

Feed an API product roadmap and explore solutions for one opportunity epic

Daniel Martins Figueiredo

Master's Dissertation

Supervisor: Prof. António Carvalho Brito



Mestrado em Engenharia e Gestão Industrial

2022-07-11

Abstract

Product management is an essential activity of any successful product. Traditionally, the concept of a product is used to describe a physical good which is produced with the intent of being commercialized. However, technological advances in the past few decades caused an emergence of software products. As so, it becomes relevant to analyze these new products not only from an IT based perspective but also through a business-oriented lens.

As so, this dissertation aims to explore opportunities to improve an API as a product and present a roadmap of planned implementations. The API in study integrates all the players of a supply chain in a platform (E-Fulfilment) which allows the monitoring and management of every step of the fulfilment process. Since, the technological start-up which developed this product (HUUB) was recently acquired by a global logistics company (Maersk), it is crucial to examine how to adapt the API to the new scale and type of clients.

The identification of opportunities was performed by direct contact with various stakeholders (clients and internal teams affected by the product performance), research of industry practices and analysis of the flows of the processes in place. Afterwards, these initiatives were ranked according to the priority estimated using the RICE scoring model.

The most urgent opportunity identified was the need to provide customers with visibility over the integration issues between systems reported by the API. This is particularly important for brands who are onboarding the platform, since the lack of integration means that their fulfilment operations cannot start. Such a situation is disadvantageous for all parties, since the customers are not transacting their products and Maersk is not receiving operational income from those brands.

To address this problem, a north star solution was designed as a guide to the development of appropriate features. Then, an initial rollout was worked out using Python to automatically retrieve the desired data from both a relational (PostgreSQL) and a non-relational (MongoDB) database and send an email containing the necessary information to correct the issues to the customers.

At the end, not only the solution for the first problem was implemented but also a roadmap mapping out the future API improvement steps was delineated.

Resumo

Gestão de produto é uma atividade essencial de qualquer produto bem sucedido. Tradicionalmente, o conceito de produto é utilizado para descrever um bem físico que é produzido com a intenção de ser comercializado. No entanto, avanços tecnológicos nas últimas décadas levaram a que produtos de software emergissem. Como tal, torna-se relevante analisar estes novos produtos não só de uma perspetiva de tecnologia da informação, mas também através de uma visão orientada a negócio.

Assim, esta dissertação procura explorar oportunidades para melhorar uma API como produto e apresentar um *roadmap* das implementações planeadas. A API em estudo integra todos os intervenientes de uma cadeia de abastecimento numa plataforma que permite a monitorização e gestão de cada passo no processo de abastecimento. Visto que a *start-up* tecnológica que desenvolveu este produto (HUUB) foi recentemente adquirida por uma empresa de logística global (Maersk), é crucial examinar como adaptar a API à nova escala e ao novo tipo de clientes que a usam.

A identificação de oportunidade foi realizada através de contacto direto com variados *stakeholders* (clientes e equipas internas afetadas pela performance do produto), de pesquisa de práticas da indústria e da análise dos fluxos dos processos existentes. Depois, estas iniciativas foram classificadas de acordo com a prioridade estimada com o modelo de pontuação RICE.

A oportunidade identificada mais urgente foi a necessidade de dar visibilidade aos clientes sobre erros de integração entre sistemas reportados pela API. Isto é particularmente importante para marcas que estão na fase de *onboarding* da plataforma, já que a falta de integração significa que as suas operações de abastecimento não podem começar. Esta situação é desvantajosa para todas as partes, visto que os clientes não estão a transacionar os seus produtos e a Maersk não está a receber receita operacional dessas marcas.

Para resolver isto, uma solução *north star* foi desenhada para servir de guia ao desenvolvimento de funcionalidades apropriadas. Seguidamente, um *rollout* inicial foi produzido usando Python para automaticamente recolher os dados desejados, tanto de uma base de dados relacional (PostgreSQL) como de uma não relacional (MongoDB), e enviar um email aos clientes com a informação necessária para corrigir os problemas encontrados.

No final, não só foi implementada a solução para o primeiro problema, mas também foram mapeados os próximos passos de melhoria da API.

Acknowledgments

Firstly, I would like to thank my supervisor António Carvalho Brito for the guidance provided throughout the project.

To Guilherme Pacheco, my mentor at Maersk, for the constant encouragement and support. His expertise and knowledge of the area not only contributed greatly for the success of this dissertation but also for my growth as an engineer.

To everyone at Maersk for welcoming me and giving me all the tools and help required to complete this dissertation, especially Mariana Romba and Diana Monteiro whose help was invaluable.

To FEUP for being more than a home to me in the last five years. I will forever cherish the countless days and nights there.

To all the friends I have made at FEUP. No one else will understand the things we have lived and experienced together. To many more years of random nights and last minute plans.

To Inês for accompanying me in this journey that is life and making everything such a little bit easier.

To my family for the unwavering support. To Miguel for being the best example of a big brother I could ask for and to my parents for not only providing me the means to dream but also for all the sacrifices made to shape me into the person I am today.

"A dream will not become an innovation if there is no realization."

Altaïr Ibn-La'Ahad

Contents

1	Introduction	1
1.1	Company Context	1
1.1.1	HUUB	1
1.1.2	Acquisition by Maersk	2
1.2	Problem Adressed	2
1.3	Project Objectives	3
1.4	Methodology	3
1.5	Thesis Outline	3
2	Literature Review	5
2.1	Introduction to APIs	5
2.1.1	Types of API	5
2.1.2	REST API Industry Practices	8
2.2	API as a Product	9
2.2.1	Customer Onboarding	9
2.2.2	Design	10
2.2.3	Documentation	10
2.2.4	Monitoring	11
2.3	Product Management	11
2.3.1	Product Roadmap	12
2.3.2	Product Discovery	13
2.3.3	Prioritization	15
3	Problem Framework	17
3.1	E-Fulfilment Architecture	17
3.2	Brand API Operating Logic	18
3.2.1	Authentication	18
3.2.2	Generic Asynchronous Calls	18
3.3	Platform Process Flows AS-IS	19
3.3.1	Products Creation	19
3.3.2	Purchase Orders Creation and Reception at Warehouse	20
3.3.3	Sales Orders Delivery Creation and Shipment Confirmation	21
3.3.4	Returns Received	22
3.3.5	Stocks Update	22
3.3.6	Exception Flows	23
3.4	Current Objectives and Opportunities	24

4	Identification of Improvement Opportunities	27
4.1	API as a Product Opportunities	27
4.1.1	Product Data Structure	27
4.1.2	Reception Creation	28
4.1.3	Handling of Exception Flows	28
4.1.4	Relation between POs and Containers	29
4.1.5	Character Translation	29
4.1.6	Barcode Options	29
4.1.7	Error Communication	30
4.1.8	Integration Errors Visibility	30
4.2	Prioritization of Opportunities	31
5	Solution Design	33
5.1	Initiative Background	33
5.2	North Star Design	35
5.2.1	Design Requirements	35
5.2.2	Solution Mock-up	36
5.2.3	Dependencies	37
5.3	MVP Design	38
5.3.1	Methodology	38
5.3.2	Solution details	39
6	Results	41
6.1	API Product Roadmap	41
6.2	Evaluate Impact of Solution	42
6.2.1	Support Resources	42
6.2.2	Number of Onboarding Customers	43
6.2.3	Assumptions Analysis	44
7	Conclusion	47
7.1	Main Outcomes	47
7.2	Next Steps	48
A	E-Fulfilment Objects	55
A.1	Products	55
A.2	Purchase Orders	57
A.3	Sales Order Deliveries	58
B	North Star Solution Pages	61
C	Solution Algorithm Flowchart	65

Acronyms and Symbols

API	Application Programming Interface
DX	Developer Experience
ECL	E-commerce Logistics
IT	Information Technology
MVP	Minimum Viable Product
OST	Opportunity Solution Tree
PO	Purchase Order
REST	Representational State Transfer
SO	Sales Order
SOAP	Simple Object Access Protocol
URI	Uniform Resource Identifier
UI	User Interface
UX	User Experience

List of Figures

2.1	Example of a format of a Product Roadmap	13
2.2	Example of Opportunity Solution Tree. Adapted from: Torres (2021).	14
3.1	E-Fulfilment high-level architecture	17
3.2	Operating logic of API asynchronous call	19
3.3	Process Flow of Product Creation	19
3.4	Process Flow of PO Creation and Reception at Warehouse	20
3.5	Process Flow of Delivery Creation and Shipment	21
3.6	Process Flow of Returns Received	22
3.7	Process Flow of Stocks Update	22
3.8	OST representing Maersk’s current objectives and opportunities	24
5.1	Percentage of integration requests with errors in each month	34
5.2	Mock-up of page displaying product integration errors	37
5.3	Section of email sent to Maersk’s support team representing errors in product integration.	40
6.1	API Product Roadmap Produced	42
B.1	Mock-up of page displaying initial dashboard and menu notifications	61
B.2	Mock-up of side panel in product integration errors page	62
B.3	Mock-up of page displaying purchase order integration errors	62
B.4	Mock-up of page displaying sales order integration errors	63
C.1	Flowchart describing the solution algorithm	65

List of Tables

2.1	Comparison between REST and SOAP (Khan and Abbasi, 2015; Patni, 2017; Soni and Ranga, 2019)	6
2.2	HTTP methods (Massé, 2011; Rodríguez et al., 2016)	8
2.3	HTTP response code categories. Adapted from Massé (2011).	8
2.4	Metrics monitored in an API (Malvik, 2022; Medjaoui et al., 2021)	11
4.1	Estimation of RICE factors for each opportunity	31
4.2	Priority ranking of opportunities	32
5.1	NU-IN's historical request data	34
6.1	Summary of current resource allocation	43
6.2	Quarterly income generated by extra onboardings in the same year	44
6.3	Additional income obtained with the implementation depending on estimated extra capacity	45
A.1	Product Model Fields in E-Fulfilment	55
A.2	Product Variant Fields in E-Fulfilment	56
A.3	Purchase Order Fields in E-Fulfilment	57
A.4	Supplier Fields in E-Fulfilment	57
A.5	PO Items Fields in E-Fulfilment	58
A.6	Sales Order Delivery Fields in E-Fulfilment	58
A.7	Customer Fields in E-Fulfilment	59
A.8	Shipping Address Fields in E-Fulfilment	59
A.9	SO Delivery Items Fields in E-Fulfilment	59

Chapter 1

Introduction

The profitability of any business is dependent on the success of the products or services it offers. However, companies face increasingly demanding challenges that can make turning a product lucrative an arduous task. Some of those obstacles are the rapidly changing technology landscape and the gradually more intense and global competition (Fernandes, 2021). Because of this, product management is becoming an even more relevant tool for organizations to be successful (Fradin, 2019).

This dissertation focuses on the management as a product of an API designed to integrate the internal systems of a brand with its logistics operations. The project was developed in a technological start-up team (HUUB) recently acquired by an international logistics company (Maersk).

1.1 Company Context

1.1.1 HUUB

HUUB was a technological start-up founded in 2015 that offered logistic services and supply chain management to brands in the fashion industry. The main objective of the company was to free brands from the task of managing their own supply chain and focus solely on growing their product. For this, HUUB positioned itself in the center of a dynamic ecosystem composed by all the stakeholders of the logistic operation: the brand, suppliers, fulfilment partners, carriers and the final customer. All of them were connected by the company's digital platform SPOKE, where they could easily check and monitor the status of the supply chain. This digital solution seeks to tackle existing problems in traditional supply chains such as high complexity and lack of process visibility and collaboration between the different members of the operation.

To ensure synchronization of the different information flows of the supply chain, both the brand's and the suppliers' internal systems were integrated with SPOKE through APIs. Due to HUUB mainly working with Small and Medium Enterprises (SMEs), most brands used e-commerce platforms (e.g., Shopify, Prestashop). However, the integration of SPOKE with ERP systems was also available.

1.1.2 Acquisition by Maersk

Maersk is an integrated transport and logistics company with over 100 years of history. The group offers logistic solutions for every step of the supply chain, fulfilling clients' needs from end-to-end. With presence in 130 countries, it currently possesses the second largest container logistics operation in terms of cargo capacity worldwide (Alphaliner, 2022).

Due to the rapid increase in online consumption in recent years, Maersk clients increasingly feel the need to adapt their business model and adopt digital strategies that can support their growing e-commerce sales. Evidently, in order to successfully implement these strategies it is imperative that their supply chains keep up with this digital transformation. Addressing this necessity, in 2021 Maersk decided to invest in E-commerce Logistics (ECL) by acquiring companies that were dedicated to that activity, namely Visible SCM which was an industry leader in the US and B2C Europe which already had a significant reach in Europe.

More than expertise in ECL, Maersk required technological expertise that could assist in developing its omnichannel offering. Being investors in HUUB since 2019, the group was already familiar with the start-up's value proposition and how it could contribute to their objectives. As so, HUUB followed the path of Visible SCM and B2C Europe and, after being acquired in September 2021, integrated Maersk's ECL team. SPOKE, the platform offered by HUUB, was rebranded as E-fulfilment.

This investment's end goal is to provide B2C Fulfilment and B2C Delivery centered on a robust and flexible e-commerce technology basis in the main e-commerce regions: Europe, North America and Asia.

1.2 Problem Addressed

As a consequence of the acquisition, the product offered by HUUB needs to be adapted to the Maersk reality. Other than variations in nomenclature and standards used, it is also important to address the transition concerning the product's use cases. For this, it is relevant to take into account the gap between the clients that used the application before and those who will start using it. This is valid regarding the type of products sold, the origin, and the size of the clients' operation. While HUUB only operated in the fashion industry with relatively small brands, Maersk works with global brands that sell every kind of product.

On account of these differences between use cases, it becomes pertinent to analyze the current state of the product and understand which improvements can be implemented to add value to the customer experience.

A relevant aspect to highlight is that at the start of the project there was not any brand using E-Fulfilment to manage their e-commerce logistics yet. However, some customers were already negotiating with Maersk the conditions to start using it, including new clients from China wishing to enter the European market.

1.3 Project Objectives

This project entails two main objectives. The first one consists of exploring the company's public API as a product and identifying opportunities to improve it in a way that adds value to the customer. The identified opportunities should then be prioritized and incorporated into the already existing product roadmap of the API.

