
Business Process Integration Lab Project

Group 17:

Dani Mahaini (s2746557)
Alan Nessipbayev (s2648997)
Gabriela Todorova (s2696177)
Miglena Pavlova (s2717972)

15.12.2024

Business Information Technology MSc

UNIVERSITY OF TWENTE.

Table of Contents

Main Business Process and Case Description	2
Primary Process and Project aim	3
Project Framework and Foundations	4
Domain model	4
Internal processes	5
Dummy projects to test the ability to integrate	6
External processes	7
Data retrieval	7
Data posting	7
Truck ETA notifications	7
Time API	7
Evaluation and Conclusion	8
Appendix A - User manual and instructions to test	9
Data retrieval testing	9
Internal logic testing	11
Data posting testing	16
Truck ETA testing	18
Time API testing	19
Appendix B - Groups connected with	20

Main Business Process and Case Description

Uniprocterleverage is a large consumer goods supplier that manages deliveries to supermarkets across the Netherlands. The process starts when an order is received, prompting Uniprocterleverage to request transport quotes from multiple transport companies. These companies respond with their best prices, and Uniprocterleverage selects the most cost-effective option, issuing a transport order to the chosen company. If the load is less than a full truckload (LTL), the transport company negotiates with other companies to combine loads, making the transport more efficient.

Once arrangements are confirmed, the transport company requests truck movement approval from the Control Tower, which oversees truck logistics. The Control Tower calculates loading and staging times, notifying the distribution center to prepare for the truck's arrival. At the distribution center, trucks may wait in a queue if capacity is full, but once space opens, the truck is loaded and departs for the supermarket. During transit, the transport company requests regular location updates from the Control Tower. Upon delivery, the transport company confirms receipt with the supermarket and notifies Uniprocterleverage that the order is complete, marking the end of the process. This streamlined workflow integrates coordination between Uniprocterleverage, the transport companies, the control tower and the distribution center to ensure efficient and reliable deliveries. The whole process has been modeled and can be seen in the BPMN in Figure 1.

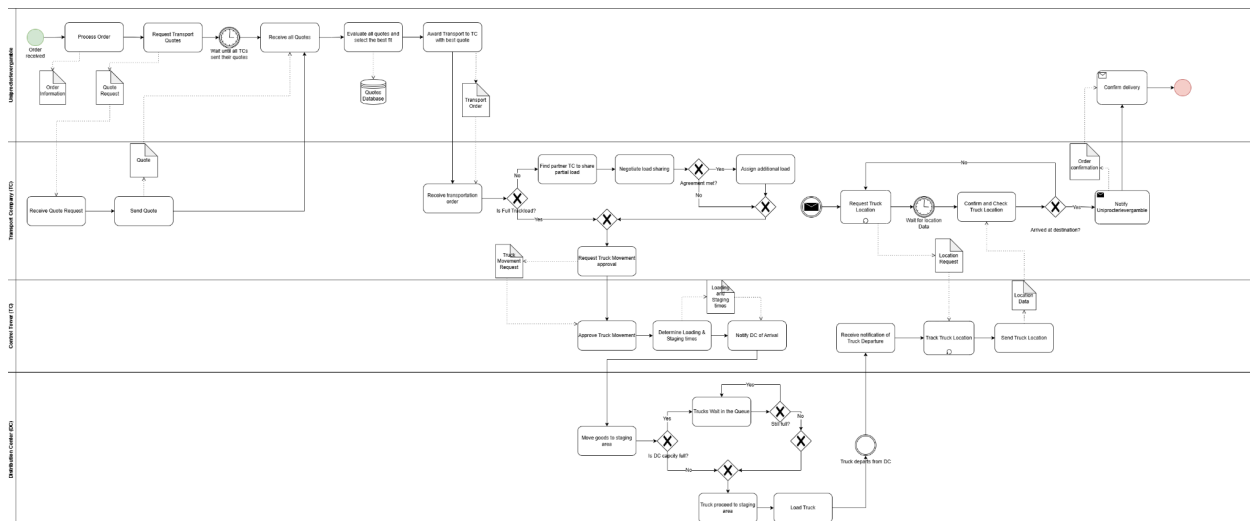


Figure 1: Business Process Model of order fulfillment and transport process for Uniprocterleverage

Note: Figure 1 is also attached to the submission as a zoomable png file with better resolution (it is called “BPMN of the whole business process”).

Primary Process and Project aim

Figure 2 illustrates the distribution center's business process in greater detail, serving as our chosen actor and the central focus of this project. The primary objective is to develop a robust web service that not only supports the functionality outlined in the case description but also supports seamless business integration with external stakeholders, such as transport companies and control towers. To achieve this, Mendix has been chosen as the main tool. Mendix's low-code platform enabled rapid prototyping, efficient iteration, and allowed collaboration among stakeholders by offering REST, OData and SOAP services. This ensured that the solution aligns closely with the business needs while maintaining scalability and adaptability to all involved stakeholders. As for the modelling tool, draw.io has been used to create the BPMNs represented in Figures 1 and 2.

Additionally some assumptions were made while developing the application. One key assumption made was the capacity of the distribution center, or in other words the number of trucks that can be accommodated inside the distribution center. For this, a capacity of six trucks was selected as a reasonable estimate. Following the previous assumption, an estimation for the amount of staging areas within the distribution center has also been selected to be six. Furthermore, the application's logic does not support a limitation in queue sizes, hence the assumption is that there may be infinitely many trucks in the queue (this of course does not happen in real world), this is due to the fact that we do not have the additional information to determine the maximum capacity for the queues.

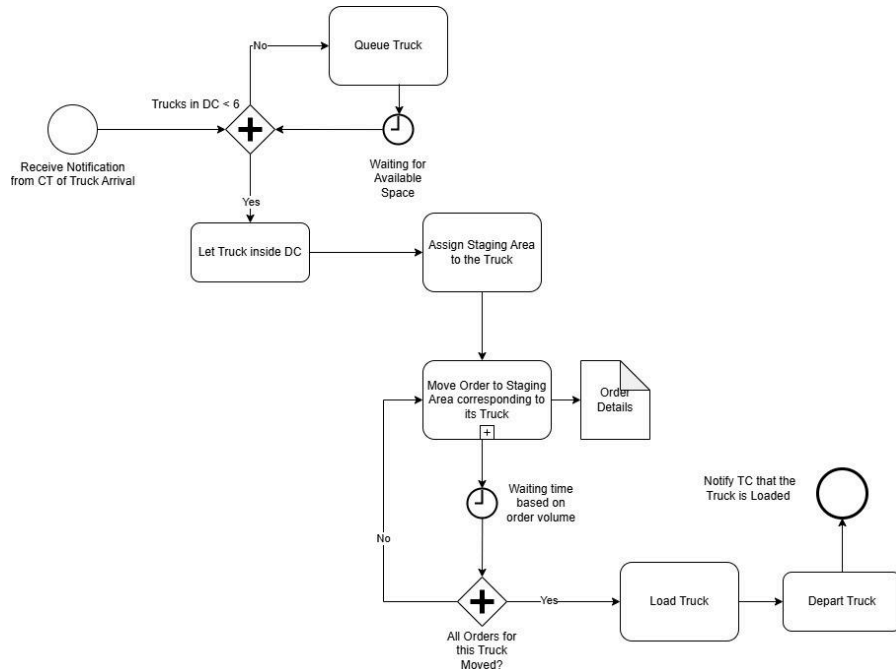


Figure 2: Business Process Model of the Distribution Center's primary process

Project Framework and Foundations

Domain model

The Mendix domain model in Figure 3 visually represents the end-to-end order delivery and truck logistics process. The Order entity manages key details like the order ID, delivery date, destination address, and the truck loading status, while being connected to Product for SKU and quantity tracking. Orders are staged in the StagingArea, which monitors space availability and load capacity. The Truck entity handles logistics information such as truck location, loaded weight, ETA, and queue positions, linking to the Order and StagingArea for efficient truck loading and movement. External communication with the Control Tower is facilitated through the Root entity, capturing truck details and ETAs. The TruckReadyNotification entity signals truck departure with loaded weights and pallets, while the Input and Time entities integrate API data for time calculations and city inputs, ensuring accurate scheduling and real-time updates. Finally, the purple entity Truck ETAs represents the external table of the control tower, which is connected to our databases by using OData standards and contains data about trucks' upcoming arrivals.

