH ASCENDANT* arranges them from soonest to latest and the frame *UNBOUNDED PRECEDING AND 1 PRECEDING* sums all batches before the current one. The resulting “qty_before” tells each batch how much quantity the older batches have already absorbed. The COALESCE converts to zero the null that the window function returns for the first batch, which has no earlier-expiring predecessors. The third temporary table calculates each batch's actual consumption. The query is structured so that a batch can give at most its own quantity and thus never reach a negative amount. If a current batch cannot fully satisfy the demand, the query will directly address the second batch.
Batches which reach zero are directly copied into the batch archive and eliminated from the actual batch table. This preserves the history of the batches so that the system can reconstruct which batches were received and consumed. This helps to respect requirements under HACCP food safety regulation. Only after the temporary tables are dropped, the node recalculates the total quantity of each product in different batches, resynchronizing the quantity column in the restaurant inventory table, making the system fast to read.The workflow proceeds splitting the outcome into 2 different branches, one for expiry, the other for shortage. The expiry node searches for batches that expire in 3 days. If the query finds a batch that respects this characteristic, the

output will be parsed, to then enter a Gmail node which will send an alert to the kitchen, highlighting the near-to-expiry batches.

The shortage query is built like this:

```
SELECT
  ri.PRODUCT_ID,
  ri.NAME AS ProductName,
  COALESCE(SUM(b.quantity), 0) AS CurrentStock,
  ri.MIN_STOCK,
  ri.UNIT,
  s.sup_name AS SupplierName,
  s.sup_mail AS SupplierEmail,
  sp.supplier_product_code AS SupplierCode,
  sp.price_per_unit AS CatalogPrice,
  CEILING(((ri.MIN_STOCK * 2) - COALESCE(SUM(b.quantity), 0)) * 10) / 10
AS SuggestedOrderQty,
  s.notes AS SupplierNotes
FROM rest_inventory ri
LEFT JOIN batch b ON b.product_id = ri.PRODUCT_ID
LEFT JOIN (
  SELECT
    product_id, supplier_id, supplier_product_code, price_per_unit,
    ROW_NUMBER() OVER(PARTITION BY product_id ORDER BY
is_preferred DESC) AS rn
  FROM supplier_products
) sp ON ri.PRODUCT_ID = sp.product_id AND sp.rn = 1
LEFT JOIN suppliers s ON sp.supplier_id = s.supplier_id
GROUP BY ri.PRODUCT_ID, ri.NAME, ri.MIN_STOCK, ri.UNIT, s.sup_name,
s.sup_mail, sp.supplier_product_code, sp.price_per_unit, s.notes
HAVING COALESCE(SUM(b.quantity), 0) <= ri.MIN_STOCK;
```

The query starts from the restaurant inventory and performs a left join to the batch. This is done so that a product with no batches will still appear with a

stock of zero rather than not appearing from the result.
Current stock is computed as the COALESCE of the sum of the batches per product. The COALESCE ensures that a product with no batches, that is one completely out of stock, is correctly identified as being in shortage. PARTITION BY PRODUCT ID groups the supplier rows per product, and ORDER BY IS PREFERRED DESC places the preferred supplier first. The outer join condition sp.rn = 1 then keeps only that top row. This guarantees exactly one supplier per product (the preferred one) avoiding the row duplication that a full join on supplier products would cause when a product has several suppliers. A further LEFT JOIN to suppliers retrieves the supplier's name, email and notes for the order. Shortage is considered only when a product is equal or below the minimum threshold established in the database
Now the query calculates the reorder amount. The logic targets a replenishment level of twice the minimum stock: the difference between that target and current stock is the quantity to order. The CEILING rounds the result up to one decimal place, ensuring the order is never short. Ordering slightly more is safer than ordering slightly less for a stock-out-sensitive operation.
What is missing now is only the filtering of products that are actually in shortage. Since the current stock computed here is the sum of the different batches per product, only the products whose total stock across all batches are below the minimum threshold must be selected. The command HAVING COALESCE(SUM(b.quantity), 0) <= ri.MIN_STOCK is the one who gets this important function done.
Expiry and shortage are consciously separated because reordering is driven only by stock going under the minimum. An imminent expiry of an individual batch does not directly trigger an order, since there might be sufficient quantities of the same product in another batch.
Now, if the system does not find any shortage, it will return a mail saying to the restaurateur that all the inventory is well-stocked.
In the other case, if it finds shortages, the system will trigger another automatic email warning the team there are shortages. From here, input data such as the missing product id and name, its current and minimum stock, the supplier’s

name, mail and product code will be passed through a parse node. The catalogue price, the suggested order quantity and any supplier notes also pass into the parsing node. Logically, mails are grouped by supplier thanks to this parsing node. This procedure makes sure that if there are more products in shortage ordered to a single supplier, the system will create a unique draft containing more products at once. In summary, this node prepares the input for the ORDER MAKER AI agent.