The second objective involves delving into one of the opportunities previously found and developing a feasible working solution for it. After its implementation, an evaluation of the solution's impact is also expected to be carried out.

1.4 Methodology

The first step of the project was to research information in the literature regarding the basic concepts that would be addressed. This included definitions of APIs and how this technology could be approached as a product and fundamentals about product management, more specifically concerning product roadmaps.

After this, a careful review of the internal processes of the company was conducted in order to gain insights on the relevance of the project inside the organization. Furthermore, this review was useful to compare the reality of the practices and technologies used by the company with the theoretical research done previously.

Next, in order to assess potential improvement opportunities, it was established contact with various stakeholders. This included communication with the teams within the company whose work was affected by problems or limitations of the API and meetings with clients to determine their needs in each phase of the customer cycle. Simultaneously, an analysis of the procedures adopted by competitors was performed so as to understand what the best practices of the industry are and whether there was a chance to implement them.

Afterwards, the company's relational and non-relational databases were explored in order to gather information that could aid in the process of developing possible solutions. Moreover, historical data regarding the application's performance was retrieved to establish a benchmark by which the final solution could be evaluated.

1.5 Thesis Outline

This dissertation is structured in seven chapters. The objective of the current chapter is to provide context and illustrate the motivation behind the work executed.

The second chapter offers insights on the state-of-the-art and theoretical background of the concepts addressed throughout the project.

Further details of the problem framework are presented in chapter 3 along with a description of the AS-IS situation.

The fourth chapter displays the improvement opportunities that were explored, explains the methodology behind their discovery and states who would be affected by their implementation and to what extent. After this individualization, the opportunities are ranked by priority.

In the fifth chapter one of the opportunities is explored in greater detail and a solution for the initiative is provided. Along with a description of the features and their usability, a mock-up design is also presented.

In chapter 6 the results of the project are exhibited, namely the updated product roadmap and the evaluation of the solution developed.

Finally, the seventh chapter presents the conclusions from this dissertation and reflects on the future work to be done.

Chapter 2

Literature Review

This chapter provides a theoretical framework of the most relevant concepts addressed in this dissertation as well as the state of the art of the respective areas of study. Firstly, an introduction to APIs, their types and their current practices and use-cases is presented. After that, the API-as-a-Product approach is discussed and different product management methodologies are explored with a higher focus on product roadmaps.

2.1 Introduction to APIs

An API is an initialism for Application Programming Interface. The literature regarding this term is extensive and varied which means that there are several different definitions that explain what this concept is. According to Shnier (1996) an API can be defined as "the calls, subroutines, or software interrupts that comprise a documented interface so that an application program can use the services and functions of another application, operating system, network operating system, driver, or other lower-level software program".

Alternatively, Medjaoui et al. (2021) suggest that the term API may be used to refer to three distinct elements: an interface, an implementation or an instance. Beginning with the first two, while the interface is the component that exposes the functionalities of the program a user intends to access, the implementation is no less than those same functionalities, i.e., the actions the user intends to do. This difference is also reflected in the way each of two elements is expressed. The interface commonly uses shared protocols, whereas the implementation relies on programming languages. The final element is described as a combination of the previous two. This means that an instance contains both the interface and the implementation.

2.1.1 Types of API

There is not a total uniform standard regarding how APIs are built and how they operate. As so, it is possible to categorize them into different types depending on their architectural style or protocols they use. Since, REST and SOAP are currently the most predominant types of API (Soni

and Ranga, 2019), a comparison of their characteristics is provided in Table 2.1. After that, both types are explored individually and their core principles are presented.

Table 2.1: Comparison between REST and SOAP (Khan and Abbasi, 2015; Patni, 2017; Soni and Ranga, 2019)

Characteristic	REST	SOAP
Implementation	Easier	Harder
Performance	Faster	Slower
Data Formats	Multiple (JSON, XML, etc.)	Only XML
Communication Protocol	Only HTTP	Any
Complexity	Low	High
Coupling	Loose	Tight
Payload Weight	Light	Heavy
Bandwidth Usage	Low	High
Security and Authorization	Harder to Enforce	Included in the protocol

REST - REpresentational State Transfer

REST, which is short for Representational State Transfer, is a web architectural style first described by Fielding (2000). The motivation for its emergence was a looming scalability crisis of the Web in the 1990s (Massé, 2011). As so, Fielding (2000) identified several constraints that would allow the Web to continue expanding and grouped them into six categories:

1. Client-Server

Clients and servers are separate components with different contributions. This means that as long as they comply to the Web's uniform interface, they can be implemented and deployed independently.

2. Uniform Interface

The interfaces of the web components need to be uniform to assure successful interactions between each other. This category encapsulates four constraints:

- Identification of resources: every web-based concept is designated a resource and can be uniquely identified by a Uniform Resource Identifier (URI);
- Representation of resources: clients do not interact directly with a resource, manipulating representations of it instead. This allows the resource to be represented in different formats (e.g., XML, JSON, HTML) without needing a different URI;
- Self-descriptive messages: a client's request message can include the state desired for a resource and, similarly, a server's response may incorporate the current state of the resource;

- Hypermedia as the engine of the application state (HATEOAS): resources representations contain links to other related resources, allowing clients to navigate between resources directly.

3. Layered System

The constraints of this category allow that proxies, gateways and other intermediaries with functions such as enforcing security be deployed between a client and server.

4. Cache

These constraints let network intermediaries store resource representations in an effort to increase the availability of an application and reduce network latency.

5. Stateless

A server does not memorize the state of a client's application. This means that interaction states between the server and the client do not exist and, consequently, a client must include all the necessary information in each request it makes.

6. Code-on-demand

Servers can temporarily transfer scripts or plug-ins to clients, which the latter can execute.

If an API complies to the constraints contained in these categories, it is considered RESTful, i.e., a REST API (Massé, 2011). However, it is important to mention that these REST principles are not defined rules but guidelines that help build scalable and robust APIs. As a consequence, there is no standardization of REST APIs, leading to different interpretations and implementations of this technology (Rodríguez et al., 2016).

SOAP - Simple Object Access Protocol

SOAP (Simple Object Access Protocol) is a messaging protocol that relies on XML formatting to communicate over a network (Soni and Ranga, 2019). Contrarily to REST, which accesses resources, SOAP retrieves operations (Khan and Abbasi, 2015). The communication is established through SOAP messages which can consist of four elements (Soni and Ranga, 2019):

- Envelope: defines the limits of the message, i.e., the beginning and the end. It is a mandatory component of the SOAP message;
- Header: contains optional attributes such as authorization data. Its inclusion is optional;
- Body: mandatory element that holds the XML data that represents the information being communicated;
- Fault: optional element that presents information regarding errors that occurred during the processing of the message.

Other characteristics of this message protocol include being platform independent and, in contrast to REST, being able to use other protocols other than HTTP (Khan and Abbasi, 2015).

2.1.2 REST API Industry Practices

In spite of the wide variety of REST implementations, there are commonly recognized good practices for its development. In fact, Massé (2011) provides design rules for REST APIs that, even if unofficially, have become standard in the industry. The literature on this subject is extensive and very detailed in technical aspects on how to properly model, identify, represent and manage resources (Berners-Lee et al., 2005; Massé, 2011; Palma et al., 2015; Pautasso, 2014; Richardson and Ruby, 2008; Rodríguez et al., 2016). Due to that vastness of information, it would not be feasible to discuss in detail the different practices in this dissertation. As so, this section will only address the state-of-the-art of HTTP interaction, namely the design of clients requests and servers responses.

Regarding requests, REST APIs should rely on the available HTTP methods to conduct operations relating to resources (Massé, 2011; Rodríguez et al., 2016). The usage of such methods should comply with the standard HTTP semantics, meaning that it is not considered good practice to tunnel requests, i.e., a method should not be called to perform the task of a different method (Rodríguez et al., 2016). Table 2.2 summarizes the main HTTP methods and their functions.

When it comes to server responses, it is essential to have rules that allow the client to understand what is meaning of the response it obtained and how it should approach it (Richardson and Ruby, 2008). As so, a well developed REST API should follow the predefined HTTP response codes, using the standardized meanings in the appropriate situations (Massé, 2011). Table 2.3 depicts the five different categories of responses as well as the respective code and meaning.

Table 2.2: HTTP methods (Massé, 2011; Rodríguez et al., 2016)

Method	Function
POST	Create new resources
GET	Retrieve a representation of a resource
PUT	Update or create resources
DELETE	Remove a resource from its parent
OPTIONS	Retrieve available interactions of a resource
HEAD	Retrieve response headers

Table 2.3: HTTP response code categories. Adapted from Massé (2011).

Codes	Category	Meaning
1xx	Informational	Server has received request and is continuing the process
2xx	Success	Client request was successful
3xx	Redirection	Additional actions are required to complete client request
4xx	Client Error	Request failed due to client error (e.g., invalid request syntax)
5xx	Server Error	Request failed due to server error (e.g., server is down)

2.2 API as a Product

An API-as-a-Product approach highlights the importance of managing an API like a traditional product is managed. This means ensuring proper handling of each stage of the product lifecycle from the design phase to implementation, release, and long-term monitoring and maintenance (Medjaoui et al., 2021). This contrasts with unplanned implementations of APIs that focus only on delivering quick digital solutions, acting on an ad-hoc fashion that leads to system redundancies and poor maintenance practices that cancel out the potential value created (Iyengar et al., 2018). Overall, this approach involves adopting a customer-oriented strategy over an IT only perspective (Malvik, 2022).

Similar to any kind of traditional product, an API-as-a-product is dependent on a number of decisions that determine its success. To make the correct decisions, it is essential to understand who the customers of the API are, what functionalities they need and to gather continuous feedback from the users of the product (Medjaoui et al., 2021).

The success of a product can also be measured by its adoption rate and the general satisfaction of the customers who use it. This satisfaction can be assessed through the user experience (UX) which encapsulates all interactions that a customer has with a product (Medjaoui et al., 2021). In the case of APIs, since the users are developers, UX is also known as developer experience (DX) (Medjaoui et al., 2021).

A multitude of factors that can help provide a good DX are discussed by Malvik (2022) and Medjaoui et al. (2021). However, due to being particularly more relevant, only four aspects are explored in this section: customer onboarding, design, documentation, and monitoring.

2.2.1 Customer Onboarding

Before beginning to fully utilize an API, it is required that the user is given access to the product functionalities as well as to the information on how to operate with the product. This first contact with the API product is the customer onboarding stage (Malvik, 2022). It is of the utmost importance that companies invest on making this transition phase as smooth as possible for the customers, since an API that is difficult for users to learn and quickly realize the value it offers leads to increased churn rates, i.e., the amount of customers that will not adopt the product after first contacting with it will increase (Medjaoui et al., 2021).

A well designed onboarding experience allows users to swiftly start working with the product and obtain value from it (Malvik, 2022). While designing the customer onboarding, the main objective is to reduce to a minimum the time it takes from the instant a client first contacts with the API to the moment they comprehend in what ways that product could be of value to them (Medjaoui et al., 2021). Methods to achieve this goal include good process documentation, live demonstrations of the provided functionalities and constant analysis of the entire process to discover possible improvement opportunities (Malvik, 2022; Medjaoui et al., 2021).

2.2.2 Design

The design of an API corresponds to all the decisions taken regarding the product's look, feel and utility (Medjaoui et al., 2021). However, an API may be evaluated on many different attributes that cannot be optimized simultaneously, meaning that there are trade-offs between investments in different qualities (Myers and Stylos, 2016). In order to know which attributes are essential for a specific API product, a strategy that addresses the aim of the product and the users it targets should first be established (Medjaoui et al., 2021).

Myers and Stylos (2016) identified eleven different quality attributes for an API and grouped them into two high level attributes:

- Usability
 - Learnability: how easy it is to learn how to use the API;
 - Productivity: how productive are the programmers who use the API;
 - Error-Prevention capability: how well are errors prevented by the API;
 - Simplicity: how simple is the API;
 - Consistency: how consistent is the design of the API;
 - Mental Models Match: how much the API corresponds to the mental model imagined by the users.
- Power
 - Expressiveness: how much abstraction the API is able to provide;
 - Extensibility: how much the API can be extended to create user-specific components;
 - Evolvability: how easy it is for designers to update and create new versions of the API;
 - Performance: how well the API operates in terms of speed and resources consumption;
 - Robustness: how safe and robust is the API implementation.

2.2.3 Documentation

Documentation is immensely important to the success of an API-as-a-Product as it contributes greatly to the developer experience (Malvik, 2022). Effective documentation should help the users answer any question that may arise during the utilization of the API (Medjaoui et al., 2021). This help may be in the form of a complete learning experience guiding through the use of the application for first time users or simply a glossary of available functionalities and possible events or exceptions that may occur and how to correctly deal with them (Medjaoui et al., 2021).

Although, documentation adds value to an API, since many changes are expected to occur during the product lifecycle it can become costly to constantly rewrite different instructions. As so, it is not only relevant to understand how much documentation is enough to give to your users but

also at what stage it should be prepared (Medjaoui et al., 2021). Nevertheless all produced documentation should be consistent both in functionality and appearance Malvik (2022). This means whenever an update occurs all previous information should be reviewed to ensure synchronization with the changes applied.

2.2.4 Monitoring

Monitoring is a crucial aspect of API management as it allows to obtain real-time data regarding the performance and usage of the application (Malvik, 2022; Medjaoui et al., 2021). Different types of data can be monitored depending on the purpose the information is intended for. Table 2.4 summarizes the different groups of metrics and their utility to the API product.

Table 2.4: Metrics monitored in an API (Malvik, 2022; Medjaoui et al., 2021)

Data	Possible Metrics	Utility
Problems	Errors and Warnings Metrics	Helpful in the detection and handling of problems. Early detection can mitigate the negative impact of an issue.
System Health	CPU and Memory Health	
API Health	API Uptime and State	
Message Logs	Request and Response Message Bodies	Useful for troubleshooting operations, i.e., understanding what may be causing an issue.
Usage data	Resource Usage and User Request Metrics	Provides information regarding how users are operating with the API that can be used to improve the UX.