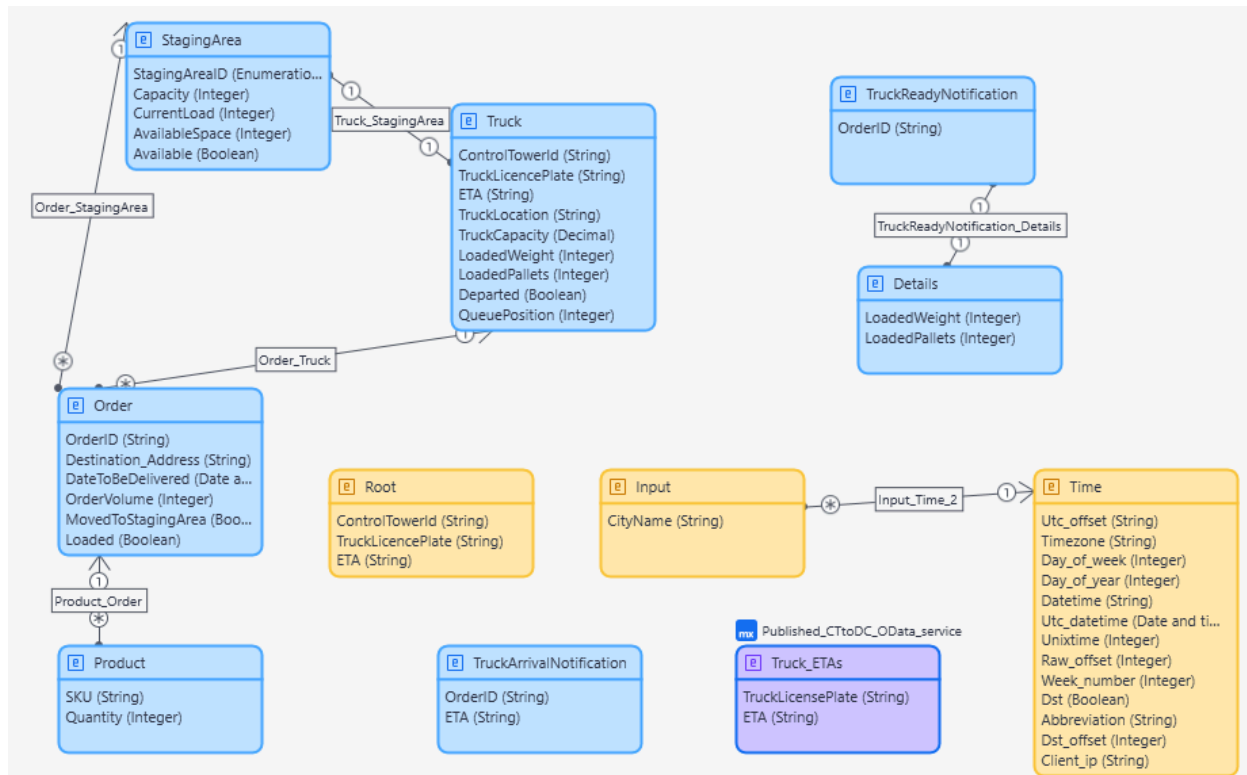


Figure 3: Mendix's application domain model

Internal processes

The internal process logic of our Mendix application is designed to manage truck arrival, staging, and loading efficiently. When a new truck arrives, the system checks if there are fewer than six trucks in the distribution center. If space is available, the truck is allowed inside, and a staging area is assigned to it using the “AssignStagingAreaToTruck” microflow. Alternatively, if there is not enough space the truck will be set in a queue, the microflow “AssignQueuePos” will assign a queue position for all of the trucks that could not fit in the distribution center. Orders are then moved to their respective staging areas via the “MoveOrderToStagingArea” microflow, with waiting times calculated based on order volume. Once all orders for a truck are moved, the “LoadOrderToTruck” microflow ensures the truck is loaded. The truck is then marked as departed using the “DepartTruck” microflow and the queue positions of the other trucks (which are not inside the distribution center yet) are updated using the “UpdateQueuePos” microflow.

To keep the system clean and efficient, additional microflows such as DeleteOrder, DeleteStagingArea, and DeleteTruck are used to remove outdated or processed data. The UnloadOrder microflow is available to handle any unloading requirements, ensuring flexibility in case of changes. This structured logic ensures smooth coordination of truck staging, loading, and departure processes while maintaining data consistency and operational efficiency. Additionally, the application supports error messages and advanced logic to handle inappropriate input or unexpected scenarios, ensuring that processes remain robust, data integrity is maintained, and users receive clear feedback to resolve any issues efficiently.

Dummy projects to test the ability to integrate

Once the foundational logic of the application (internal processes) was implemented, the next step was to assess whether the application was prepared for integration with external stakeholders. Such integration is crucial for ensuring smooth collaboration and data exchange between the application and external systems like those used by transport companies or control towers. To set up this evaluation, two additional Mendix projects were developed to simulate external systems: one serving as a provider for truck data and the other for order data. These projects acted as test environments to emulate real-world interactions, ensuring the main application could handle external data inputs effectively.

The communication between these additional applications and the main application was achieved through OData services, a standardized protocol widely supported in Mendix for querying and exchanging data. To enable this integration, external entities were introduced into the domain model of the main application (note that these entities are not included in the final domain model shown in Figure 3 and are shown in Figure 4). These external entities functioned as proxies for the tables in the external databases, mirroring their structure and providing a bridge for data exchange. Mendix's built-in features, combined with additional microflows, were used to retrieve data from these external entities and populate our own application's database.

The integration process demonstrated the readiness of the application to interact with external stakeholders. By adopting this approach, the project showcased the effectiveness of OData services in enabling robust, scalable, and real-time data exchange across distributed systems, providing a strong foundation for future integrations.

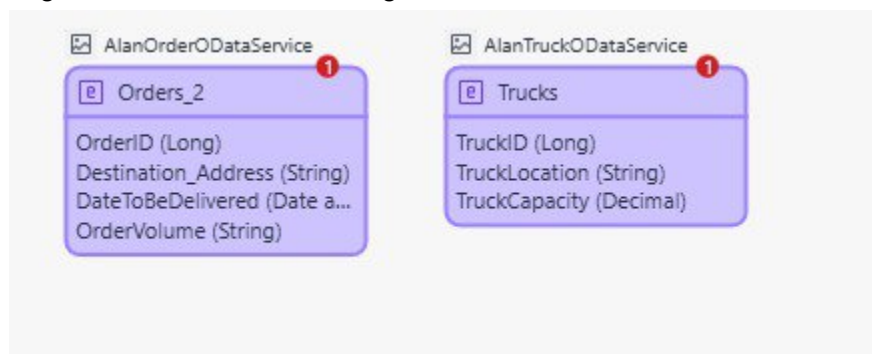


Figure 4: External data entities from old domain model

External processes

Data retrieval

The data retrieval process at the Distribution Center begins with receiving truck information from the Control Tower via a REST request at the endpoint `/rest/share-eta`. This request contains details such as the `ControlTowerId`, `TruckLicencePlate`, and a list of associated Orders. Each order includes the `OrderId`, product details (like `SKU` and `Quantity`), and the estimated time of arrival (ETA). Upon receiving this information, the system processes the data and displays it on the overview pages. This enables the Distribution Center to identify which orders need to be loaded onto the specified truck, streamlining the loading process and ensuring accurate order management.