An AI Agent in n8n is a workflow node that implements a Large Language Model, allowing it to interpret natural language, invoke external tools, make decisions and produce a result. There are 3 main components to an AI Agent, with the first one being the chat model, or the “engine” which interprets the prompt and generates results. In this case, Gemini is used.

As anticipated, the AI agent node offers a set of tools, better said additional functions that the AI agent can perform. In this case, the last component is an optional memory which allows the agent to remember the context of the various conversations. No memory was configured, as it would increase computational cost without altering the outcome for this single-shot task.

The AI agent operates on 2 different prompts: a system message, which defines the agent’s role and a user message, which underlines to the AI the specific tasks and the input data it must consider.

The user and system messages are reported below. They are written in Italian, since the system is designed for the Italian Ho.Re.Ca. sector.

-USER MESSAGE:

Componi il testo di una email d'ordine per il fornitore {{ $json.SupplierName }}:

{{ $json.ProductSummary }}

Note fornitore: {{ $json.SupplierNotes }}

-SYSTEM MESSAGE:

Ruolo: Sei l'Assistente Logistico del ristorante. Il tuo compito è comporre il testo di una email d'ordine professionale per il fornitore.

Linee Guida per le Email:
Approccio: Sii gentile e conciso.
Contenuto: Richiedi le quantità indicate per ciascun prodotto elencato.
Precisione: Cita sempre il codice articolo del fornitore per ogni prodotto.
Personalizzazione: Leggi attentamente le note fornitore. Se indicano orari preferiti o referenti (es. "Chiedere di Mario"), usali nella mail.
Firma sempre con: Cordiali saluti, Lo staff del Ristorante Boria's

Output richiesto: Restituisci ESCLUSIVAMENTE il corpo testuale della email, pronto da inviare. Nessun oggetto, nessun commento, nessuna spiegazione: solo il testo della mail.

The text generated from the AI agent is then created as a draft into Gmail thanks to the subsequent Gmail node. The information about the draft then passes through a parsing node to eliminate possible errors, creating a pending order with MySQL node. This is the query for doing this procedure, which will

link the first automation to the second.

```
INSERT INTO pending_orders
  (supplier_id, product_id, supplier_product_code, ordered_quantity,
ordered_unit, gmail_thread_id)
SELECT
  s.supplier_id,
  sp.product_id,
  '{{ $json.supplierCode }}',
  {{ $json.orderedQty }},
  '{{ $json.orderedUnit }}',
  '{{ $json.threadId }}'
FROM suppliers s
JOIN supplier_products sp ON s.supplier_id = sp.supplier_id
WHERE s.sup_mail = '{{ $json.supplierEmail }}'
AND sp.supplier_product_code = '{{ $json.supplierCode }}'
LIMIT 1;
```

⚠️ PRODOTTI IN ESAURIMENTO (5) Posta in arrivo ×

Davide Boria <boria.davide@gmail.com>
a me

18:00 (0 minuti fa)

I seguenti prodotti sono sotto la soglia minima di scorta:

Prodotto	Giacenza	Minimo	Da ordinare	Fornitore
Pasta (Spaghetti)	9.60 kg	10.00 kg	**10.4 kg**	DAVIDEFORNITORI
Uova	18.00 unità	30.00 unità	**42.0 unità**	DAVIDEFORNITORI
Pecorino Romano	2.88 kg	3.00 kg	**3.2 kg**	DAWEFOODS
Pomodori	6.00 kg	6.00 kg	**6.0 kg**	DAWEFOODS
Cipolle Bianche	2.00 kg	2.00 kg	**2.0 kg**	DAWEFOODS

Le bozze d'ordine sono state preparate automaticamente.