Given the benefits depicted, the ideal scenario would be to monitor closely every aspect of the API product in order to maximize error prevention and improvement opportunities (Medjaoui et al., 2021). However, this process entails costs of data production, harvesting, storage, and analysis that may outweigh the value it creates (Medjaoui et al., 2021). As so, it is imperative to decide which aspects should be monitored in order to generate the maximum value in a sustainable way that complies with resources constraints (Medjaoui et al., 2021).

2.3 Product Management

The concept of product management is not new and has been widely discussed both in practical contexts and in scientific literature. According to Pohl et al. (2005), traditional product management can be defined as the "planning, organising, executing, and controlling of all tasks, which aim at a successful conception, production, and marketing of the products offered by a company". This point of view is shared by Ebert and Brinkkemper (2014) who emphasize the end-to-end scope of this subject, i.e., since the inception of the product until its deployment and end of life cycle.

Product managers are responsible for ensuring the execution of all the tasks along the product's life cycle in a way that fulfils customer needs and at the same time greatly contributes to the company's long-term value (Gorchels, 2006). For this, it is essential to balance both extensive knowledge of the company and its processes and intensive knowledge of the product itself (Gorchels, 2006).

To successfully implement a product management methodology three concepts are of vital importance: product roadmaps, product discovery, and prioritization. For the purpose of better understanding the details behind this subject, all three of them will now be analyzed individually.

2.3.1 Product Roadmap

Product roadmaps are one of the main tools of product management. Their main objective is to establish a vision and general direction for the product offering by describing how the product will help the company achieve its business goals and the features that should be implemented for that to happen (Münch et al., 2019b). However, more than value to the company, a product roadmap should also strive to provide value to the customers (Lombardo et al., 2017).

Regarding the industry practices, Münch et al. (2019a) identified nine dimensions as being relevant in assessing a product roadmap's maturity, i.e., factors which contribute to the effectiveness of this tool.

1. **Items granularity:** the items depicted should be of different granularity. This means that it should not only be stated what will be built but also the reasoning behind it. In practical terms the product vision, high-level customer and system needs, customer and business-oriented goals and short-term features should all be included.
2. **Detail level:** the level of detail portrayed for each item should be dependent on the timeline. Short-term features should be described in detail while long-term planning should only be high-level. This means that the decisions regarding medium and long-term planning are delayed, promoting flexibility, avoiding waste of resources and reducing unnecessary effort.
3. **Reliability:** the roadmap should be under systematic change management, i.e., it should be regularly reviewed and any adjustment should be appropriately justified. This allows the company to continuously adapt in a proactive way instead of only reacting to its environment.
4. **Confidence:** the effect of the items of the roadmap should be determined and validated. In other words, there is high confidence that the items presented will contribute to a proposed goal.
5. **Discovery capability:** a process for identifying and validating items to assure that the features implemented are useful and wanted is in place.
6. **Responsibility assignment:** the responsibility of placing items in the roadmap should be clearly defined.

7. **Prioritization:** the items portrayed should be prioritized in order to maximize the efficiency and effectiveness of value delivery to the company and customers.
8. **Alignment:** the roadmap should be aligned for all the stakeholders. This implies the existence of a common roadmap from which individual representations are mapped for each stakeholder depending on the information they require.
9. **Ownership assignment:** the ownership the roadmap should be clearly defined.

Concerning the presentation format, Munch et al. (2020a) addresses many possible options comprising different elements and complexity. Taking into account the purpose of a product roadmap and the industry practices discussed in this subsection, Figure 2.1 illustrates a model format for this tool.

Product Vision: (...)			
Quarter 1	Quarter 2	Quarter 3	Quarter 4
Theme 1	Theme 2	Theme 3	Theme 4
<u>Objective 1</u> - Feature 1 - Feature 2	<u>Objective 1</u>		
	<u>Objective 2</u>	Theme 1	
<u>Objective 2</u> - Feature 1 - Feature 2 - Feature 3 - Feature 4	Theme 1 <u>Objective 3</u>		
	<u>Objective 4</u>	Theme 2	

Figure 2.1: Example of a format of a Product Roadmap

In the specific case of software development there are a few important concepts to address which relate directly to the items that should be included in product roadmapping. Those elements follow a hierarchical structure where the accomplishment of smaller and more descriptive elements leads to progress in achieving the high-level items (Leffingwell, 2011):

- Themes are the key strategic objectives for the product;
- Epics are high-level requirements representing customer needs that fulfill the potential value of a theme;
- Features are small and specific deliveries of value which contribute to the realization of an epic;
- User Stories are every task executed in order to deliver a feature.

2.3.2 Product Discovery

Product discovery is a fundamental technique for the elaboration of a successful product roadmap. It aims to determine which features should be implemented to fulfill customer needs by identifying

problems that the clients may face and developing feasible solutions to deal with them (Munch et al., 2020b). Other than identifying the problems that customers are willing to pay to be solved, Joh and Mayfield (2009) also recommend further dimensions that should be explored to carry out effective product discovery: understanding what features can be deploy given the company's capabilities and analyzing what the organization can offer better than the competition.

An effective tool to structure product discovery is the Opportunity Solution Tree (OST), since it helps visualize both the information obtained and the decisions made along the process of discovery (Torres, 2021). There are several additional advantages associated with the use of OSTs. As stated by Torres (2021), it fosters abstraction from individual ideas, avoiding fixation on single solutions for open-ended problems. As a consequence, it leads to product managers questioning what is the outcome they are trying to achieve and encourages them to explore different ways to reach their goal. An important aspect to take into account while doing this abstraction is that the initiatives being compared with each other must contribute to the same outcome and every solution conceptualized needs to be connected to an opportunity (Torres, 2021).

In Figure 2.2 a general model for the structure used to build an OST is presented. Depicted are three different levels with different characteristics (Torres, 2021):

- Outcomes represent a business necessity which indicates in what way the product can create value;
- Opportunities represent the customer necessities and difficulties that can contribute to the stated outcome if they are explored;
- Solutions represent features that address the situations brought up by the opportunity to which they are connected.

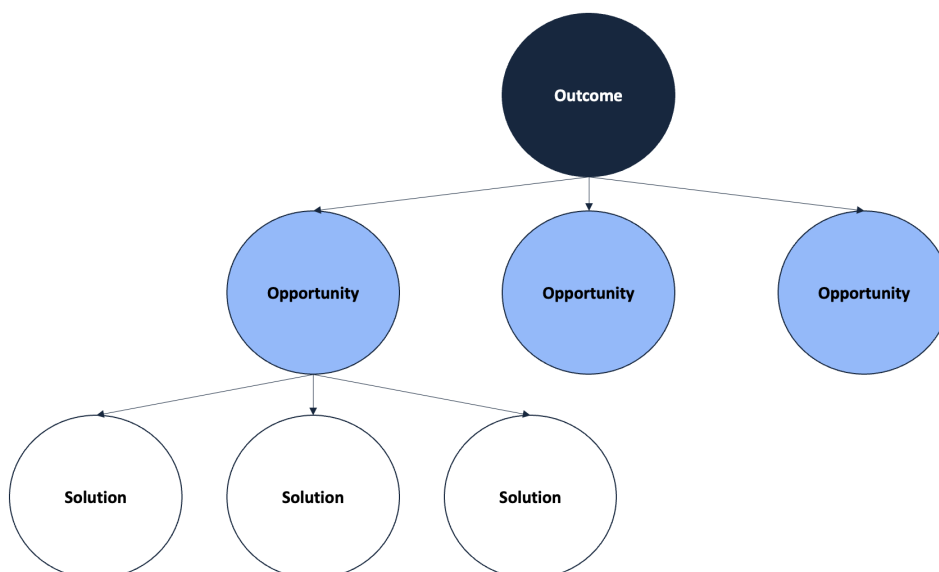


Figure 2.2: Example of Opportunity Solution Tree. Adapted from: Torres (2021).

It is important to highlight that the opportunity space in the OST is not restricted to a single level, meaning that an opportunity can be split into smaller and more specific opportunities (Torres, 2021).

2.3.3 Prioritization

Prioritization consists of selecting the order in which the features idealized in the roadmap will be deployed (Berander and Andrews, 2005; McBride, 2018). Ideally, every feature would be designed and implemented. However, in reality there are cost and time constraints as well as limited resources, meaning that it is essential to understand which requirements are more urgent (Berander and Andrews, 2005; Trieflinger et al., 2021). After acquiring that knowledge the most important initiatives are added to the release plan in the product roadmap, while the remaining are stored in a product backlog which contains all desired items that have already been prioritized but not yet implemented (Leffingwell, 2011).

Even though there is the possibility to correct poor prioritization decisions in later development stages through change management, it is considerably more expensive to do so. Due to this, an early selection of the correct order in which to implement features leads to the maximization of the company's cost efficiency (Berander and Andrews, 2005).

Regarding the process itself, there are many factors that can be taken into account when prioritizing, such as the importance the item brings to the system, the penalty associated for not fulfilling a requirement, the estimated cost and time, or even the risk associated with it (Berander and Andrews, 2005). As a consequence, many different techniques may be employed depending on the overall objective of the company.

One common prioritization method is the RICE scoring model. This tool combines four evaluation factors to assign a priority score that represents the expected impact of a feature given the time spent developing it (McBride, 2018). The name of the technique is an acronym of the first letter of each of the factors considered (McBride, 2018):

- *Reach* measures the number of people affected by the item;
- *Impact* measures how each individual is impacted by the item;
- *Confidence* measures the confidence in the estimations of the other factors;
- *Effort* measures the amount of resources needed to complete the item.

McBride (2018) also offers methods to estimate each of those factors. Regarding *Reach*, the value estimated takes into account the number of people in the considered time period affected by the item. For *Impact*, a scale ranking the qualitative impact of an implementation can be adopted: 3 for massive, 2 for high, 1 for medium, 0.5 for low, and 0.25 for minimal. *Confidence* follows a similar approach: 100% for high confidence, 80% for medium, and 50% for low. Finally, *Effort* can be estimated as the amount of people and time required to complete the work. After having

an estimation for all of the factors it is possible to calculate the RICE score. The higher the score, the more of a priority an item is (McBride, 2018).

$$RICE\ Score = \frac{Reach \times Impact \times Confidence}{Effort} \quad (2.1)$$

Chapter 3

Problem Framework

This chapter aims to provide context and present the problem to be tackled. In Section 3.1 the architecture of the application offered by Maersk (E-Fulfilment) is described. After that, the operating logic of the API that allows brands to integrate with E-Fulfilment (Brand API) is illustrated. Section 3.3 lays out the details of the different processes that occur between the application and the customers' systems. Finally, Section 3.4 addresses the company's current objectives and opportunities.

3.1 E-Fulfilment Architecture

In order to understand the existing flows in E-Fulfilment's ecosystem, it is relevant to first explore how the platform is structured and how the different information systems communicate with each other. A visual representation of the architecture of that ecosystem is provided in Figure 3.1.



Figure 3.1: E-Fulfilment high-level architecture

Brands can integrate with E-Fulfilment using either a general e-commerce platform as Shopify or a full-fledged information system such as an ERP. In any alternative, there is a layer between the brand's system and the Brand API to translate processes between systems. Since most platforms share similar processes, if the integration is done via e-commerce platform the translation is made by an Universal Vendor Layer owned by Maersk. When it comes to ERP systems, the translation layer is known as a middleware. This middleware can be owned by the brand or Maersk, depending on who was responsible for its development.

At the other end, the company's warehouse partners connect to E-Fulfilment through the 3PL (Third Party Logistics) API. Just like in the brand side, it is required that the partners system and E-Fulfilment are properly adjusted to each other. This can be achieved by building a connector

similar that assimilates the two systems. Alternatively, a partner can accept to adapt their information system to conform to E-Fulfilment, meaning that there is no need for a connector and the system can be directly connected to the 3PL API.

Given the fact that the project developed focuses on understanding the value that the Brand API can add to the customer, that component will be explored in further detail in the following sections. Moreover, since the majority of the customers that will start using E-Fulfilment are expected to integrate with an ERP, the work developed will be centered on that use case.

3.2 Brand API Operating Logic

Maersk's Brand API adheres to the REST principles presented in Section 2.1.1. The communication between the customer's ERP and the Brand API is done via HTTP requests and responses. In order to fully understand how that communication is established, it is necessary to describe how the client authentication is granted and generic asynchronous calls are executed.

3.2.1 Authentication

Due to security reasons, whenever a customer wishes to execute calls to the Brand API a valid JSON Web Token (JWT) needs to be included in the request header as an authentication factor. As so, the brand's first request must be to obtain this authentication token and it must include the predefined email for login and the matching password. If the request is valid, the API returns a response message containing the standard HTTP code for success (200) along with details regarding the request and the JWT that should be included in the subsequent data requests. The expiration time of the JWT is agreed between the brand and Maersk in advance depending on the level of security required.

3.2.2 Generic Asynchronous Calls

Other than the authentication, most of the calls to the API are not processed synchronously. Instead, they follow the sequence depicted in Figure 3.2.

First, the customer sends a HTTP request containing the information about the operation they wish to perform. At that time, the Brand API attributes an identifier to that request and produces a HTTP response acknowledging the success of its reception.

Afterwards, the API processes the request asynchronously and builds a message to report the status of the process. That message contains a standard HTTP code relating to the success or failure of the process as well as the identifier previously assigned to the request of the customer. Additionally, a description of the error details is also included in the case of failure.

Finally, after sending the asynchronous response to the brand, a final HTTP message is received from the ERP informing if the response was successfully handled by the customer's system.

the format restrictions of each field.

A product's identifiers are its Stock Keeping Unit (SKU) and European Article Number (EAN). As so, it is required that each product has a unique pair of identifiers. As a consequence, physically different products cannot have the same SKU and each SKU is associated to only one EAN that cannot be re-used.

3.3.2 Purchase Orders Creation and Reception at Warehouse

A Purchase Order (PO) is a commercial object containing the information regarding goods that a brand ordered to a supplier. In Appendix A, Tables A.3, A.4 and A.5 summarize the available fields for a PO. Similarly to product creation, the integration will fail if the data inserted is in the wrong format, a mandatory field is empty or the object identifier already exists. Furthermore, as discussed in the product creation process, a PO can only be created if the products contained in it are already created in E-Fulfilment.