Data posting

The Data Posting process involves notifying the Control Tower when a truck with a specific order has been loaded and is ready to depart. This is achieved by sending a POST request to the endpoint `/rest/notify-truck-ready` with a JSON body containing the `OrderId` to identify the order, along with additional details such as the `LoadedWeight` and the number of `LoadedPallets`. The request includes the `Content-Type` header set to `application/json` to ensure proper formatting. The Control Tower should process the request to update its records and confirm the truck's status. If the data is sent correctly and matches the expected format, it should respond with a success message, such as "Notification received successfully".

Truck ETA notifications

One of the final communication matters between the distribution center and control tower is the notification about the estimated time of arrival of a truck. The initial purpose is to notify the distribution center that a certain truck is arriving in 15 minutes. This had to be done using SOAP standard, however due to the time constraint and inability to test the integration with other stakeholders we have decided to implement this functionality using OData standard. We have created another dummy data Mendix project, which sends data (notification) to our main application about truck arrivals.

Time API

The Time API is essential in the application to ensure accurate scheduling and coordination of truck movements, loading times, and deliveries. By providing real-time data such as the current time, day of the week, and time offsets across different time zones, it enables precise calculations for ETAs, truck staging, and loading schedules. This guarantees that all processes align with the delivery timeline, reducing delays and improving overall efficiency. The Time API is a consumed web service available on the internet (<http://worldtimeapi.org/>), allowing the application to retrieve up-to-date time-related information dynamically, ensuring reliability and synchronization across various components of the logistics workflow.

Evaluation and Conclusion

Throughout the development of this project, our team encountered numerous challenges that required swift and effective resolution due to the limited timeframe. One of the first difficulties arose during the initial stages when selecting an actor from the provided case description. While the project description initially seemed straightforward, closer examination revealed complexities that were not immediately apparent. After thorough consideration, we chose the Distribution Center as our primary actor, as implementing functionalities related to trucks, orders, and queues in a web service appeared both engaging and educational. Fortunately our team had enough experience with modelling business processes, which allowed us to identify and design the core processes without significant obstacles.

However, greater challenges surfaced as we began working in Mendix. Being unfamiliar with this low-code development platform, all team members had to dedicate extra time to complete introductory courses and familiarize themselves with the environment. While developing the internal processes and logic was time-intensive, it was not the most significant hurdle of the project. The real difficulties emerged after completing the internal logic, when we needed to determine where, what, and how to integrate our application with those of other groups. This phase demanded additional research into how Mendix handles external data and web services, including the use of SOAP, OData, and REST. Creating dummy external projects to simulate data providers was a crucial step, as it confirmed that our application was ready for integration with other teams.

The greatest challenge of the project, however, lay in its core objective: business integration! Each group developed its own applications, solutions, and strategies, leading to differing visions for integration, communication protocols, and service usage. This created a scenario akin to real-world stakeholder conflicts, where differing priorities and approaches needed to be coordinated properly. Despite these difficulties, we collaborated closely with the other groups, engaging in consistent communication and negotiation to develop a shared integration plan. As a result, we successfully adapted our project to use REST services for communication with external stakeholders, ultimately transitioning entirely from OData to REST.

In conclusion, while the path to completing this project was filled with challenges, it was also rich in opportunities for growth and learning. We not only enhanced our technical skills, particularly in web services and low-code development with Mendix, but also gained valuable experience in team collaboration and intergroup communication. These lessons extended beyond technical knowledge, teaching us the importance of adaptability and compromise when working toward a shared goal. Ultimately, the project was not just about building a functional web service but about navigating the complexities of integration and teamwork—skills that are essential in any real-world scenario. The experience has prepared us to tackle similar challenges in the future with greater confidence and competence.

Appendix A - User manual and instructions to test

Data retrieval testing

Using Postman it can be tested that our mendix application successfully receives and processes data sent via a REST API from the Control Tower. The test should target the endpoint <http://localhost:8080/rest/share-eta> with a POST method and a JSON payload as the input. The request was properly configured with the Content-Type header set to application/json to ensure the server correctly interprets the data format.

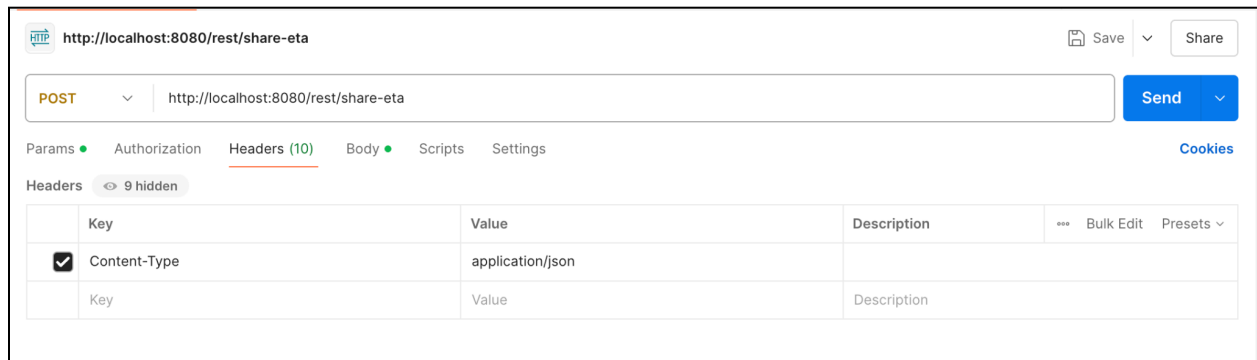


Figure 5: Postman overview

The payload should contain crucial details, including the ControlTowerId ("CT01"), the truck's license plate TruckLicencePlate ("TC06ABC") and an Orders array with an OrderID ("AB123"). The order should also include product information, such as the SKU ("PT-001") and Quantity (2), along with an ETA ("2024-12-10T16:26:50Z") specifying the truck's estimated arrival time.

```
{
  "ControlTowerId": "CT01",
  "TruckLicencePlate": "TC06ABC",
  "Orders": [
    {
      "OrderID": "AB123",
      "Products": [
        {
          "SKU": "PT-001",
          "Quantity": 2
        }
      ]
    }
  ],
  "ETA": "2024-12-10 16:26:50Z"
}
```