Figure 2.2-2: Automatic shortage alert

Figure 2.2-3: Automatic expiry alert

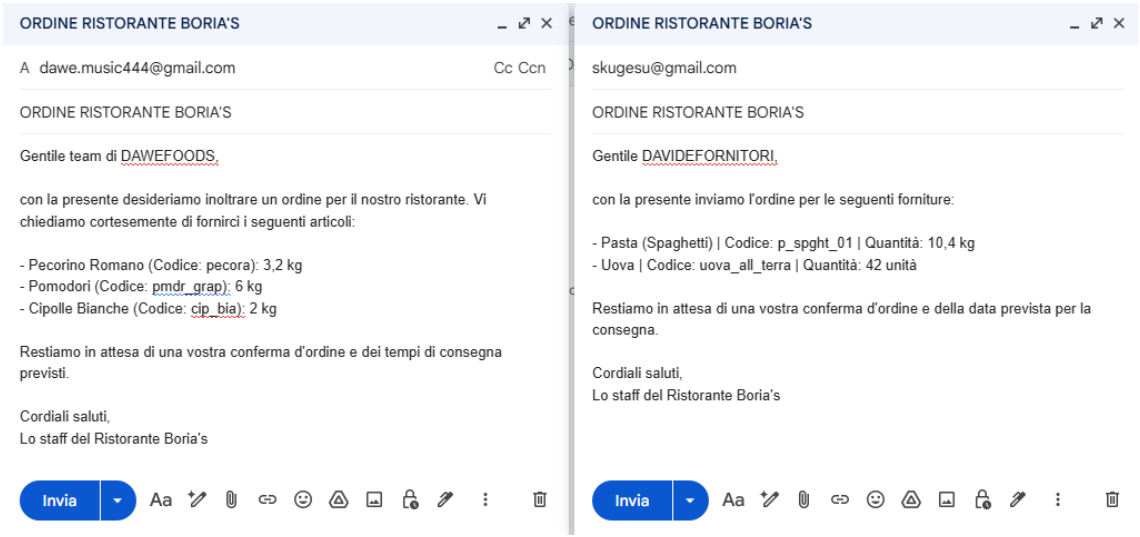
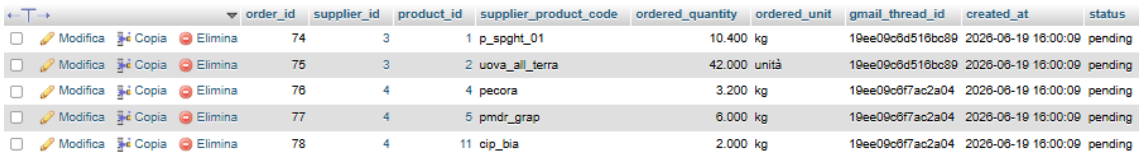


Figure 2.2-4: Two example automatic drafts to different suppliers – generated in the same automation run.



	order_id	supplier_id	product_id	supplier_product_code	ordered_quantity	ordered_unit	gmail_thread_id	created_at	status
Modifica Copia Elimina	74	3	1	p_spght_01	10.400	kg	19ee09c6d516bc89	2026-06-19 16:00:09	pending
Modifica Copia Elimina	75	3	2	uova_all_terra	42.000	unità	19ee09c6d516bc89	2026-06-19 16:00:09	pending
Modifica Copia Elimina	76	4	4	pecora	3.200	kg	19ee09c6f7ac2a04	2026-06-19 16:00:09	pending
Modifica Copia Elimina	77	4	5	pmdr_grap	6.000	kg	19ee09c6f7ac2a04	2026-06-19 16:00:09	pending
Modifica Copia Elimina	78	4	11	cip_bia	2.000	kg	19ee09c6f7ac2a04	2026-06-19 16:00:09	pending

Figure 2.2-5: Creation of the 5 different pending orders

As can be seen in the example, the run on first automation found five different products to be ordered. Pasta, eggs, pecorino romano, tomatoes and white onions. At first, the system generated the shortage alert, saying that there was a shortage of these five products (figure 2.2-2). Pecorino romano was the only batch to also be near to expiry, thus another alert mail was sent, warning the restaurateur (figure 2.2-3). Then, all this data was passed to the AI agent which wrote the 2 separate email texts that were created as drafts thanks to the Gmail *create draft* node (figure 2.2-4). The creation of the 5 different pending orders is the last step of the first automation (figure 2.2-5).

2.3 - AUTOMATION 2 - SUPPLIER REPLY AND BATCH INTAKE.