As seen in Figure 3.4, the PO creation process is not directly propagated to the warehouse after reaching in E-Fulfilment. This happens because Maersk needs information from the supplier regarding the warehouse to which the goods will be delivered and the date in which that will occur. After obtaining that knowledge, the E-Fulfilment team responsible for inbound orders imports the data into the system, creating a booking for the reception of a PO and informing the respective warehouse. Since, a PO can be divided into multiple receptions, the flow from E-Fulfilment notifies the warehouse about the products and quantities that will be received as well as which PO that reception concerns. It is important to highlight that a reception cannot be associated to more than one Purchase Order.

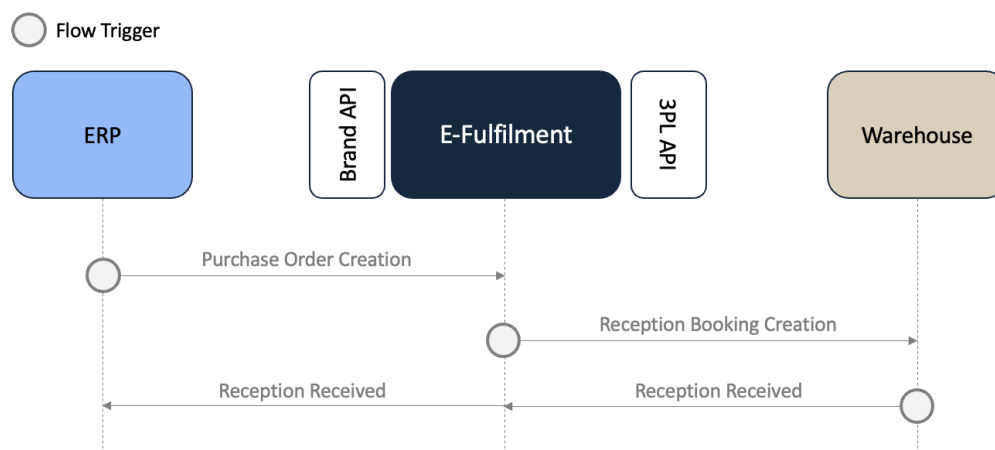


Figure 3.4: Process Flow of PO Creation and Reception at Warehouse

The final flow regarding POs occurs when a reception is received in the warehouse. When that happens, the items are checked and categorized as either compliant or non-compliant, depending on whether they are in proper conditions to be sold to customers. Afterwards, that information is communicated to E-Fulfilment and automatically reported to the ERP, raising the stock level

of the respective sales channel if the items are compliant. This means that while POs possess the potential to raise the stock of a product, that potential will only be fulfilled when a reception occurs.

3.3.3 Sales Orders Delivery Creation and Shipment Confirmation

A Sales Order (SO) represents a request from a customer to purchase products from the brand. As so, this object is comparable to a PO but from a customer point of view instead of a supplier perspective. However, unlike what happens in the suppliers process, the brand does not create the SO individually. Instead, single deliveries associated to a SO are created in the ERP. Nevertheless, the relation between SOs and deliveries is equivalent to that of POs and receptions. While one SO can have multiple deliveries, a delivery can only be associated to one SO.

The required as well as the optional fields of a SO delivery are presented in tables A.6, A.7, A.8 and A.9 of Appendix A. It is relevant to highlight that if the products to be shipped are not created in E-Fulfilment, the integration will also fail.

After the creation of the delivery in the system, the warehouse ships the items and sends a confirmation detailing the items that were shipped, the carrier and service level used, and the shipping track number. Simultaneously, the stocks of the products shipped are decreased accordingly throughout the entire system.

The fulfilment of a delivery can be total or partial depending on available stock. However, currently the information on the level of fulfilment is not reported to the ERP, meaning that the brand must analyze the items shipped to understand if the delivery was fully shipped.

The complete process flow of the creation of a sales order delivery and the confirmation of its shipping is illustrated in Figure 3.5.

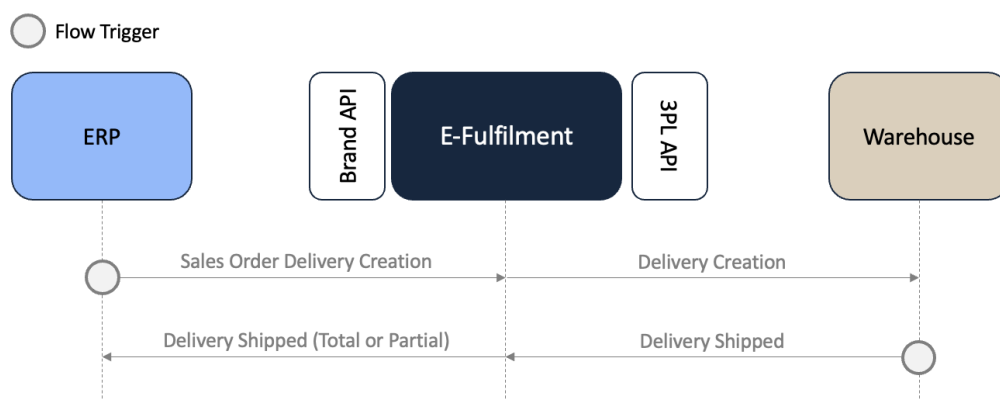


Figure 3.5: Process Flow of Delivery Creation and Shipment

3.3.4 Returns Received

If the final customer is not satisfied with the products received, it is possible for them to return them. The solution offered by Maersk is a pre-printed return label that is included in the items delivered at the moment the order is shipped.

Digitally, the process begins when a warehouse receives a return and reports to E-Fulfilment the amount of products returned and whether they are compliant or not, i.e., if they are in condition to be resold. The brand can check the details of the return in the ERP, including reasons for non-compliance of an item, aiding in the decision of a possible refund to the customer. Figure 3.6 depicts the flow of this process.

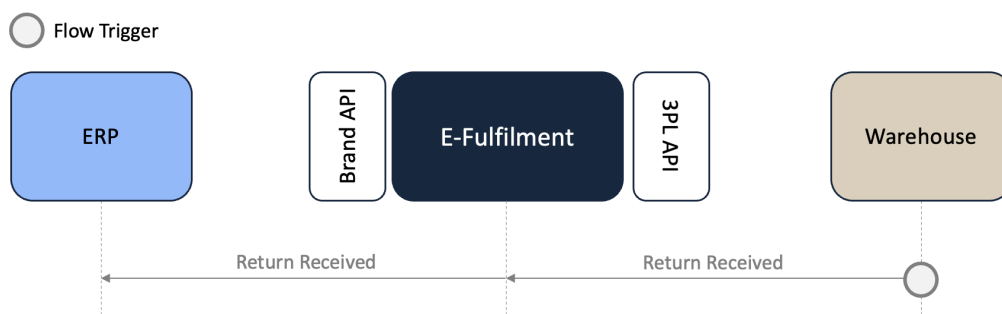


Figure 3.6: Process Flow of Returns Received

3.3.5 Stocks Update

In any logistic operation it is natural that changes in stock occur in situations other than purchases and sales. Examples of reasons this can occur are receptions being done by the box, meaning that not all the items are checked individually or degradation of items that become unfit to sell. As so, E-Fulfilment possesses a process that allows to update stocks in the platform and the ERP when that happens. This action, presented in Figure 3.7, is triggered by warehouse inventory operations or movements between stock allocation channels.

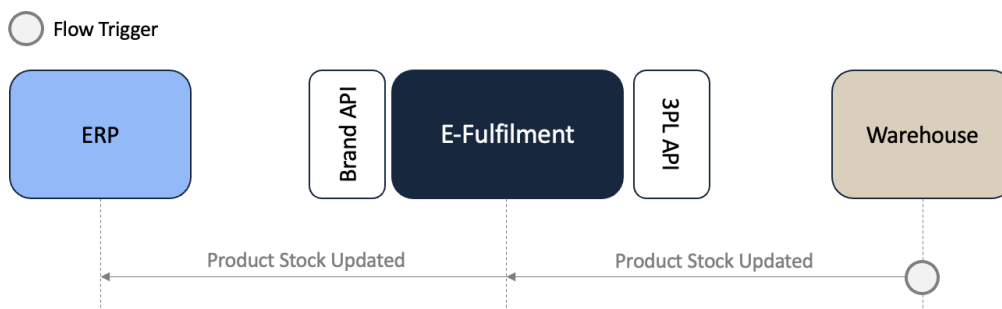


Figure 3.7: Process Flow of Stocks Update

3.3.6 Exception Flows

Other than the normal processes described in this section, there are certain exceptional flows that can occur from time to time due to misalignment of stocks between the different players involved. Evidently, it is important that all the parts of the logistic operation are synchronized in terms of stock to ensure that the brand does not sell items it does not possess. Due to that, there are techniques in place to minimize the problems caused by such an exception and to quickly synchronize stocks across the systems. There are two different types of synchronization failure that can transpire:

- **Misalignment of stock between systems**

This exception concerns a virtual inconsistency between the stocks of an item in the ERP and E-Fulfilment. The reasons behind it may be a failed communication to decrease stock from E-Fulfilment to the ERP or a manual stock operation in the ERP that incorrectly raised stock.

In this situation, whenever a delivery is created, the stock availability of the selected items is confirmed before communicating it to the warehouse. If none of the items are available, the delivery is cancelled and it is reported to the ERP that the service was denied due to lack of stock. Alternatively, if only some of the items are unavailable, the delivery is partially shipped. As discussed previously, the brand can only confirm if a delivery was fully shipped by comparing the items documented in the shipping confirmation.

In any of the previous situations, E-Fulfilment also sends an absolute stock adjustment to the brand, setting the stock to zero to avoid creating more delivery requests containing that product.

- **Misalignment of stock due to warehouse discrepancy**

This exception represents a physical discrepancy in stock, i.e., the real stock in warehouse is not synchronized with the virtual in the ERP and E-Fulfilment. The possible reasons for this divergence were already explored while discussing the Stocks Update process.

If the warehouse has some of the items requested, the delivery is partially shipped. As in the previous misalignment, the brand must check the items in the shipment confirmation to understand if the order was fully fulfilled or not. If there are no items available, the delivery will be cancelled. However, currently the Brand API does not have the functionality to send that information to the ERP, meaning that the E-Fulfilment employee responsible for managing the brand must notify them of the cancellation.

To synchronize the virtual and physical stock, when any of the previous situations occur, the previously described Stocks Update process is triggered.

3.4 Current Objectives and Opportunities

As discussed in Section 2.3.2, product discovery is an essential step in the development of product roadmaps since it allows to identify items to place in the roadmap. Due to this, it is indispensable to contextualize the current company objectives and expected outcomes as to make sure that the opportunities identified and described in chapter 4 are aligned with Maersk's vision. To accomplish that, a top-down analysis will be conducted with the visual aid of an OST (Figure 3.8). This will serve as the basis for the creation of a product roadmap.

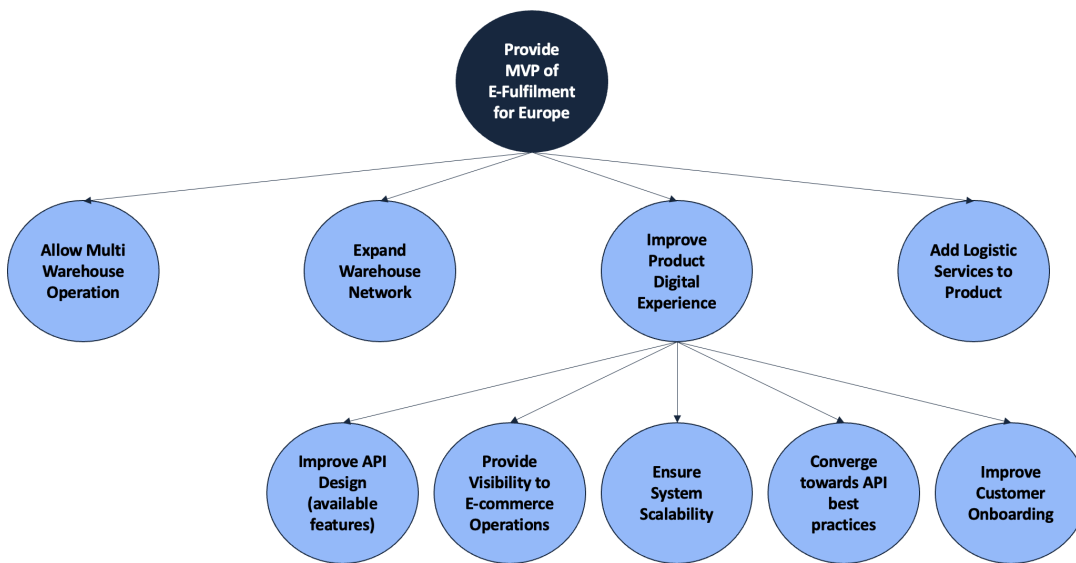


Figure 3.8: OST representing Maersk's current objectives and opportunities

It is important to highlight that only the elements relating to the project and its expected outcome will be presented. Such a constraint is aligned with the practices discussed in Section 2.3.1 which state that various representations of a product roadmap should be mapped out depending on the information required.

Starting from the top, the desired outcome is to provide an E-Fulfilment solution for Europe. This means developing a Minimum Viable Product (MVP), so customers can begin using the platform to run their ECL operations in Europe.

To achieve this outcome, there are already high-level opportunities being explored by the company regarding the warehouse operations provided, the size of the warehouse network, the quality of the product digital experience, and the additional logistic services offered. Considering this dissertation's objectives, it is particularly relevant to focus on opportunities connected to the improvement of the product's digital experience. As mentioned previously, this focuses specifically on the case of brands integrating the platform with an ERP.

Descending in the OST, the opportunity to improve the product's digital experience unfolds into five further opportunities:

- Improve API design (available features);
- Provide visibility over e-commerce operations;
- Ensure system scalability;
- Converge towards API best practices;
- Improve customer onboarding.

Taking into account the content reviewed in chapter 2, it is possible to associate the API as a product approach to the opportunities to improve API design and customer onboarding. Considering the objectives of this project, these two initiatives can be categorized as very relevant to the work developed.