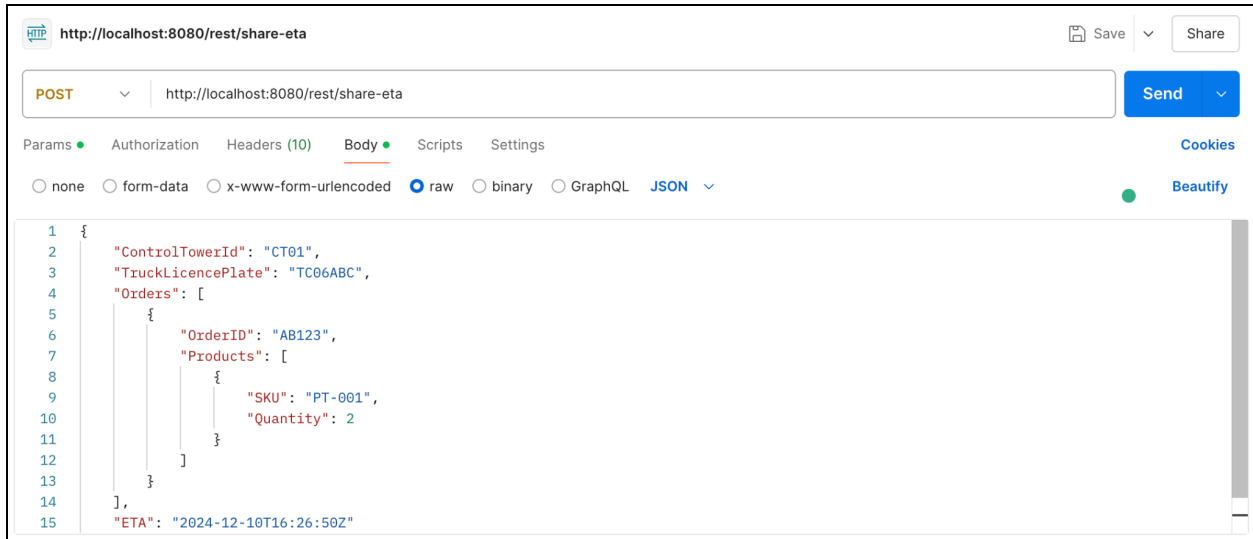


Figure 6: Postman JSON structure

As a result of sending the request the server should respond with a 200 OK status and a success message in the body, indicating that the payload was successfully received and processed.

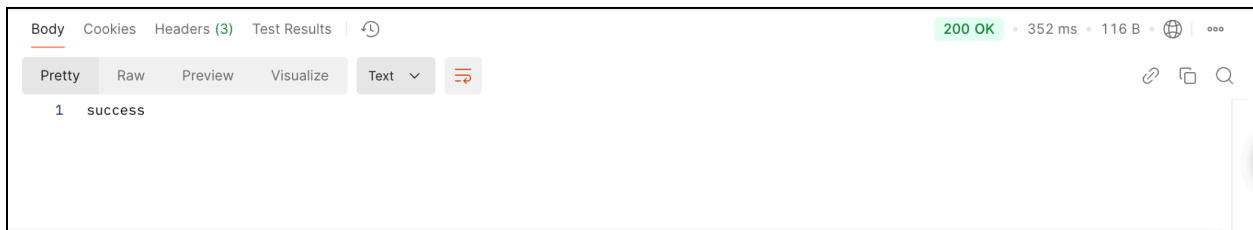


Figure 7: Postman success message

Upon completion of the data processing, the information should also appear in the Mendix application interface under the "Trucks" section. As shown in the application, the truck with the plate number "TC06ABC" now appears in the list, along with its corresponding ETA.

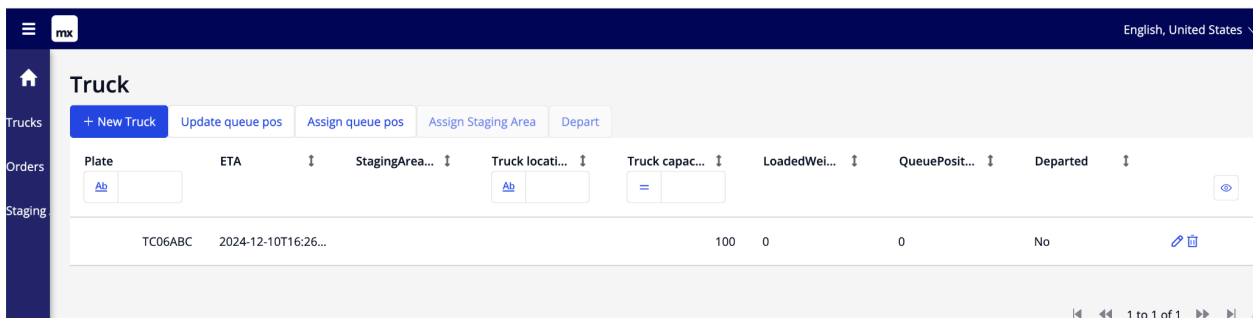


Figure 8: Trucks overview with the new truck object

Also under the "Orders" section, the order should appear alongside its attributes, such as OrderID, Order volume, and the corresponding TruckID.

Order ID	Order volume	Moved to st...	StagingArea...	Loaded	TruckID	Departed	Products	Edit
AB123	2	No		No	TC06ABC	No	Show Products	

Figure 9: Orders overview with the new order object

Internal logic testing

The internal logic of the system can be tested by first sending POST requests to store trucks and their associated orders.

Plate	ETA	StagingArea...	Truck locati...	Truck capac...	LoadedWei...	QueuePosit...	Departed
BE09435	2024-09-02 16:26:...			100	0	0	No
BG09435	2024-09-02 16:26:...			100	0	0	No
BT09435	2024-09-02 16:26:...			100	0	0	No
ET09455	2024-09-02 16:26:...			100	0	0	No
GG09455	2024-09-02 16:26:...			100	0	0	No
FH09455	2024-09-02 16:26:...			100	0	0	No
JJ09455	2024-09-02 16:26:...			100	0	0	No

Figure 10: Trucks overview with populated trucks

Once the trucks have been stored, queue positions can be assigned using the "Assign queue pos" button. This action updates the queue position for each truck. After the queue positions have been assigned, the "Update queue pos" button can be used after a truck leaves the distribution center, this will reduce the queue positions of the trucks by 1 and will move the first

truck from the queue to the "distribution center."

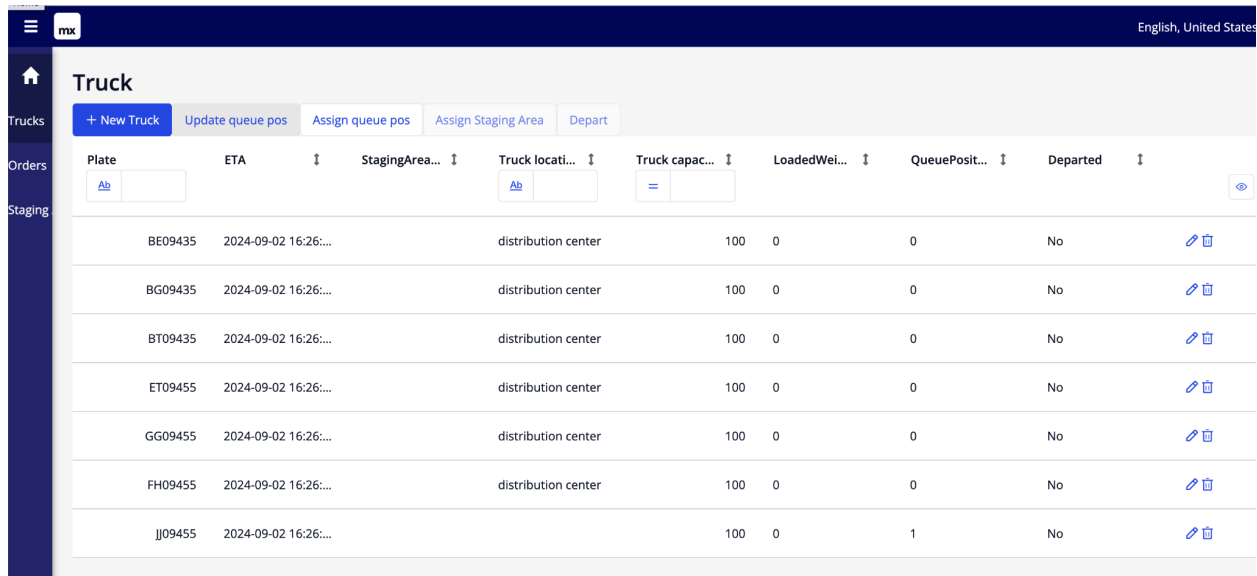


Plate	ETA	StagingArea...	Truck locati...	Truck capac...	LoadedWei...	QueuePosit...	Departed
BE09435	2024-09-02 16:26:...		distribution center	100	0	0	No
BG09435	2024-09-02 16:26:...		distribution center	100	0	0	No
BT09435	2024-09-02 16:26:...		distribution center	100	0	0	No
ET09455	2024-09-02 16:26:...		distribution center	100	0	0	No
GG09455	2024-09-02 16:26:...		distribution center	100	0	0	No
FH09455	2024-09-02 16:26:...		distribution center	100	0	0	No
JJ09455	2024-09-02 16:26:...			100	0	1	No