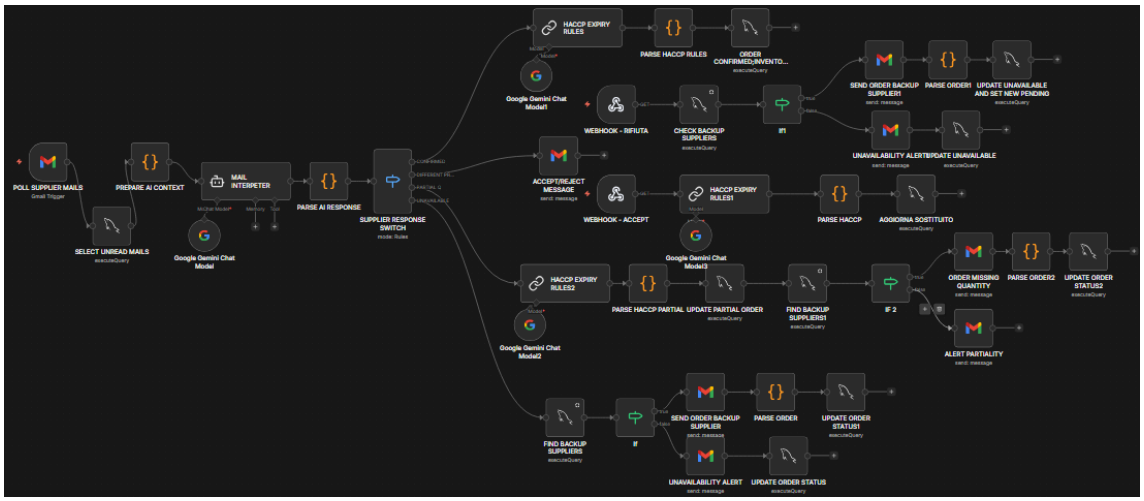


Figure 2.3-1: Automation 2 overview

The role of the second automation is to close the loop started by the first. The first subtracts stocks and places orders, this one receives and records them. After sending the draft emails to the suppliers, we wait for their response and, as soon as they send it, the automation begins. From there an AI Agent interprets the free text reply from the supplier. The workflow foresees 4 different options: the order can be fully confirmed, partially confirmed in quantity, substituted by a similar product or directly declined. The inventory will intake products accordingly. The real core element in this workflow is the Gmail thread id. This information resolves any potential ambiguity when referring to a single order, by referring to the “pending order” created by workflow one.
Now let’s dive into the details.
Everything starts with a polling trigger on Gmail. Polling means that the system periodically queries the Gmail account for newly arrived messages. In this case, the polling happens each minute, to make sure that no supplier response is lost. The output deriving from the Gmail polling passes directly into a MySQL node, which selects only the unread emails from suppliers. Here is the query:

```
SELECT
```

```
  s.supplier_id,
  s.sup_name,
  s.sup_mail,
  sp.product_id,
  sp.supplier_product_code,
  sp.price_per_unit,
  ri.NAME,
  ri.UNIT,
  ri.PRODUCT_ID AS rest_product_id,
  po.ordered_quantity,
  po.ordered_unit,
  po.order_id
FROM suppliers s
JOIN pending_orders po ON po.supplier_id = s.supplier_id
JOIN rest_inventory ri ON po.product_id = ri.PRODUCT_ID
JOIN supplier_products sp ON sp.product_id = po.product_id
  AND sp.supplier_id = po.supplier_id
WHERE po.gmail_thread_id = '{{ $json.threadId }}'
  AND po.status = 'pending'
  AND s.sup_mail = '{{ $json.from.value[0].address }}'
```

The query starts by selecting the supplier who sends the mail, joining on the products ordered and the pending orders. This allows the system to find the

order to which this reply belongs. To avoid ambiguity, the order thread id must be equal to the thread of the incoming mail, the sender's address equal to the matching supplier and the pending order must still be on "pending" status, because an order that was already confirmed must not be matched. All this mechanism allows primary and backup orders to be automatically processed by the system, avoiding human contribution. The query also pulls the list of the ordered product's specifics, functioning as a bridge with the next nodes.
All this data is then passed through a parsing node. Its job is to assemble the correct formatted user message prompt for the AI, combining two elements: the supplier's raw email body and the list of ordered products pulled by the previous query. This formatted prompt is given to the *MAIL INTERPRETER* AI Agent.
The choice of AI with parsed inputs and outputs is due to the unpredictability of the supplier's response. AI's semantic matching (free text interpretation) gives the system flexibility of execution but since AI is a probabilistic component, its output is not guaranteed to match the precise structure needed. These parsing nodes are so crucial because they make sure that we normalize the variable AI output into structured data.