This exercise of aligning the appropriate research to be done with its underlying business motivation, means that the next stage of actually identifying the improvement opportunities will be undertaken with greater overall context, which in turn can enhance the adequacy of the whole process.

Chapter 4

Identification of Improvement Opportunities

The objective of the current chapter is to explore and describe in detail the possible improvements to the Brand API as a product. Afterwards, an analysis of the urgency of the initiatives is conducted and the RICE scoring model is used to establish priority rankings.

4.1 API as a Product Opportunities

This section presents opportunities to improve the functionalities available in the Brand API as well as the overall onboarding experience of the customer. These are closely related as features initially conceptualized for the onboarding stage can be useful in ongoing stage and the other way around. Moreover, the design of an API also influences significantly the process of onboarding.

Addressing the parallelism of concepts explored in section 2.3.1, it can be stated that improving the API as a product represents a theme and the opportunities identified correspond to the epics driven by that theme.

4.1.1 Product Data Structure

Currently the data structure of products created in E-Fulfilment is very fashion-oriented. Such a statement can be confirmed by analyzing the existing fields for products and verifying fashion specific characteristics such as the season of an item, the size (e.g., small, medium, large), target gender, target age group or even the fabric of the item.

The reason for this focus is the fact that HUUB only operated with brands of the fashion industry. On the other hand, Maersk works with brands that commercialize their items in a wide range of different markets. In order to deal with the widening of types of product offered, the fields that do not fit into this holistic perception were deemed as optional for integration and brands started having the option to use those fields as a space to add information about other characteristics their products may have.

Even though the initial solution allows to integrate more customers it is not very intuitive for the brands. As so, it increases the efforts of integration and the work of the company's support team. With this mind, it becomes pertinent to research and study possible procedures to change the data structure of products in the platform so it becomes more agnostic, i.e., easily adaptable to the generality of items.

4.1.2 Reception Creation

The analysis of the processes flows conducted in chapter 3 pointed out inefficiencies in some of the current operations. One to be considered is in the booking of receptions as, at the moment, the functionality for the brands to create receptions in their ERP is not available. This contrasts with the delivery creation action which is the counterpart for receptions within the customers processes.

As a consequence, this process incurs on an extra flow, extending the overall time to execute it. Additionally, it requires supplemental effort from Maersk's inbound team as its members must receive the information and manually insert it in E-Fulfilment. In a scenario where the scale of brands in the platform is considerable such a situation may become very difficult to handle. Taking this into account, exploring solutions to allow the creation of receptions directly on the ERP assumes great relevance.

4.1.3 Handling of Exception Flows

Another process which is holding back the value of the API as a product is the handling of exception flows, more specifically the communication of partial shipping or order cancellation. In fact, as mentioned in chapter 3 for a brand to know that a delivery was not fully fulfilled it must compare the items it requested to be shipped with the items present in the actual shipping confirmation. In other words, the brands are required to analyze the level of fulfilment of an order for themselves.

The current situation does not add much value to the user experience, yet, as presented in Section 2.2, providing a good UX is a crucial factor in the API as a Product approach. Being that the case, it is essential to explore methods to include the information of whether a delivery was fully shipped or not when sending the details of the shipment confirmation. Finding a solution for this would increase UX and consequently their satisfaction with the product.

Regarding the cancellation of deliveries, there is a disparity between those caused by virtual misalignment of stock and the ones caused by a physical misalignment. As of now, in the first situation the cancellation of the order is propagated via API, meaning that it is automatically reported to the ERP. However, in the second scenario that communication is not implemented, meaning that the Maersk's personnel responsible for managing that brand must check those occurrences and inform the brand that they need to cancel that order in their ERP.

Given the fact that a similar functionality is already deployed, working on mirroring that solution could help reduce the workload of Maersk's brand managers.

4.1.4 Relation between POs and Containers

As indicated in chapter 1, Maersk's new clients include Chinese companies intending to establish their e-commerce operations in Europe. Through contact with those customers, it was possible to assess some necessities which E-Fulfilment is currently not capable to fulfil. One of them concerns the relation between a PO and a container.

Firstly, it should be clarified that a container corresponds to what a supplier sends to a warehouse. From the brand's point of view it matches the reception of items in that warehouse. As presented in chapter 3, a reception can only be associated with one PO. However, in the case of Chinese brands, since they already have an operation in Asia with their own warehouses, they will be their own suppliers for their European operation. As so, since they are in control of both creating and supplying their POs for their European operation, in reality a shipped container may carry items from more than one PO.

In order to fulfil the needs brought up by the clients, it becomes pertinent to study options which can bring the reality of their operation and the virtual representation closer together.

4.1.5 Character Translation

Another opportunity that ensued from the contact with Chinese clients is the possibility to translate characters from the Chinese alphabet to the Latin. Evidently, due to their origin and main market location, Chinese clients use their mother tongue in their internal systems. However, as of now, the Brand API is not able to process Chinese characters. Consequently, the complete integration between a brand's ERP and E-Fulfilment is dependent on a previous translation to English in the client's side.

Since customers from China are one of the main targets for E-Fulfilment in Europe, it may be advantageous to consider investing in an internal solution for this issue. In fact, by doing so customers would be relieved from having to spend their resources in translation. Such a feature could attract more brands and even be a factor differentiating Maersk from its competitors.

4.1.6 Barcode Options

The final opportunity uncovered from the Chinese clients' needs concerns the barcode used in E-Fulfilment. Presently, the option available is exclusively the EAN. However, some clients do not have an EAN associated with their products.

As introduced in Section 3.3, for a brand to integrate a product in E-Fulfilment it must include the respective EAN. If clients are not able to provide this barcode, the integration with the platform is not possible. Such a restriction means that Maersk would be losing clients for not being able to adapt to their operation. Logically, looking at the API as a product, its success can be measured by the number of clients who are using it. As so, it is crucial to understand what kind of solutions can be developed to integrate these clients.

4.1.7 Error Communication

Whenever a brand attempts to integrate objects which contain invalid data, it receives a response from the API notifying that there are errors in their request. The objective of that communication is to help the brands understand the existing issues and autonomously correct them. However, at the present time that procedure is not optimized, making it more challenge to achieve those goals.

The first shortcoming encountered concerns the inability to report multiple errors. In fact, the API interrupts the problem search when it detects an issue. Due to this, the brands are only informed about one error even if more are present. As a consequence, the number of creation attempts until a brand is able to integrate an object with E-Fulfilment increases. Being able to report multiple errors at once would not only drastically reduce the time it takes for Maersk's customers to synchronize their operations with E-Fulfilment but also relieve the work of the company's support team.

A further limitation identified is in the messages used to describe the errors. As a matter of fact, presently the error notes are very technical and often difficult to understand without deep knowledge of how the system functions. Due to this, the brands frequently need to rely on Maersk's support team to clarify what is wrong with their requests. As so, in order for brands to be able to correct their mistakes more autonomously it becomes essential to make the error messages more descriptive and informational.

4.1.8 Integration Errors Visibility

As just discussed, the visibility of the brand regarding integration errors is restricted to the HTTP response sent from the API after processing a request. This means that the brands need to have technical resources allocated that can handle and read that information. However, many of Maersk's customers lack those capabilities and while they can confirm the success of an object integration by accessing E-Fulfilment and checking if that object is there, the information regarding failures is not displayed on the platform.

From HUUB's previous experience, this situation leads to clients not understanding the problem and overloading Maersk's support team with help requests. As discussed previously, such a scenario is far from ideal as for the most part the brand should be able to use the API in a self service fashion. This is especially problematic during the onboarding stage as clients are more likely to make mistakes because they are still getting adapted to the software.

Again, it is indispensable to highlight that if this case is already troublesome for Maersk's internal teams, the potential scalability of the platform makes it crucial to address it. In fact, in an operation of considerable scale it would be unfeasible for the support teams to aid every single brand in every failed integration attempt. As so, it is urgent to develop a feasible solution to provide brands with visibility over their integration errors.

4.2 Prioritization of Opportunities

Having identified the improvement opportunities, before adding them to the product roadmap and start exploring solutions it is necessary to evaluate them and establish a priority regarding their development. To make that assessment the RICE scoring model was used.

In order to simplify the referral of each opportunity throughout this process each of them was attributed a label:

- O1 - Improve product data structure;
- O2 - Allow ERP reception creation;
- O3 - Improve handling of exception flows;
- O4 - Allow containers to be associated with multiple POs;
- O5 - Provide character translation;
- O6 - Allow more barcode options for product integration;
- O7 - Improve error communication;
- O8 - Provide visibility over integration errors.

Table 4.1 depicts the estimated value of the RICE factors for each opportunity. In order to calculate those values, the methods presented by McBride (2018) were considered.

Table 4.1: Estimation of RICE factors for each opportunity

	Reach	Impact	Confidence	Effort
O1	3	1	80%	1
O2	7	2	80%	4
O3	5	1	50%	2
O4	2	1	100%	3
O5	2	0.5	80%	2
O6	2	1	100%	1
O7	6	2	80%	3
O8	8	3	100%	2

Using the formula presented in 2.3.3 a RICE score for each initiative can be calculated and the opportunities prioritized accordingly. Table 4.2 presents an ordered summary of that information.

Table 4.2: Priority ranking of opportunities

	Opportunity	RICE Score
1	O8	12
2	O7	3.2
3	O2	2.8
4	O1	2.4
5	O6	2
6	O3	1.25
7	O4	0.67
8	O5	0.40

As shown in Table 4.2, it was estimated that providing visibility over integration errors was the most urgent matter to attend to. As so, in chapter 5 this opportunity/epic will be further detailed and appropriate solutions will be explored.

Chapter 5

Solution Design

In this chapter the methods adopted to develop a solution for the integration errors visibility problem presented in the previous chapter are described. Firstly, further context is provided to assert the relevance of improving the current situation. Afterwards, a north star vision of the solution is depicted, i.e., a visual representation of the ultimate goals of the solution. Finally, it is explored the development of a Minimum Viable Product (MVP) solution derived from the north star design.

5.1 Initiative Background

As outlined in chapter 4, brands are not provided with visibility over integration errors. In the past, this proved to be problematic for the teams supporting the customers. In fact, the only brand which worked with HUUB using an ERP (NU-IN) needed to constantly reach out to the company for support in solving issues. More than overloading the employees with work, this kind of situation stops them from working on other features or projects which could be of more value to the organization.

The historical data regarding NU-IN's requests was retrieved from the company's databases (POSTGRESQL and MongoDB) and analyzed to understand the real impact of this issue. For this it was considered the time interval between June 2020 and June 2021 which corresponds to the period in which the brand operated with the platform.

In Table 5.1 it is summarized the number of monthly integration requests sent by NU-IN as well as how many of those resulted in an error. From that data it is possible to infer that the minimum number of requests containing errors in a month was 11,958 and the maximum 117,723. Given the short time interval, it is possible to say that even in the month with the least integration errors the support team needed to be constantly contacted. Evidently, in any of the cases it is possible that the brand submitted identical requests multiple times, meaning that it would not need support with all of them. Nonetheless, it is important to consider that even in that scenario these only represent the requests of a single brand. Since the platform needs to be able to integrate with multiple customers, it is unsustainable to let this situation go unaddressed.

Table 5.1: NU-IN's historical request data

	Total Requests	Requests with Errors
June 2020	83 188	63 425
July 2020	116 291	93 039
August 2020	124 247	103 371
September 2020	147 124	106 857
October 2020	164 734	117 723
November 2020	154 753	87 994
December 2020	112 739	41 641
January 2021	130 863	80 425
February 2021	82 042	26 959
March 2021	76 430	17 426
April 2021	48 708	11 958
May 2021	49 362	12 538
June 2021	60 297	16 548

Figure 5.1 illustrates the monthly evolution of the percentage of the total integration requests with errors. Analyzing it, aside from a spike in January, the percentage of requests which contained errors decreased over time. This is easily explainable because in the first months of operation, NU-IN was still getting adjusted to the API, consequently being prone to make more mistakes. The more comfortable and adapted to the product the brand became, the less errors it made. From this, it is possible to state that providing visibility over integration issues is even more crucial during the onboarding stage of a customer. Since at the moment of the project several brand are starting their operations in E-Fulfilment this initiative becomes especially relevant.

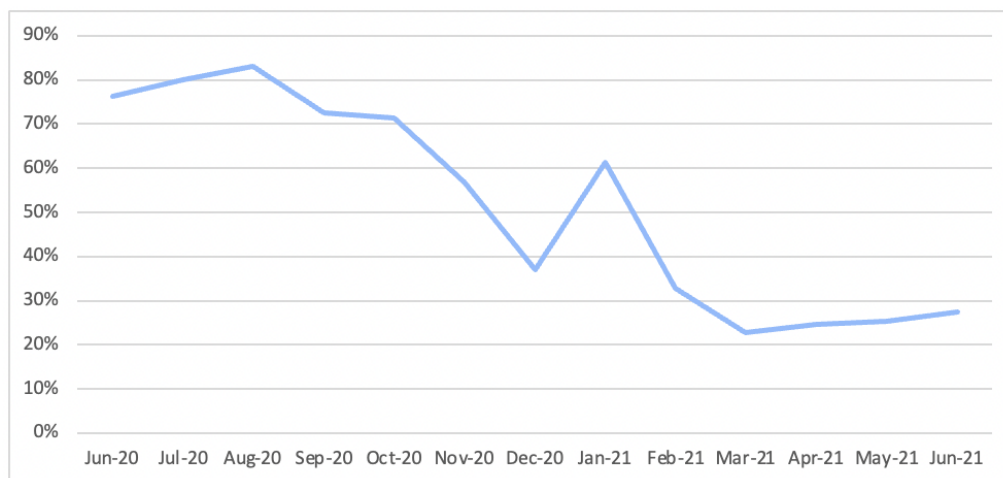


Figure 5.1: Percentage of integration requests with errors in each month

5.2 North Star Design

The objective of the north star scenario is to establish an ideal solution as a goal to reach. For the development of the north star solution it is necessary to turn abstract customer needs into design requirements fulfilled by a concrete visual solution. As so, this section is divided into two main stages: the definition of the design requirements and the concretization of those requirements into a visual implementation.

5.2.1 Design Requirements

Previous to the design of the solution, a number of intended functionalities based on customer needs were defined. In order to organize the design process, these requirements were grouped into six categories.