Figure 11: Trucks overview with 1 truck in the queue

The system restricts the number of trucks allowed in the distribution center to 6 at any given time. If we attempt to move a 7th truck while the maximum capacity is reached, the operation will fail. This can be verified by clicking the "Update queue pos" button again, which will prevent further trucks from entering until a spot becomes available.

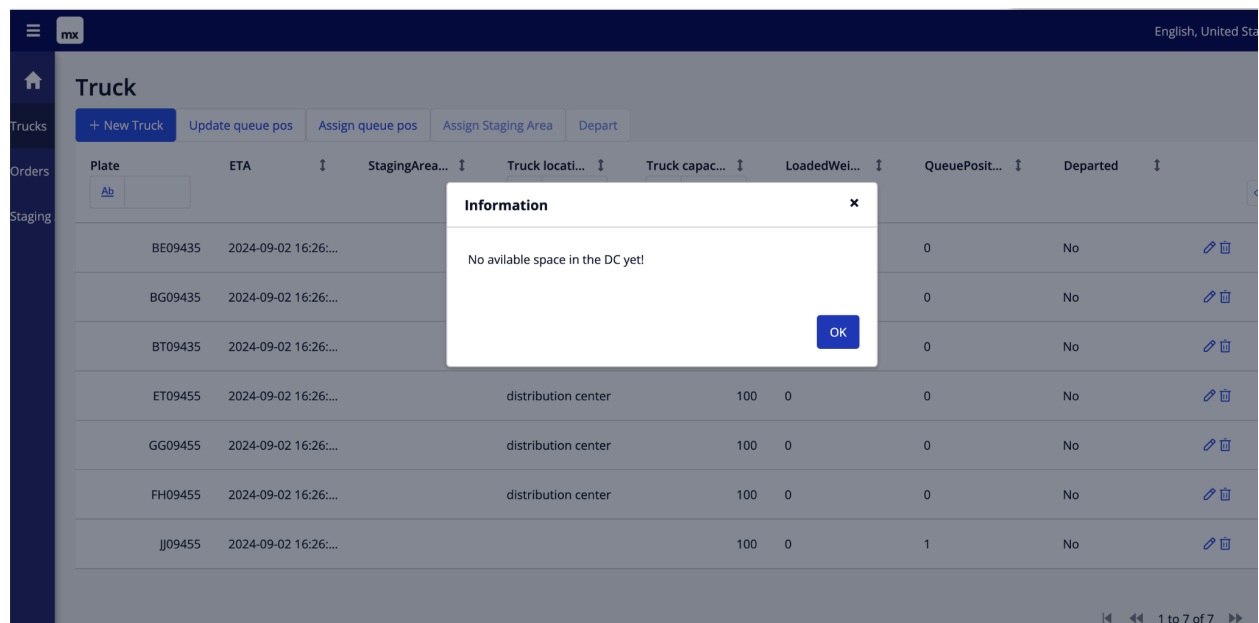


Figure 12: No available space error pop-up

Once a truck has entered the distribution center, a staging area can be assigned to it. To do this, we must first create staging areas. This can be done in the "Staging Area" section using the "New Staging area" button.

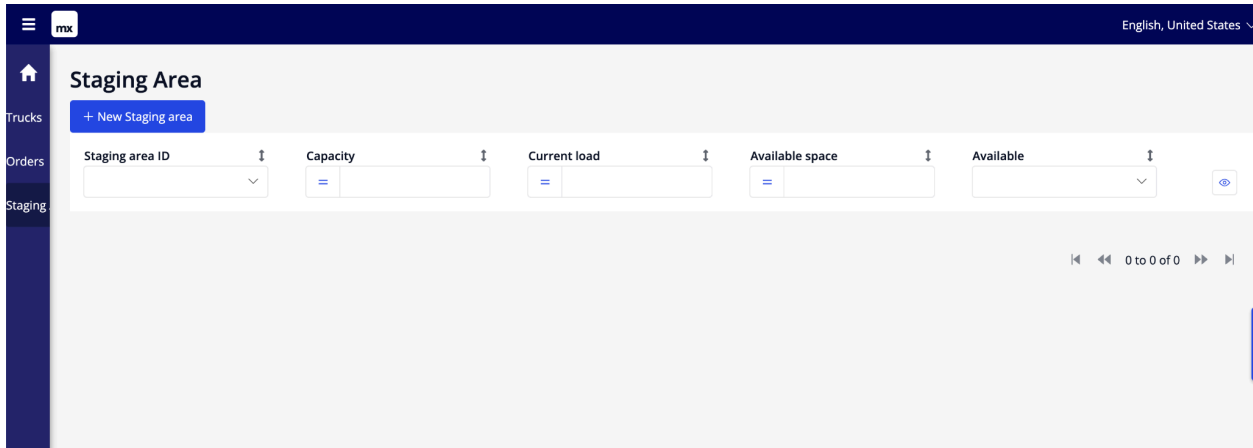


Figure 13: Staging Area overview

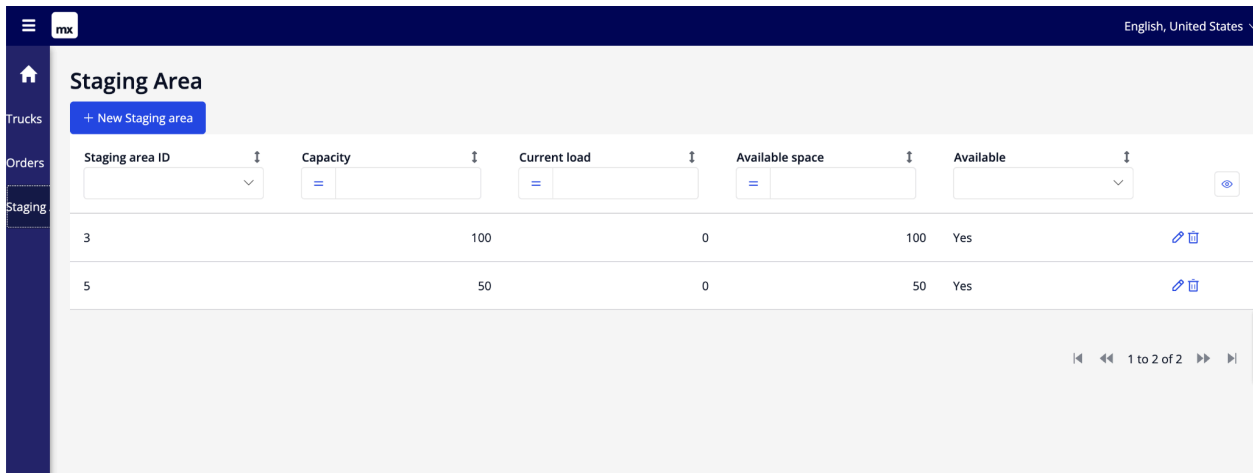


Figure 14: Staging Area overview with created staging areas

After creating the necessary staging areas, they can be assigned to trucks in the "Trucks" section using the "Assign Staging Area" button. For example, truck "BE09435" can be assigned to staging area 3.