The formatted user message and the predefined system message are the following:
USER MESSAGE:
"Sei un assistente che analizza email di risposta di fornitori alimentari italiani.\nIl fornitore risponde ad un ordine che conteneva PIU PRODOTTI — analizza la risposta per CIASCUN prodotto ordinato.\n\nEMAIL DEL FORNITORE:\n – QUA LA MAIL DEL FORNITORE \nPRODOTTI ORDINATI (analizza lo status di ognuno):\n- QUA LA LISTA DEI PRODOTTI ORDINATI \nISTRUZIONI:\n1. Per ogni prodotto nell'elenco determina lo status in base alla risposta del fornitore.\n2. confirmed = il fornitore lo consegna.\n3. different_product = il fornitore propone un alternativo.\n4. unavailable = il fornitore non può consegnarlo.\n5. partial = il fornitore conferma il prodotto ma in quantità inferiore a quella ordinata.In questo caso confirmed_quantity è la quantità che il fornitore può consegnare.\n6. Se la quantità o il prezzo non sono specificati per un prodotto lascia null.\n7. batch_number = il numero di lotto che il fornitore

comunica nella mail (es. \"lotto 2024-A\", \"LOT123\"). Se non lo comunica, lascia null.\n8. Usa la data odierna precisamente. Calcola delivery_date in base a questa: se il fornitore scrive \"domani\" aggiungi 1 giorno, \"dopodomani\" 2 giorni, ecc. Usa SEMPRE l'anno corrente o futuro, mai passato.\n\nRestituisci ESCLUSIVAMENTE un array JSON che inizia con [e termina con].\nNON avvolgere l'array in un oggetto contenitore. NON usare una chiave \"products\" o simili. NON aggiungere testo, spiegazioni o backtick.\nIl primo carattere della tua risposta deve essere [e l'ultimo deve essere].

SYSTEM MESSAGE:
Sei un assistente che analizza email di fornitori alimentari.

Rispondi SEMPRE e SOLO con un array JSON valido, che inizia con [e termina con].

Non avvolgere l'array in un oggetto contenitore, non usare chiavi come "products".

Nessun testo aggiuntivo, nessun markdown, nessun backtick.

Regarding the parsed final AI output, it contains structured information on the order status and ID, the product ID, price and quantity ordered. The batch number provided by the supplier, the delivery date and any alternative product details.
This output goes directly into a switch node, which will route the following automation into four different paths, one for each order status determined by the AI. As anticipated the four statuses are “confirmed”, “partial”, “substitution product” and “unavailable”.
Here are the two examples of supplier responses.

Figure 2.3-2: Supplier's free text response.

The first supplier, "DAVIDEFORNITORI" to which we ordered pasta and eggs, claimed that pasta can be fully confirmed, but that they can delivery only thirty eggs. The second supplier "DAWEFOODS" responded by saying that pecorino romano is completely unavailable and that they don't have alternatives; Tomatoes and white onions are also not available, but they offer an alternative for both. Resuming this part, pasta will pass on the "confirmed" branch, eggs in the "partial quantity", pecorino romano in the "unavailable" and tomatoes and white onions in the "substitution". Furthermore, each supplier is assumed to respond fully in text with each information that we need to complete the automation. The example of the previous section will continue during the following explanation of the four branches to ensure coherency.