1. Solution Availability

One of the big objectives in providing visibility over errors is to decrease the amount of support that needs to be provided to brands. In other words, the goal is to give the means for customers to be as autonomous as possible. To accomplish that it is fundamental that the feature developed is available at any time for the client to consult. This implies that there should be no need for a brand to ask Maersk if there any integration errors at a certain time as they should have access to that information at all times.

2. Information Update

The information displayed should be up to date. This means that only the latest incorrect integration request should be reported for each object. Furthermore, if an object which previously contained errors is successfully integrated, it should not be presented anymore.

3. Information Recognition

The information provided by the solution should be recognizable to the customers, i.e., a client should be able to associate an error to a specific object and request.

4. Information Comprehensibility

The information provided should be comprehensible to the customers. In other words, the brand should be able to understand what caused the integration to fail.

5. Information Usefulness

The solution should provide information regarding how to correct the existing integration errors.

6. Information Manipulation

The information displayed should be manipulable. This means that a customer should be able to interact with the solution to restrict the data displayed.

5.2.2 Solution Mock-up

The basis of the solution designed is a space dedicated to this issue in the E-Fulfilment platform. As of now, the brands already have access to the E-Fulfilment website where they can check which objects are created in Maersk's system as well as consult information regarding their operations. Since they can access that data any time they wish by logging in the platform, this implementation complies with the system availability restriction.

Regarding the location of the information, two main areas were defined. The first is the page displayed immediately after the website confirms the user credentials and starts the session. There, a dashboard indicating if any integration errors exist in each type of object is presented. Since this issues may compromise the synchronization between systems, it is shown before anything else to warn the brand to take corrective actions.

The second space of information is a page detailing all the errors concerning a type of object. Each of the possible objects (products, POs, SOs) has its own individual page that a customer can reach directly through the initial dashboard or by clicking on the notifications displayed on the side menu. That page contains a table enumerating every item with integration errors. Since that table is where most of the data is shown it is important that it follows the information requirements presented previously.

The first aspect in exhibition is the date in which the last request associated with the specific object was made. After that, the references which allow the item identification are shown. This answers the need set by the information recognition constraint. However, since one of the possible integration errors is the lack of object references, that cell may be empty as it is impossible to associate the request with a specific item. Next, to fulfil the need for information comprehensibility, the existing errors and the data fields in which they occurred are depicted. Finally, addressing the information usefulness requirement, a small explanation on how to fix the issue is given.

Even though an object can have multiple errors, in the table only one is represented. If one of the errors concerns the object identifier, that will be the one depicted in the row. However, in the field and issue description cell there will be an indication of how many more problems are associated with the item. That information is displayed on a side panel that can be accessed by clicking on that specific row of the table. Additional information contained in the side panel includes an identifier of the last request associated with that object and, in the case of products, the list of POs and SOs that could not be integrated because of that product.

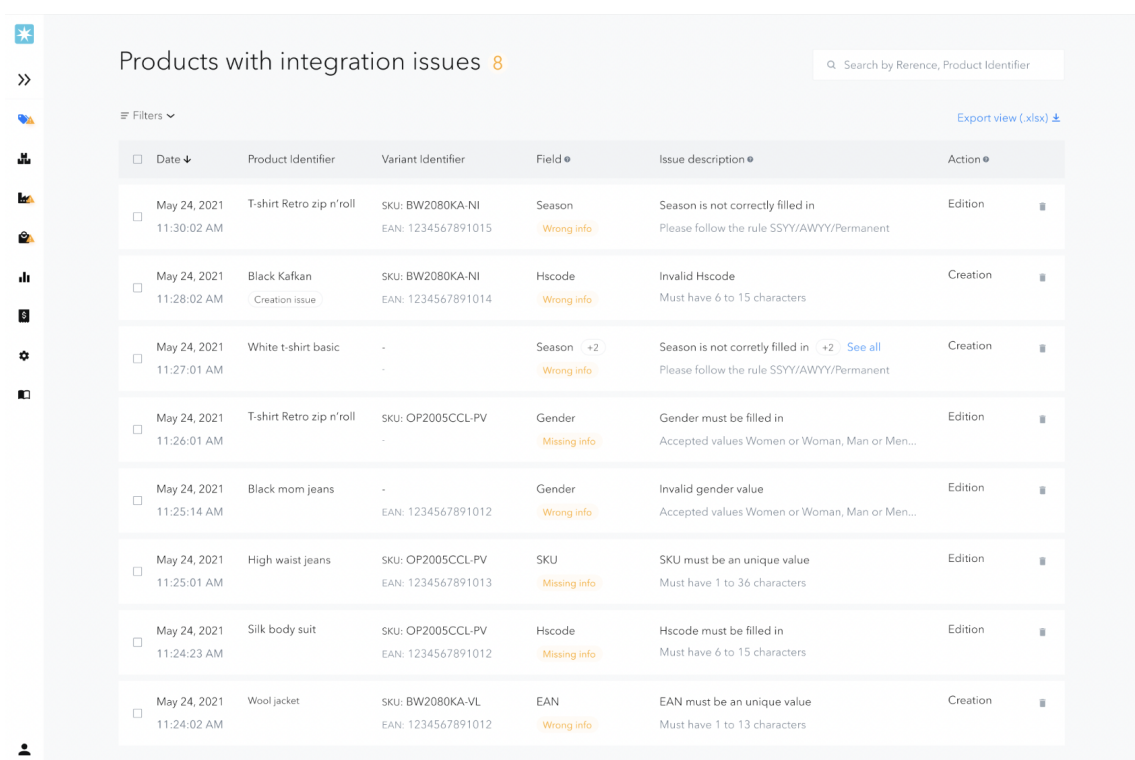
Initially, all the objects whose integration failed are represented in the page. However, the brand has the possibility to restrict what is shown by applying a filter. The general filters conceptualized are: date range; issue type, i.e., missing info or invalid info; field with error. In the case of the products page it is also possible to filter by other objects whose integration is failing because of an error in that product. This happens whenever a PO or SO cannot integrate due to having products that were not successfully integrated.

Another functionality included is the selection in bulk of the records in the table. After selecting objects is possible to dismiss the issues associated with them. This dismissal can also be done

individually for each object. This feature is especially useful in situations where the brand has no interest in integrating an object anymore as it was removed from their system. A further use is in the case of duplicated or empty references errors, since it is impossible to identify the object affected and therefore understand when an error has been corrected.

The user can also find a specific object, by looking it up in the provided search bar using their unique reference. Finally, an option to export the data on view to an excel file is provided as well.

Figure 5.2 illustrates the mock-up of the page that shows the integration errors of products. In Annex B the pages of the remaining object types, the initial dashboard and all other figures relevant to the visualization of the solution are depicted.



☐	Date ↓	Product Identifier	Variant Identifier	Field ●	Issue description ●	Action ●
☐	May 24, 2021 11:30:02 AM	T-shirt Retro zip n'roll	SKU: BW2080KA-NI EAN: 1234567891015	Season <i>Wrong info</i>	Season is not correctly filled in Please follow the rule SSYY/AWYY/Permanent	Edition
☐	May 24, 2021 11:28:02 AM	Black Kafkan <i>Creation issue</i>	SKU: BW2080KA-NI EAN: 1234567891014	Hscode <i>Wrong info</i>	Invalid Hscode Must have 6 to 15 characters	Creation
☐	May 24, 2021 11:27:01 AM	White t-shirt basic	-	Season +2 <i>Wrong info</i>	Season is not correctly filled in +2 See all Please follow the rule SSYY/AWYY/Permanent	Creation
☐	May 24, 2021 11:26:01 AM	T-shirt Retro zip n'roll	SKU: OP2005CCL-PV	Gender <i>Missing info</i>	Gender must be filled in Accepted values Women or Woman, Man or Men...	Edition
☐	May 24, 2021 11:25:14 AM	Black mom jeans	- EAN: 1234567891012	Gender <i>Wrong info</i>	Invalid gender value Accepted values Women or Woman, Man or Men...	Edition
☐	May 24, 2021 11:25:01 AM	High waist jeans	SKU: OP2005CCL-PV EAN: 1234567891013	SKU <i>Missing info</i>	SKU must be a unique value Must have 1 to 36 characters	Edition
☐	May 24, 2021 11:24:23 AM	Silk body suit	SKU: OP2005CCL-PV EAN: 1234567891012	Hscode <i>Missing info</i>	Hscode must be filled in Must have 6 to 15 characters	Edition
☐	May 24, 2021 11:24:02 AM	Wool jacket	SKU: BW2080KA-VL EAN: 1234567891012	EAN <i>Wrong info</i>	EAN must be a unique value Must have 1 to 13 characters	Creation

Figure 5.2: Mock-up of page displaying product integration errors

5.2.3 Dependencies

Analyzing the final design of the north star solution, a few dependencies on other opportunities can be identified. More specifically, the mock-up produced assumes that the API is able to detect multiple errors in a request and that it constructs a clear, informational message which helps customers solve their issues. As discussed in chapter 4, currently none of those features is implemented meaning that in order to develop a working solution similar to the north star the initiative of improving error communications would have to be explored as well.

5.3 MVP Design

As stated previously, the north star design represents an ideal solution to the problem at hand. However, its exact development entails a great investment of time and resources as well as dependencies on unrelated opportunities. During that time, it is possible that the surrounding circumstances change and the project loses relevance. Furthermore, it would be unfeasible to complete a development of that dimension during the execution of this dissertation. As so, it is essential to divide the work and continuously deliver smaller implementations. This is aligned with the concepts described in chapter 2 regarding continuous product discovery.

Taking this into account, it is crucial to make a comparative analysis between the present situation and the north star scenario and conceive an initial solution to approximate the two settings. In this section, a detailed description of the methodology followed to reach a working solution is presented.

5.3.1 Methodology

The first step in the solution design was to define the basis of the development. While in the north star the implementation was imagined on the E-Fulfilment platform, such an answer would not be feasible for the initial rollout. As so, it was decided that the concept would consist of sending daily emails to the brands reporting the integration errors of that same day. An email aggregating the information from all the brands would also be sent to Maersk's support team to keep them updated on the customers' situation.

After that, it was required to understand which data would be displayed on the emails and how it would be gathered. In fact, the requests and responses of the Brand API are stored in an internal database. However, that database is based on MongoDB which is non-relational. That characteristic makes it easier to store large amounts of data but harder to work with it, since its purpose is not to establish relations between the information it contains.

However, the MongoDB database is not the only one kept by Maersk. In reality, a relational database built on PostgreSQL is also available. There, all the information regarding brands and their E-Fulfilment objects (Products, POs, SOs) is stored. Additionally, the logs of the API processes are also preserved, showing for each event if it was successful or not. However, those logs do not contain any information regarding the objects associated with the processes.

The properties of the two databases lead to a situation where the simplest method to adopt is working with both at the same time, using the relational database to retrieve the requests with errors and the non-relational to associate those requests with different objects. Since the goal is to portrait solely the errors which have not been corrected yet, after associating the requests with objects it is required to check if any of those objects has already been integrated in the PostgreSQL database.

Having gathered the data, the subsequent step was to define how to register that information in an email and send it to the intended recipient.

Evidently, it is not practical to execute all the tasks involved in this process manually every day, including writing individual emails to every customer reporting the findings. As so, it was essential to devise a plan to automate the process of retrieving the data from the databases, preparing it, formatting all the emails and sending them. To accomplish this, an algorithm was developed using Python as the programming language.

5.3.2 Solution details

The sequence of steps of the constructed algorithm is presented below. Furthermore, a flowchart illustrating the process of the solution is depicted in Annex C.

1. Access relational database and retrieve requests for creating purchase orders which returned an error on that day. Other than the request identifier, the date of the request and the error details are also collected.
2. Create a query to check in the non-relational database which of the retrieved requests do not possess an object identifier. Associate those requests with the brand that made them.
3. Create a query to check in the non-relational database which of the retrieved requests failed because the object identifier already existed. Associate those requests with the brand that made them and the object reference given.
4. Create a table with both sets of requests that have been treated. These represent requests in which it is not possible to track error correction, since it is not possible to associate them with an object.
5. Create a query to check the remaining requests. That query will aggregate the requests according to their object reference and brand. Only the latest request for each pair of reference and brand is saved, as the others represent previous attempts and are not up to date.
6. Access the relational database again to check if the objects references of the previous step already exist. If they do, those requests are disregarded. The reason for this is that the objects they refer to have already been integrated, meaning that the error they report is no longer relevant.
7. Create a table with the requests still in consideration. In contrast with the previous table, this displays requests in which the error correction can be tracked, as there is a specific object associated with them.
8. Repeat all the procedure for the other two processes of object integration: creating sales order deliveries and creating products.
9. Format all the created tables in HTML and put them as a body of an email.
10. Send the created email to Maersk's support team.

11. Divide all the tables by brand.
12. Send an email to each brand containing the information about their respective integration errors.

An example of the section depicting errors in product integration in the email sent to Maersk's support team is represented in Figure 5.3.