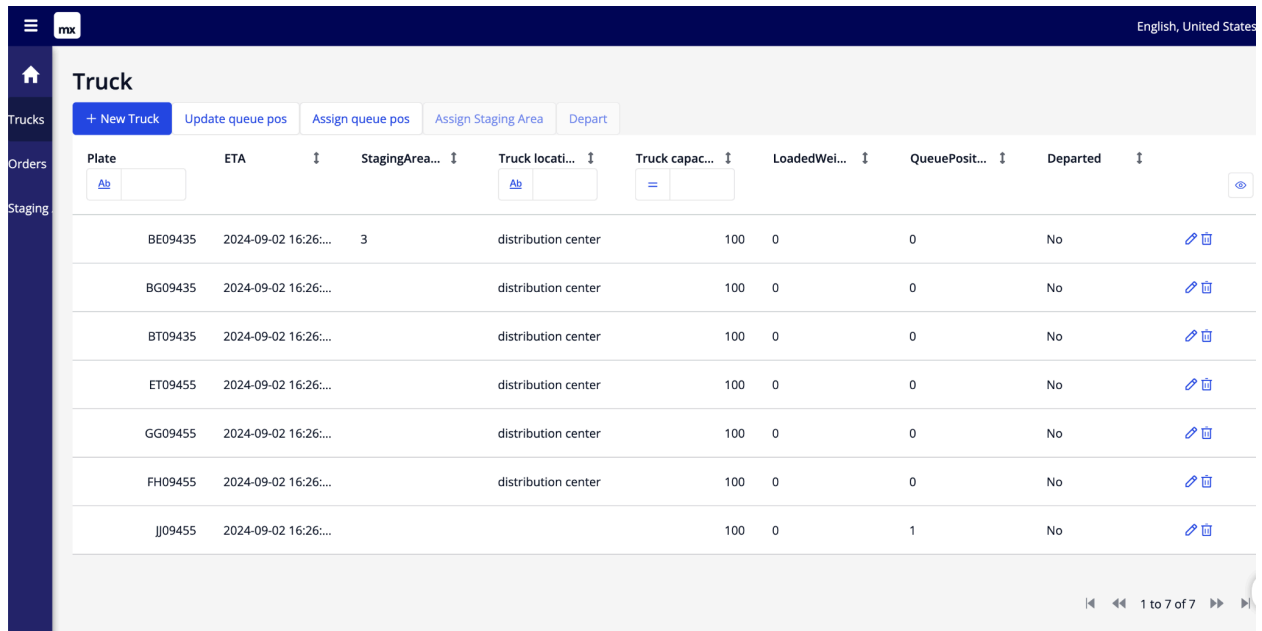
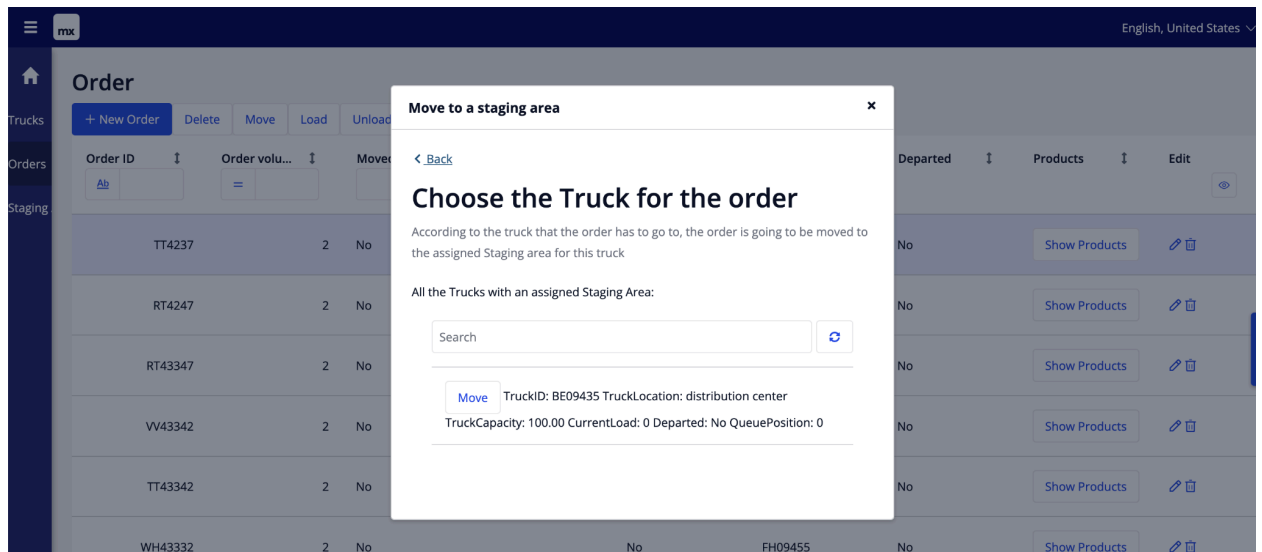


Plate	ETA	StagingArea...	Truck locati...	Truck capac...	LoadedWei...	QueuePosit...	Departed
BE09435	2024-09-02 16:26:...	3	distribution center	100	0	0	No
BG09435	2024-09-02 16:26:...		distribution center	100	0	0	No
BT09435	2024-09-02 16:26:...		distribution center	100	0	0	No
ET09455	2024-09-02 16:26:...		distribution center	100	0	0	No
GG09455	2024-09-02 16:26:...		distribution center	100	0	0	No
FH09455	2024-09-02 16:26:...		distribution center	100	0	0	No
JJ09455	2024-09-02 16:26:...			100	0	1	No

Figure 15: Trucks overview with 1 truck assigned to a staging area

Once staging areas have been assigned to trucks, orders can be moved to their corresponding staging areas. This process is performed in the "Orders" section using the "Move" button. A window will appear displaying the appropriate staging area based on the truck to which the order belongs. After an order has been moved, the system updates the staging area's current load, reducing the available space accordingly.



Move to a staging area

< Back

Choose the Truck for the order

According to the truck that the order has to go to, the order is going to be moved to the assigned Staging area for this truck

All the Trucks with an assigned Staging Area:

Search

Move TruckID: BE09435 TruckLocation: distribution center
TruckCapacity: 100.00 CurrentLoad: 0 Departed: No QueuePosition: 0

Figure 16: Move order to staging area

Staging area ID	Capacity	Current load	Available space	Available
3	100	2	98	Yes
5	50	0	50	Yes

Figure 17: Staging area overview with current load and available space changed

When the truck and its orders are ready, the orders can be loaded into the truck using the "Load" button in the "Orders" section. This action updates the LoadedWeight attribute in the section "Trucks" for the respective truck.

Plate	ETA	StagingArea...	Truck locati...	Truck capac...	LoadedWei...	QueuePosit...	Departed
BE09435	2024-09-02 16:26:...	3	distribution center	100	2	0	No
BG09435	2024-09-02 16:26:...		distribution center	100	0	0	No
BT09435	2024-09-02 16:26:...		distribution center	100	0	0	No
ET09455	2024-09-02 16:26:...		distribution center	100	0	0	No

Figure 18: Trucks overview page with loaded weight changed

If a mistake occurs during the loading process, it can be reversed using the "Unload" button in the same section.