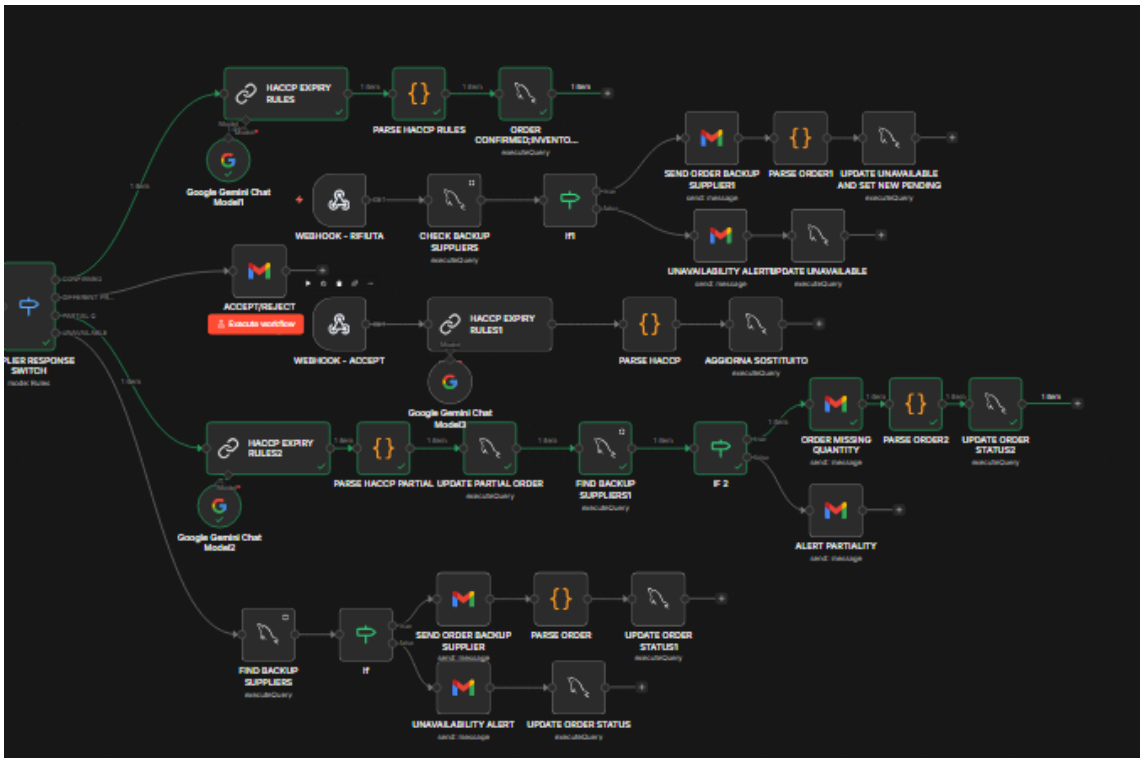


Figure 2.3-3: Confirmed and Partial quantity branches

The confirmed branch starts with a HACCP LLM node to estimate the expiry of the product. This Large Language Model is prompted to use and cite the specific HACCP regulation for each product category. This estimation is obviously a limitation of the system. In real kitchens, expiry is directly determined by the producer and must not be guessed. This could be implemented through an AI-based camera or barcode scanning that records specific information on the batches entering the inventory. Continuing with the workflow, the LLM output is parsed to ensure that the subsequent MySQL node receives formatted data. As can be seen from the image below, a new batch for the confirmed product (pasta) is inserted into the batch table. The MySQL node also sets the pending order status as confirmed, closing the loop. Then it resynchronizes the full inventory table recalculating the product's total quantity as the sum of its active batches and the unit cost as their weighted average. The partial quantity branch functions very similarly to the confirmed one, with

the difference that, after having updated the partial quantity product like the confirmed branch does , it searches for backup suppliers for ordering the missing part. Since the product in question(eggs) has a backup supplier, an automatic order is sent and a new pending order is created in the database.

	batch_id	batch_number	product_id	initial_quantity	quantity	expiry_date	unit_cost	supplier_id	received_date	note
Modifica Copia Elimina	62	UOVA_02	2	30.000	30.000	2026-07-21	0.2500	3	2026-06-22 15:09:46	NULL
Modifica Copia Elimina	61	PASTA_02	1	10.400	10.400	2027-06-23	1.9000	3	2026-06-22 15:09:46	NULL

Figure 2.3-4: Two batches' invoices: Pasta fully confirmed and eggs partially confirmed in quantity.

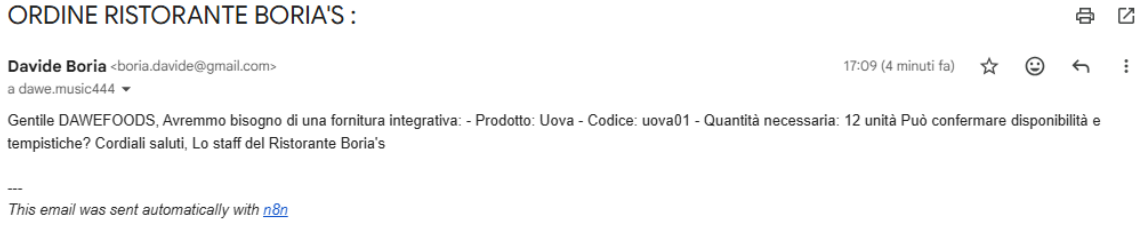


Figure 2.3-5: Automatic backup order to second supplier.