Products Integration Errors					
Objects with errors					
Brand Name	Request Date	SKU	EAN	Error Message	Request Identifier
Brand X	2022-05-26 07:34:25.532760+00:00	32999111114	506062479	Validation Error. '506062479' is too short. Review: data (position: 0) > variants (position: 0) > ean .	628f2d810f47410009df421f
Brand X	2022-05-26 07:12:14.666760+00:00	32999111114	506062476	Validation Error. '506062476' is too short. Review: data (position: 0) > variants (position: 0) > ean .	628f284e2b66f40009524931
Brand X	2022-05-26 02:53:41.232451+00:00	32999111114	506062477	Validation Error. '506062477' is too short. Review: data (position: 0) > variants (position: 0) > ean .	628eebb50f47410009df420b
Duplicated and Empty References					
Brand Name	Request Date	SKU	EAN	Error Message	Request Identifier
Brand Y	2022-05-26 08:04:33.554370+00:00	-	-	Validation Error. 'data' is a required property.	628f34912b66f40009524945
Brand Y	2022-05-26 07:22:54.208378+00:00	-	-	Validation Error. 'data' is a required property.	628f2ace2b66f4000952493c
Brand Y	2022-05-26 07:20:10.745236+00:00	-	-	Validation Error. 'data' is a required property.	628f2a2a2b66f4000952493a
Brand Y	2022-05-26 06:28:01.257974+00:00	-	-	Validation Error. 'data' is a required property.	628f1df12b66f4000952492e
Brand Y	2022-05-26 02:29:10.131392+00:00	-	-	Validation Error. 'data' is a required property.	628ee5f60f47410009df4205
Brand Y	2022-05-26 02:28:45.598509+00:00	-	-	Validation Error. 'data' is a required property.	628ee5dd2b66f40009524921
Brand X	2022-05-26 02:19:17.143936+00:00	32999111114	5060624771127	EAUs already inserted: 5060624771127	628ee3a50f47410009df41fc
Brand X	2022-05-26 01:39:46.555581+00:00	32999111112	5060624771125	EAUs already inserted: 5060624771125	628eda622b66f4000952490b
Brand X	2022-05-26 01:38:39.266679+00:00	32999111112	5060624771125	SKUs already inserted: 32999111112	628eda1f0f47410009df41ed
Brand X	2022-05-26 01:35:39.746197+00:00	32999111112	5060624771124	EAUs already inserted: 5060624771124	628ed96b0f47410009df41eb

Figure 5.3: Section of email sent to Maersk's support team representing errors in product integration.

Chapter 6

Results

This chapter displays the results of the project. Section 6.1 exhibits the final API product roadmap updated with the initiatives discovered in chapter 4. Then, in Section 6.2 the impact of the implementation described in chapter 5 is evaluated.

6.1 API Product Roadmap

Having properly identified and prioritized the improvement opportunities it is possible to build a version of a product roadmap. Again, since the only theme relevant within the scope of the project is approaching the API as a product, the version of the roadmap presented only addresses content related with that. Furthermore, the time frame considered solely includes the quarter in which the project was developed and the three subsequent months, i.e., the second and third quarters of 2022.

In order to comply with roadmapping good practices, the level of detail of the items was adjusted depending on their respective quarter. As so, for the first period, the specific features to develop were specified, while in the next only the general objectives to address were stated.

Regarding the content, for the period in which the project took place only the opportunity in which the work was centered was included. As mentioned, both the feature to develop and the objective that drives it were described. This corresponds to the solution presented in the previous chapter.

For the following quarter, the two opportunities categorized with the greatest priority were defined as the objectives to fulfil: improving error communication and allowing the creation of receptions in the ERP. This contrasts with the first period, where only one objective was addressed. However, the reason for having a sole opportunity was the lack of a defined roadmap regarding this theme at the beginning of that quarter. Since that is not a problem in the future, more initiatives can be explored in the same space of time.

Figure 6.1 depicts the visual representation of the API product roadmap described in the current section based on the format discussed in Section 2.3.1.

Product Vision: Provide E-Fulfilment Solution for Europe	
Q2 2022	Q3 2022
API as a Product <u>Provide Visibility Over Integration Errors</u> Daily email containing all the relevant information for brands to correct their integration errors autonomously.	API as a Product <u>Improve Error Communication</u> <u>Allow ERP Reception Creation</u>

Figure 6.1: API Product Roadmap Produced

The remaining opportunities, which were discovered but not incorporated into the product roadmap, were stored in a product backlog. That way, the details regarding those initiatives are not disregarded, instead being saved for a moment in which the exploration of those opportunities is more pertinent to the company objectives.

6.2 Evaluate Impact of Solution

As stated previously, during the development of the project there were not any brands actively working with E-Fulfilment, with the first five only beginning their onboarding at the end of that period. Consequently, the empirical data has not been produced yet, meaning that it is not possible to directly assess the impact of the solution. However, there is data available regarding company resources and customers which allows a reliable estimate to be calculated.

Since it was established in Section 5.1 that this feature is particularly relevant in the onboarding stage and at the moment of this dissertation all the existing customers were in that phase, the evaluation is centered on that period. Regarding the factors affected, two dimensions were explored: the direct impact on the support resources during onboarding and the influence on the number of clients that can be integrated at the same time. At the end, the assumptions made throughout the estimation process were analyzed.

6.2.1 Support Resources

The aim of the implementation is to completely eliminate the need for providing support to integrate objects with E-Fulfilment. Since the brand is given visibility over those issues, it can autonomously understand and correct the errors without contacting Maersk.

To quantify the impact on the support resources, it is required to first analyze which resources are currently allocated to that activity. At the moment, a total of three people devote ten weekly hours each to help five brands in the onboarding phase rectify integration issues. Given that the average onboarding period lasts three months, by approximating each month to 4 weeks it can be determined that each brand requires a total of 72 hours of support during the integration stage.

It is important to highlight that the process of retrieving information regarding integration errors is complex, meaning that the workforce deployed needs to be highly skilled. As this employees are specialized, they require higher wages which leads to considerable accumulated costs. Since it is estimated that the average hourly rate of the workforce is 25€, the present total cost of supporting an onboarding brand is 1,800€.

Table 6.1 summarizes the distribution of resources just described. From this evaluation, it can be concluded that with the feature developed, for each brand it integrates Maersk can save 1,800€ and free 72 hours of work for specialized workers.

Table 6.1: Summary of current resource allocation

Variable	Resource
A - Weekly time allocated to support/person	10 hours
B - Workforce	3 people
C - Integration period	3 months
D - Number of brands in integration	5 brands
E - Total onboarding support time/brand ¹	72 hours
F - Support cost/hour	25 €
G - Total onboarding support cost/brand ²	1,800 €

$$^1 E = \frac{A \times B \times C \times 4}{D}$$

$$^2 G = E \times F$$

6.2.2 Number of Onboarding Customers

For a company of the dimension of Maersk, the money savings registered with support resources are almost negligible. However, the savings in time present an opportunity to greatly increase earnings, as one of the objectives of the company is to integrate the maximum number of brands with E-Fulfilment. Evidently, the employees which no longer have support tasks assigned to them still need to work the same hours as before. The consequence of this is an increase in capacity regarding the number of customers onboarding at the same time.

It is expected that the new capacity is of 8 integrations at the same time, i.e., an increase of 3 brands. As it is known that the average annual income that a customer brings to Maersk is 3,000,000€, an estimate of the increase in earnings can be calculated. For that prediction a few assumptions were made:

1. Every integration period lasts a quarter;
2. A brand only generates income after being fully integrated;
3. In each quarter Maersk integrates the maximum number of brands.

Taking all the information into account, in each year 4 waves of integration occur. With the implementation, in each of those waves there are three additional brands onboarded. Since each

brand generates 750,000€ per quarter for Maersk, each wave will produce more 2,250,000€ than before each quarter in which it is fully integrated. Table 6.2 depicts the extra earnings per quarter that can be achieved by integrating three more brands in a year. It is important to mention that the extra income brought by brands integrated in the same year is considered. This means that in the first quarter no earnings are considered as no brand is fully integrated yet. Additionally, the fourth wave of onboardings does not generate any value in that period.

Table 6.2: Quarterly income generated by extra onboardings in the same year

	2nd Quarter	3rd Quarter	4th Quarter
1st Quarter Extra Onboardings	2,250,000 €	2,250,000 €	2,250,000 €
2nd Quarter Extra Onboardings		2,250,000 €	2,250,000 €
3rd Quarter Extra Onboardings			2,250,000 €
Quarter Total	2,250,000 €	4,500,000 €	6,750,000 €

With this information, it is possible to estimate that at the end of the first full year Maersk can earn an additional 13,500,000€. In the following years, since all brands will be fully integrated in every quarter, the expected annual income from the extra brands integrated in the first year is 36,000,000€.

6.2.3 Assumptions Analysis

In order to understand the reliability of the values estimated, it is important to analyze the validity of the assumptions which sustain them.

Regarding the support resources, the factors considered are drawn from the real situation of the company, meaning that no assumptions were made that could compromise the estimate. However, that is not the case for the capacity to onboard customers. In fact, the prediction for that dimension is merely extrapolated from the current situation, which may not correspond to the future reality.

Firstly, it was assumed that every integration lasted precisely a quarter of a year. In reality, that may not be the case, since some brands are able to complete the onboarding in only two months, while others take up to four months. However, it is unlikely that that variation influences considerably the final value, as the money lost on the slower clients ends up being balanced by the money gained on the faster customers.

The second assumption stated that a brand only generates income when fully integrated. This is not an exact representation of the reality, since customers start running their operation still during the integration stage. However, the volume of operations during that time is extremely reduced compared to the situation after the onboarding is finished. As so, the impact of this disparity also does not disrupt the estimation performed.

A further assumption was that Maersk was able to attract customers, in a way that the onboarding was always at full capacity. Such presumption is actually corroborated by the current situation. Given the status and dimension of Maersk, its services are widely sought. However, since the current capacity only allows five brands to be integrated at a time, many of those potential customers

had to be temporarily rejected. As so, if the available capacity increased, these brands who are on the waiting list could be accepted sooner.

Finally, it was estimated that three more brands could be integrated if the solution was implemented. This prediction is based on the work that the brands which are currently being integrated require. However, since it is not possible to measure the full extent of the implementation of the solution yet, that value may not be exact. As each brand has a major impact on the earnings received, it is essential to understand how the uncertainty regarding the number of additional customers influences the monetary impact of the feature developed. Table 6.3 addresses that issue, presenting the estimated income for different capacity increases.

Table 6.3: Additional income obtained with the implementation depending on estimated extra capacity

Number of Additional Brands	1st Year Income	Following Years Annual Income
1	4,500,000 €	12,000,000 €
2	9,000,000 €	24,000,000 €
3	13,500,000 €	36,000,000 €
4	18,000,000 €	48,000,000 €
5	22,500,000 €	60,000,000 €

Chapter 7

Conclusion

The work described in this dissertation addressed the relevance of product management in the development of technological goods, more specifically of an API. The context of the project also meant studying how to successfully adapt that product from the reality of a start-up (HUUB) to the world of an established global company (Maersk).

The first objective was to map out a plan of initiatives to explore across six months. For this, it was necessary to understand the various requirements and customer needs to successfully identify opportunities to add value to the API as a product. Afterwards, relying on that plan, the aim was to explore in detail the most urgent initiative and develop an appropriate solution. The methodology followed to reach a solution meant that first a final design containing every relevant feature to tackle the problem was idealized and, deriving from that, an MVP implementation to promptly address the issue at hand was developed.

The following sections will summarize the main outcomes produced by the project and reflect on further work to be done.

7.1 Main Outcomes

Regarding the initial goal, a concrete plan concerning the development of the API in study as a product was delineated. This maps out the overall direction of the product and provides a visual guide for all stakeholders to be aligned with the purpose of the API and the reasons why specific features should be developed. More than this, a backlog containing opportunities to be addressed in the future was constructed. As so, the roadmap can be continuously fed with new initiatives each time an epic is completed.

It is important to mention that the identification of opportunities relied on the API as a product approach explored in Section 2.2, meaning that their implementation would boost the UX in many identifiable dimensions such as: API usability, power, and even onboarding experience.

When it comes to the exploration of a specific opportunity, several outcomes were achieved. The first was the design of a solution to provide visibility over integration errors. The process of reaching a solution was strongly centered on approaching that functionality as a product. This

means that the underlying methodology was trying to prove the validity of the initiative with the least effort possible. This provided diverse simple solutions such as connecting a query to the company's helpdesk service or giving the information via database connections with excel. However, the implementation of these ideas was not feasible. The main difficulty lied in the data being separated in both a relational and a non-relational database. As a consequence, the handling of the information needed to be performed with different languages and later adapted in order to be processed together. The chosen solution was based on a Python script and was able to overcome this setback, opening the door for more projects in the company using data from distinct types of database.

For the company, as of now the emails composed by the developed script are the unique source of visibility over integration errors available for the teams. Given that, as stated previously, all brands are currently onboarding the platform and, as discussed in Section 5.1, that is the most critical stage in the client life-cycle concerning integration errors, the solution developed assumes a role of great importance within the company. In fact, the implementation is already being used to report issues to the customers daily. An extremely relevant outcome of automatically generating this report is that Maersk's support and implementation teams are now dealing with integration errors in a preventive manner instead of only being reactive as before. Evidently, this is a way to avoid friction with discontent customers, as the information is transmitted before a complaint is made.

Finally, it was also designed a mock-up of a north star solution. For this it was not only necessary to think in terms of the UX but also the user interface (UI). In fact, it was required to develop a front-end design for a web solution where the visibility was provided in the E-Fulfilment platform. Evidently, fully developing such an initiative would be impossible in the time of this project. Moreover, since it was urgent to implement a solution, it would not make sense to develop straight away a very complex solution when a simpler one could be worked out much quicker and start delivering value to customers sooner. As so, the remaining features depicted in the north star solution should be properly prioritized and planned for future rollouts regarding this opportunity. In other words, the outcome of the north star design is a visual guide for the future improvements of the feature implemented.

7.2 Next Steps

The very nature of product management implies a continuous work throughout the entire life cycle of a product. As so, it is only natural that during the execution of this project further opportunities to complement the work were identified.

Firstly, even though a backlog which can feed the API product roadmap for a considerable period of time was created, a continuous generative research of improvement opportunities should be carried out. In fact, requirements that possibly did not exist or were not relevant during the identification of opportunities conducted in the scope of this dissertation may become crucial at

some point in the future. By engaging in consistent product discovery, the company can be more aware of customer needs and manage their product to be more suitable for the users.

Along with product discovery, a recurrent analysis of the initiatives stored in the product backlog should be performed. Since the circumstances surrounding a product are not static, the initial factors which were the basis of prioritization of an item may change, e.g., an unrelated development greatly decreased the necessary effort to explore an opportunity, consequently increasing its priority. After re-estimating priorities, it is fundamental to compare the new score with the items on the roadmap and, if need be, make the appropriate replacements. In essence, to maintain the validity of the product roadmap developed, the subsequent step of the project would be applying change management to the work produced.

When it comes to the solution developed, as soon as it is available, data regarding its real performance should be collected. This will not only allow to verify the precision of the impact estimation but also provide valuable information regarding which aspects of the implementation should be improved first.