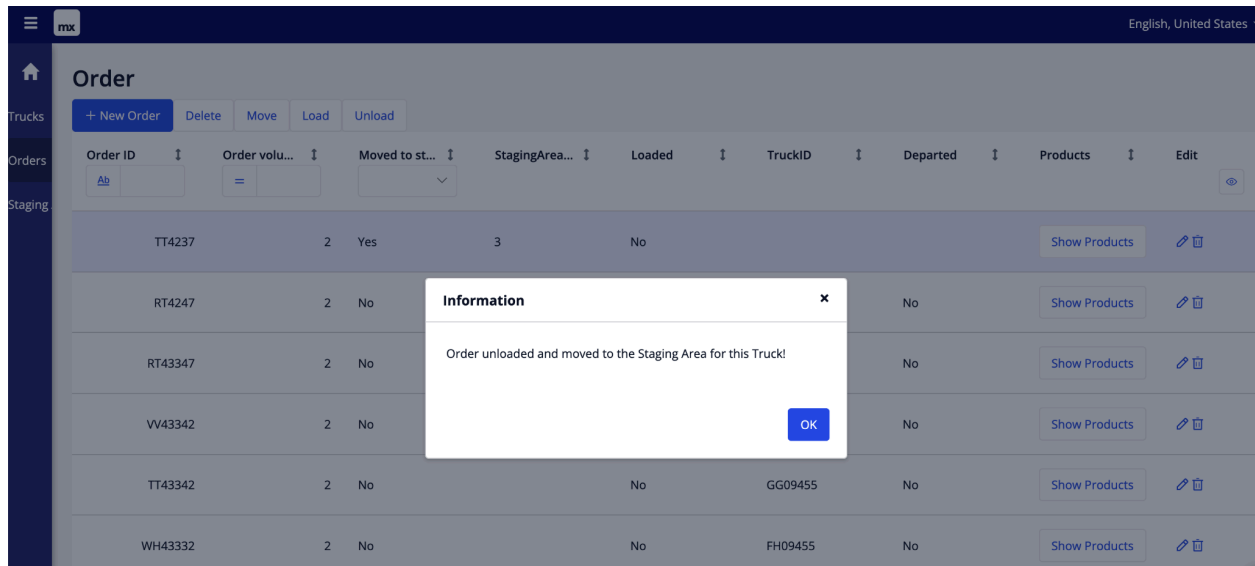


Figure 19: Unload message

Once a truck has been fully loaded, it is ready to depart. This can be done using the "Depart" button in the "Trucks" section. Departing a truck also sends a notification to the Control Tower, confirming that the truck has left the distribution center.

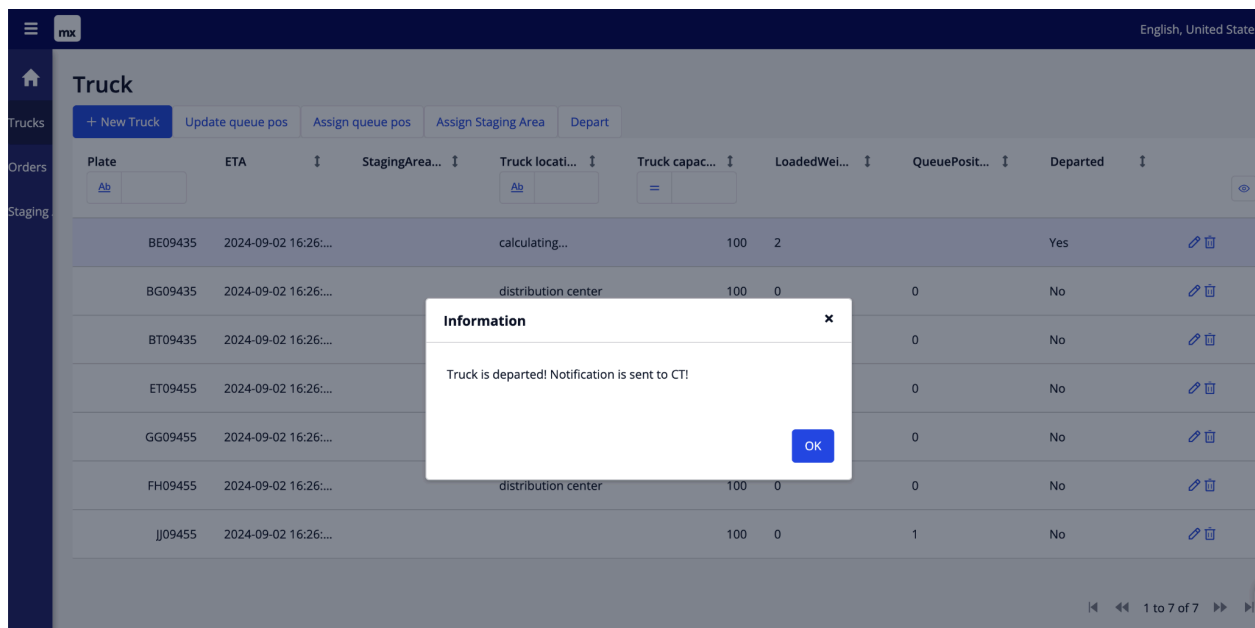


Figure 20: Truck departed message

Data posting testing

We used Postman to test that our application successfully sends a 'truck ready notification' by simulating the Control Tower with a mock server. The Postman mock server was configured to expect a POST request at the endpoint `/rest/notify-truck-ready` and respond with a confirmation

message, "Notification received successfully", when the request is correct. After loading the truck with its corresponding orders and preparing it for departure, clicking the 'Depart' button triggers a microflow in our application. This microflow sends a request to the specified URL with a JSON body in the following format:

```
{
  "OrderID": "AB123",
  "Details": {
    "LoadedWeight": 8900,
    "LoadedPallets": 4
  }
}
```

For testing purposes, the URL used is the Postman mock server:

<https://54d1c8a3-3618-44ec-8d66-c268bcfc3865.mock.pstmn.io/rest/notify-truck-ready>