Now the substitution product branch will be introduced.

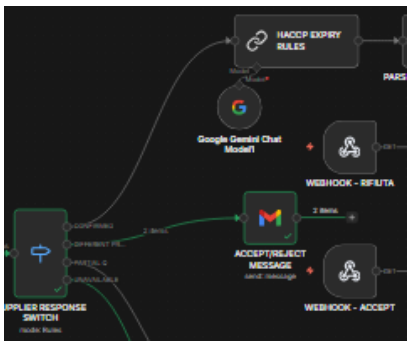


Figure 2.3-6: Substitution product branch. Creation of the two substitution alerts.

If the AI detects a substitution proposal from the supplier, it will automatically send a mail through which the restaurateur can accept or decline the proposal. Here are the two different proposal mails for the products in the example. Let

the substitute product for white onions be refused, and the one for tomatoes accepted.

Figure 2.3-7: Substitution product alerts: "Cipolle" are rejected. "Pomodori" are accepted.

The two emails contain the "accept" and "reject" URLs that will let the following automation start. A webhook is the trigger for the two paths. This type of trigger waits for a specific URL to be called. Only after the URL has been called, it makes all the subsequent automation begin.

Starting from the reject path, after the webhook URL has been called, a MySQL node searches for backup supplier, as in the partial quantity branch. This time it did not find any backup supplier, leading to an automatic email that alerts the restaurant. The alert mail suggests a manual and traditional approach of intaking the missing product. Then the order status is set as "unavailable", closing the loop. This type of procedure is triggered in each part of the automation where a product that is not available (fully or partially) has no backup supplier.

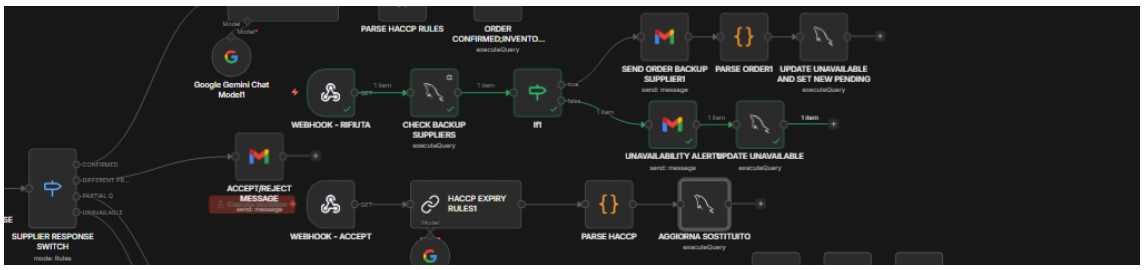


Figure 2.3-8: Reject path

Figure 2.3-9: No backup supplier alert.

Proceeding with the “accept” path, the substituted product is registered as the original product, with the same method with which we update products in the “confirmed” branch. The LLM chain estimates the expiry date, then its output is passed through a parsing node, and only then the substituted batch is uploaded in the database. Here, as figure 2.3-10 shows, traditional tomatoes are substituted with “pomodori del piennolo”, with the correct pricing and expiry date.

batch_id	batch_number	product_id	initial_quantity	quantity	expiry_date	unit_cost	supplier_id	received_date	note
Modifica Copia Elimina 63	pomopien	5	6.000	6.000	2026-07-02	1.5000	4	2026-06-24 09:33:15	Sostituto con pomodori del piennolo, accettato il ...

Figure 2.3-10: Accept substitution and new substituted batch identified by the note

The last branch to be described is the “unavailable” branch.