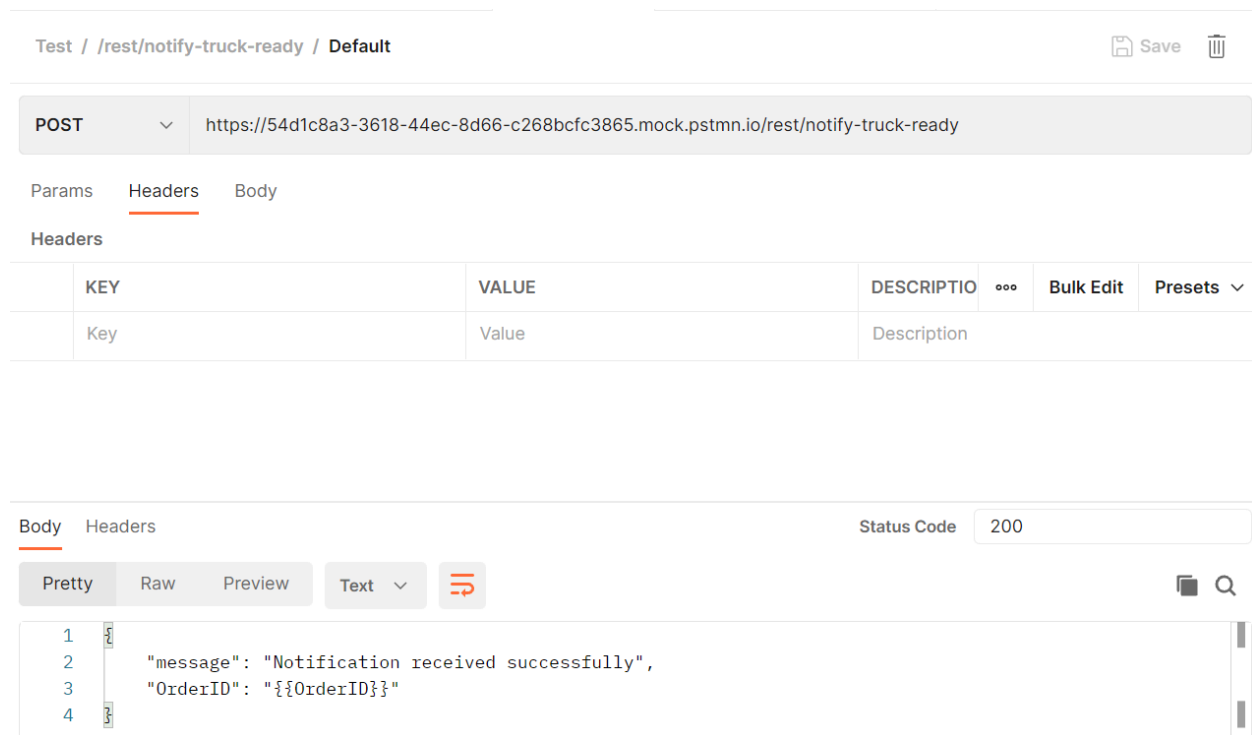


Figure 21: Postman overview

Once a truck is departed, we can verify that the notification was successfully received by checking the recent logs in Postman:

TIME	REQUEST	RESPONSE
Today, 9:22 pm	POST /rest/notify-truck-ready	Default
<div> <div>REQUEST HEADERS</div> <div>REQUEST BODY</div> <div>RESPONSE HEADERS</div> <div>RESPONSE BODY</div> </div>		
<div> <div> <div>▼ {OrderID: "12345", Details: {...}}</div> <div>OrderID: "12345"</div> <div>▼ Details: {...}</div> <div>LoadedWeight: 2</div> <div>LoadedPallets: 1</div> </div> <div> <div>This response does not have headers.</div> <div> <div>► {message: "Notification received successfully", OrderID: "{OrderID}"}</div> </div> </div> </div>		

Figure 22: Postman recent logs

Truck ETA testing

The ETA testing is simple. Firstly the data needs to be generated in a dummy application named CT_dummy_App (note that you need to run both the main application and the dummy application for it to work). In the dummy application you need to click on the “Truck ETA overview” button and create a new dummy instance of a truck and its ETA. This data stays the same up to the point when the truck arrives in 15 minutes. Then, you can click in the dummy app the green button next to the data row and update the ETA to be equal to 15 minutes.

Truck ETA

+ New Truck ETA

Truck license plate

ETA

Data Changes

ABC123

15 Minutes

Update 15-min Timer

Figure 23: Truck ETA overview

All of this data is transferred via OData service to the main app. The data is transferred in a live format, meaning the most up-to-date information is accessible by simply reloading the overview page. The overview page can be accessed by clicking on the “Truck ETAs Overview” button on the main page as shown in Figure ?.

Truck Overview

Order Overview

Staging Overview

Time

Truck ETAs Overview

Figure 24: Truck ETA overview button

If everything is done correctly, the loaded page should show the data from an external database as shown in the picture below.

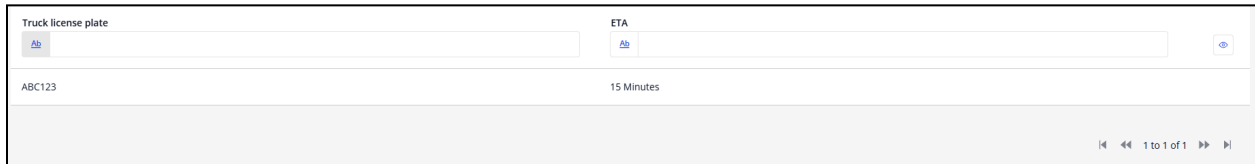


Figure 25: Truck ETAs overview page

Also note that this dummy CT application is only used for the ETA truck notifications, the other functionality related to the communication with CT is implemented using REST and can be directly tested with other groups.

Time API testing

In the navigation page of the application you can click on the “Time” button, you will get this overview on your screen:

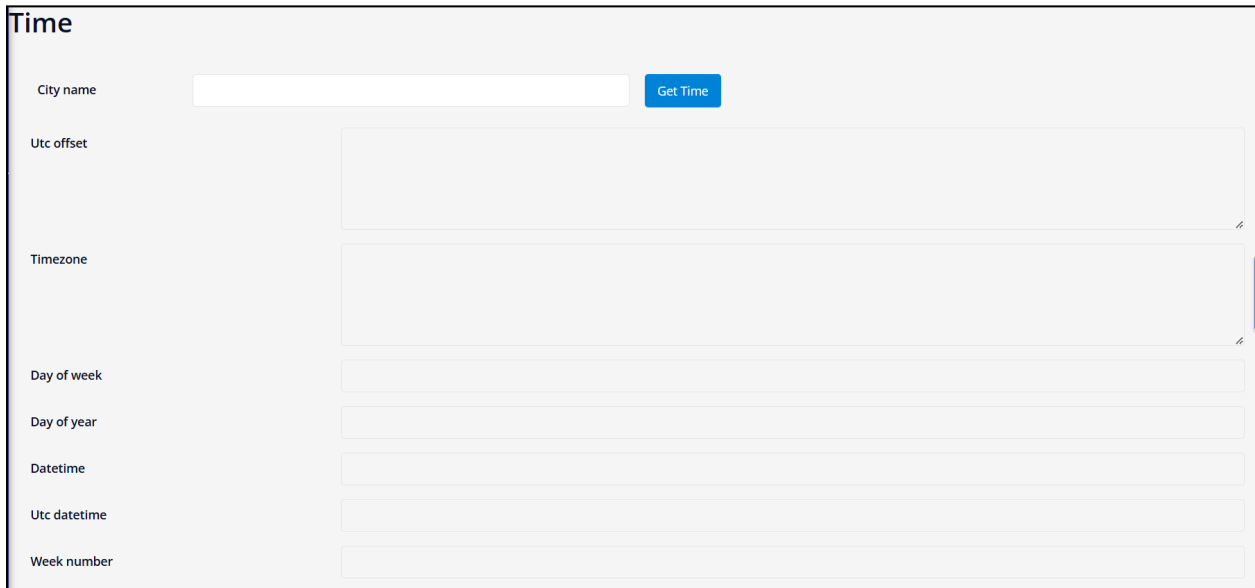


Figure 26: Time overview page

In the input box “City name” you can enter a city name (works only with European capitals) and click on “Get Time”. In the boxes below you will receive detailed Time and Date information regarding the city you have entered (like in the example below).

Time

City name	<input type="text" value="Berlin"/>	<input type="button" value="Get Time"/>
Utc offset	+01:00	
Timezone	Europe/Berlin	
Day of week	0	
Day of year	350	
Datetime	2024-12-15T16:38:35.737003+01:00	
Utc datetime	<input type="text" value="12/15/2024"/>	
Week number	50	
Abbreviation	CET	

Figure 27: Time overview page with a filled City name

Appendix B - Groups connected with

We successfully connected and tested the integration with one group, specifically Control Tower group 9, for REST data retrieval. Although our application is fully prepared to integrate with other groups and services, time constraints limited our ability to test additional functionality with other groups, so the testing was conducted only within the scope of our project.