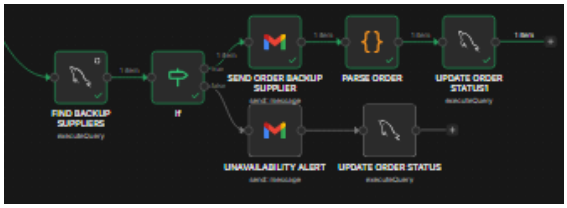
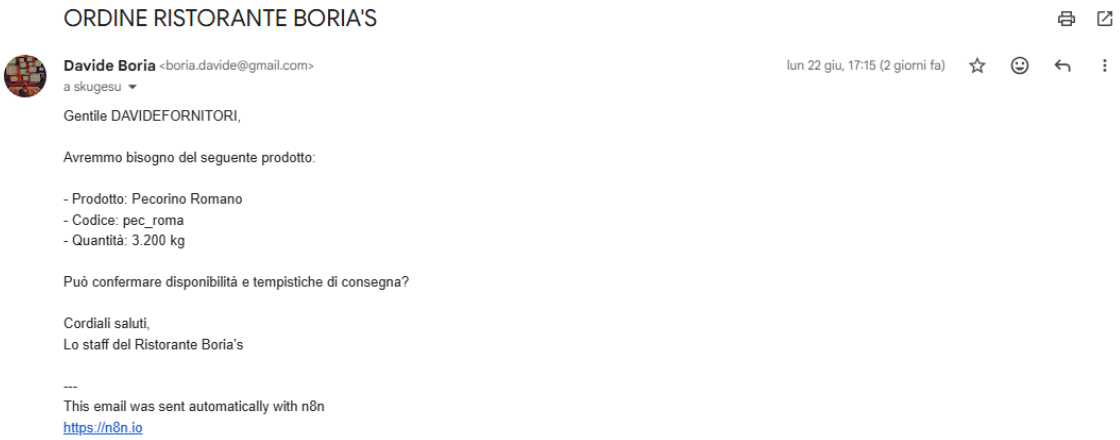


Figure 2.3-11: Unavailable product branch

The product cannot be delivered at all. The system searches for backup suppliers and, if one exists, it automatically sends an order mail to them, saving a new thread id and pending order, so that the reply enters back in the workflow. If it does not find any backup supplier, the procedure is the same as the one explained in the reject path of the substitution product branch. Figure 2.3-12 shows the automatic backup order.


ORDINE RISTORANTE BORIA'S

Davide Boria <boria.davide@gmail.com>
a skugesu

lun 22 giu, 17:15 (2 giorni fa)

Gentile DAVIDEFORNITORI,

Avremmo bisogno del seguente prodotto:

- Prodotto: Pecorino Romano
- Codice: pec_roma
- Quantità: 3.200 kg

Può confermare disponibilità e tempistiche di consegna?

Cordiali saluti,
Lo staff del Ristorante Boria's

This email was sent automatically with n8n
https://n8n.io

Figure 2.3-12: Unavailable product automatic order to backup supplier.

Concluding the explanation, despite four different branches, the system converges on two outcomes. Either a new stock enters the inventory as a traceable batch, or a new order re-enters the supply cycle, making sure that no order is left behind.

3. CONCLUSION

This thesis presented the design and the implementation of an automation system for Ho.Re.Ca sector procurement, built on the n8n platform integrated with a relational database. The objective of the project was to automate restaurant inventory management at each stage, from monitoring stock levels to placing orders and recording incoming products. This was achieved through two workflows. The first one deducts ingredients from the batches in inventory according to dishes sold and automatically drafts email orders when the available stock reaches its minimum threshold. The second workflow polls the email response from suppliers and interprets them thanks to an AI agent and updates the inventory with traceable product batches. The orders can fall in four different categories that, apart from straightforward confirmed orders, include also partial deliveries, substitution and unavailability.

In light of the results obtained, the objectives set at the beginning can be considered fully achieved. The implemented system automates the procurement processes of a restaurant in an effective way. With its complete and self-closing loop, from shortage detection, through order drafting, to the interpretation of supplier's reply and the update of inventory, the system does exactly what it was supposed to do.

Since the complete process is based on various assumptions, and since the Ho.Re.Ca sector is full of unpredictable events, the system is not to be considered ready for use, but it can of course be tested on real data on a daily basis.

Several future developments could be implemented. On the operational side, a mobile application for taking orders to diners could be developed and linked to the database, so that the loop starts even earlier and no manual orders have to be inserted.

On the technical side, the system should implement a QR or bar code-based solution to securely record real products' expiry date, The system may also be enriched with reporting and analytics on food cost and supplier performance, building on the weighted average cost data it already records.