Other angle to work on is approximating the MVP solution to the north star designed. Even if the feature implemented solves the main issue of visibility over integration errors, it obviously does not provide as much value to the customers as the final design created would. For example, the first rollout deployed does not address the restriction of the information being updated and available to the clients at all times. Applying such a feature would mean that the brands would not have to wait for an email to know their integration issues, considerably cutting down their onboarding period and escalating Maersk's income. Nevertheless, just as every other initiative, further improvements to the implementation should be prioritized and compared with the remaining opportunities on the backlog. This ensures that all effort deployed to improve the API as a product is aligned with the company objectives.

Bibliography

- Alphaliner (2022). Alphaliner top 100. <https://alphaliner.axsmarine.com/PublicTop100/>. (Accessed on 05-05-2022).
- Berander, P. and Andrews, A. (2005). *Requirements prioritization*, pages 69–94. Springer.
- Berners-Lee, T., Fielding, R., and Masinter, L. (2005). Uniform resource identifier (URI): Generic syntax. Technical report, RFC 3986.
- Ebert, C. and Brinkkemper, S. (2014). Software product management - an industry evaluation. *Journal of Systems and Software*, 95:10–18.
- Fernandes, H. (2021). Why is product management important? *The Product Manager*.
- Fielding, R. T. (2000). *REST: architectural styles and the design of network-based software architectures*. PhD thesis, University of California.
- Fradin, D. (2019). The importance of product success management. *Spice Catalyst*.
- Gorchels, L. (2006). *The Product Manager’s Handbook: The Complete Product Management Resource*. McGraw-Hill.
- Iyengar, K., Lau, L., Ramadath, S., and Sohoni, V. (2018). The seven make-or-break API challenges CIOs need to address. *Digital/McKinsey: Insights*.
- Joh, J. M. and Mayfield, M. (2009). The discipline of product discovery: Identifying breakthrough business opportunities. *Journal of Business Strategy*, 30(2-3):70 – 77.
- Khan, M. W. and Abbasi, E. (2015). Differentiating parameters for selecting simple object access protocol (SOAP) vs. representational state transfer (REST) based architecture. *Journal of Advances in Computer Networks*, 3(1):63–6.
- Leffingwell, D. (2011). *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise*. Agile Software Development Series. Addison-Wesley Professional, 1 edition.
- Lombardo, C. T., McCarthy, B., Ryan, E., and Connors, M. (2017). *Product roadmaps relaunched: how to set direction while embracing uncertainty*. O’Reilly Media, Inc.
- Malvik, S. (2022). *Mastering Azure API Management: A Practical Approach to Designing and Implementing an API-Centric Enterprise Architecture*. Apress, 1st edition.
- Massé, M. (2011). *REST API Design Rulebook: Designing Consistent Restful Web Service Interfaces*. O’Reilly Media, Inc.

- McBride, S. (2018). RICE: Simple prioritization for product managers. *Intercom*.
- Medjaoui, M., Wilde, E., Mitra, R., and Amundsen, M. (2021). *Continuous API management*. O'Reilly Media, Inc.
- Munch, J., Trieflinger, S., Bogazkoy, E., Eisler, P., Roling, B., and Schneider, J. (2020a). Product roadmap formats for an uncertain future: A grey literature review. In *Proceedings - 46th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2020*, pages 284–291.
- Munch, J., Trieflinger, S., and Heisler, B. (2020b). Product discovery - building the right things: Insights from a grey literature review. In *Proceedings - 2020 IEEE International Conference on Engineering, Technology and Innovation, ICE/ITMC 2020*.
- Myers, B. A. and Stylos, J. (2016). Improving API usability. *Communications of the ACM*, 59(6):62–69.
- Münch, J., Trieflinger, S., and Lang, D. (2019a). Deep: The product roadmap maturity model. In *IWSiB 2019 - Proceedings of the 2nd ACM SIGSOFT International Workshop on Software-Intensive Business: Start-ups, Platforms, and Ecosystems, co-located with ESEC/FSE 2019*, pages 19–24.
- Münch, J., Trieflinger, S., and Lang, D. (2019b). Product roadmap - from vision to reality: A systematic literature review. In *Proceedings - 2019 IEEE International Conference on Engineering, Technology and Innovation, ICE/ITMC 2019*.
- Palma, F., Gonzalez-Huerta, J., Moha, N., Guéhéneuc, Y.-G., and Tremblay, G. (2015). Are RESTful APIs Well-designed? Detection of their Linguistic (Anti)Patterns. In *International Conference on Service-Oriented Computing*, pages 171–187. Springer.
- Patni, S. (2017). *Pro RESTful APIs*. Springer.
- Pautasso, C. (2014). RESTful Web Services: Principles, Patterns, Emerging Technologies. In *Web Services Foundations*, pages 31–51. Springer.
- Pohl, K., Böckle, G., and Van Der Linden, F. (2005). *Software product line engineering: Foundations, principles, and techniques*. Software Product Line Engineering: Foundations, Principles, and Techniques. Springer.
- Richardson, L. and Ruby, S. (2008). *RESTful web services*. O'Reilly Media, Inc.
- Rodríguez, C., Baez, M., Daniel, F., Casati, F., Trabucco, J. C., Canali, L., and Percannella, G. (2016). REST APIs: A large-scale analysis of compliance with principles and best practices. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9671, pages 21–39.
- Shnier, M. (1996). *Dictionary of PC Hardware and Data Communications Terms*. O'Reilly Media, Inc., Sebastopol, CA.
- Soni, A. and Ranga, V. (2019). API features individualizing of web services: REST and SOAP. *International Journal of Innovative Technology and Exploring Engineering*, 8(9 Special Issue):664–671.

- Torres, T. (2021). *Continuous Discovery Habits: Discover Products that Create Customer Value and Business Value*. Product Talk LLC.
- Trieflinger, S., Munch, J., Bogazkoy, E., Eissler, P., Schneider, J., and Roling, B. (2021). How to prioritize your product roadmap when everything feels important: A grey literature review. In *2021 IEEE International Conference on Engineering, Technology and Innovation, ICE/ITMC 2021 - Proceedings*.

Appendix A

E-Fulfilment Objects

A.1 Products

Table A.1: Product Model Fields in E-Fulfilment

Field	Mandatory
Product Type	Yes
Product Family	Yes
Product Subfamily	No
Supplier Name	Yes
Season	No
Product Name	Yes
Currency	Yes
Variants ¹	(See Table A.2)

¹ A product model can have multiple variants.

Table A.2: Product Variant Fields in E-Fulfilment

Field	Mandatory
Description	Yes
Colour	No
Size	No
Reference (SKU)	Yes
EAN	Yes
Country of Origin	Yes
Gender	No
Age Group	No
HS Code	Yes
Material	Yes
Fabric	No
Weight	Yes
Width	Yes
Height	Yes
Depth	Yes
Retail Price	No
Wholesale Price	No
Cost	No

A.2 Purchase Orders

Table A.3: Purchase Order Fields in E-Fulfilment

Field	Mandatory
PO Reference	Yes
PO Creation Date	Yes
Product Subfamily	No
Supplier	(See Table A.2)
Items ¹	(See Table A.5)

¹ A PO can have multiple items.

Table A.4: Supplier Fields in E-Fulfilment

Field	Mandatory
Name	Yes
Email	Yes
Phone	Yes
Default Currency	Yes
Tax Number	Yes
Address	Yes
Zip Code	Yes
City	Yes
Country	Yes
Return Contact	Yes

Table A.5: PO Items Fields in E-Fulfilment

Field	Mandatory
Reference (SKU)	Yes
EAN	Yes
Expected Quantity	Yes
Expected Reception Date	Yes

A.3 Sales Order Deliveries

Table A.6: Sales Order Delivery Fields in E-Fulfilment

Field	Mandatory
SO Reference	Yes
SO Creation Date	Yes
Delivery Reference	Yes
Estimated Delivery Date	Yes
Delivery Creation Date	Yes
Currency	Yes
Service Level	Yes
Preselected Warehouse	Yes
Customer	(See Table A.7)
Shipping Address	(See Table A.8)
Items ¹	(See Table A.9)
Documentation	No

¹ A SO Delivery can have multiple items.

Table A.7: Customer Fields in E-Fulfilment

Field	Mandatory
Name	Yes
Contact Person	Yes
Email	Yes
Phone	Yes
Default Currency	Yes
Tax Number	Yes

Table A.8: Shipping Address Fields in E-Fulfilment

Field	Mandatory
Address	Yes
Zip Code	Yes
City	Yes
Country	Yes
State	No
Name of recipient	Yes

Table A.9: SO Delivery Items Fields in E-Fulfilment

Field	Mandatory
Reference (SKU)	Yes
EAN	Yes
Quantity	Yes
Unit Base Price	Yes
Unit VAT Price	Yes

Appendix B

North Star Solution Pages

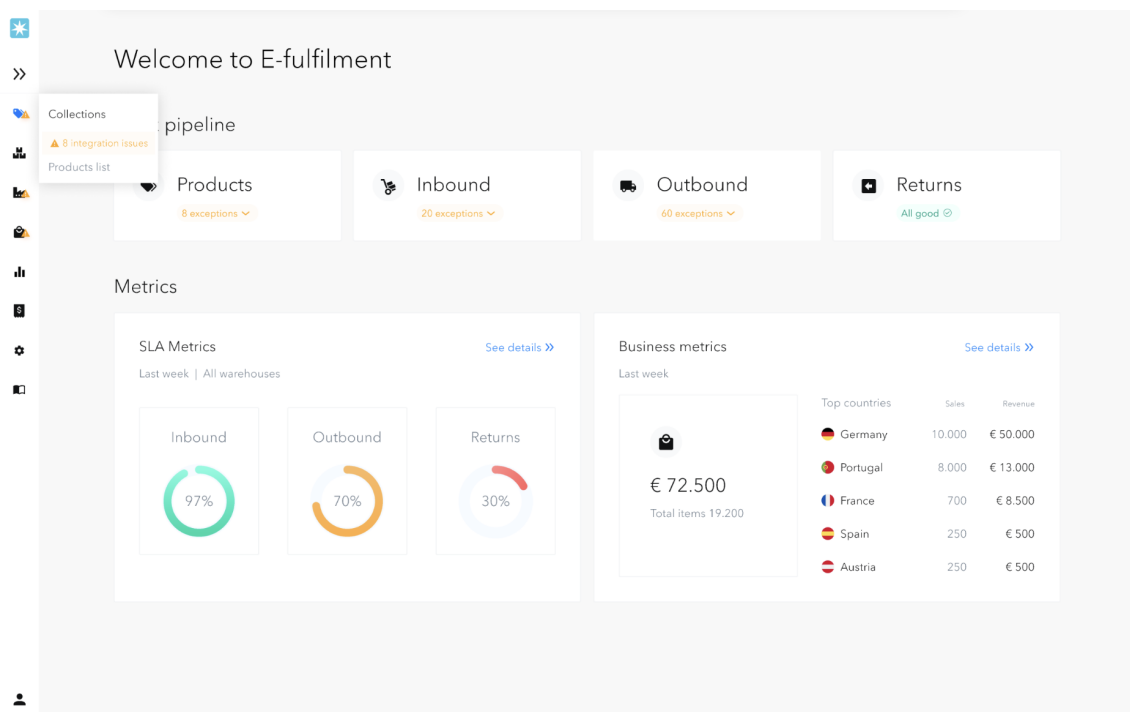


Figure B.1: Mock-up of page displaying initial dashboard and menu notifications

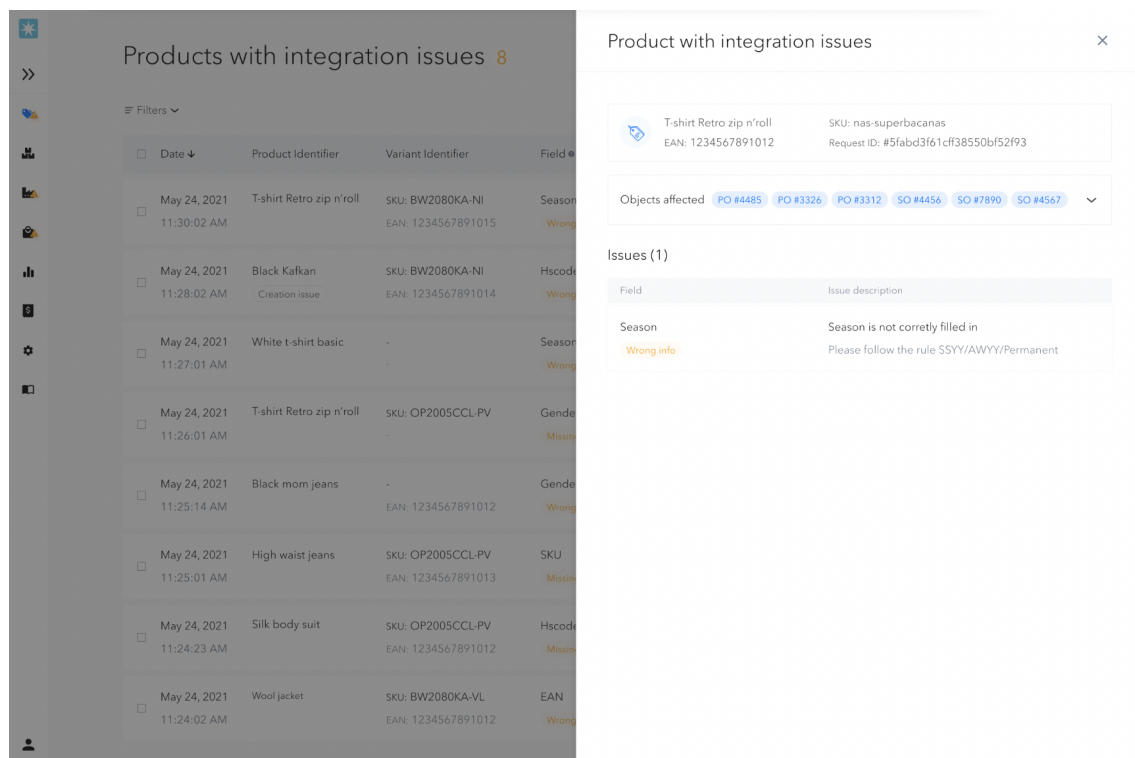


Figure B.2: Mock-up of side panel in product integration errors page

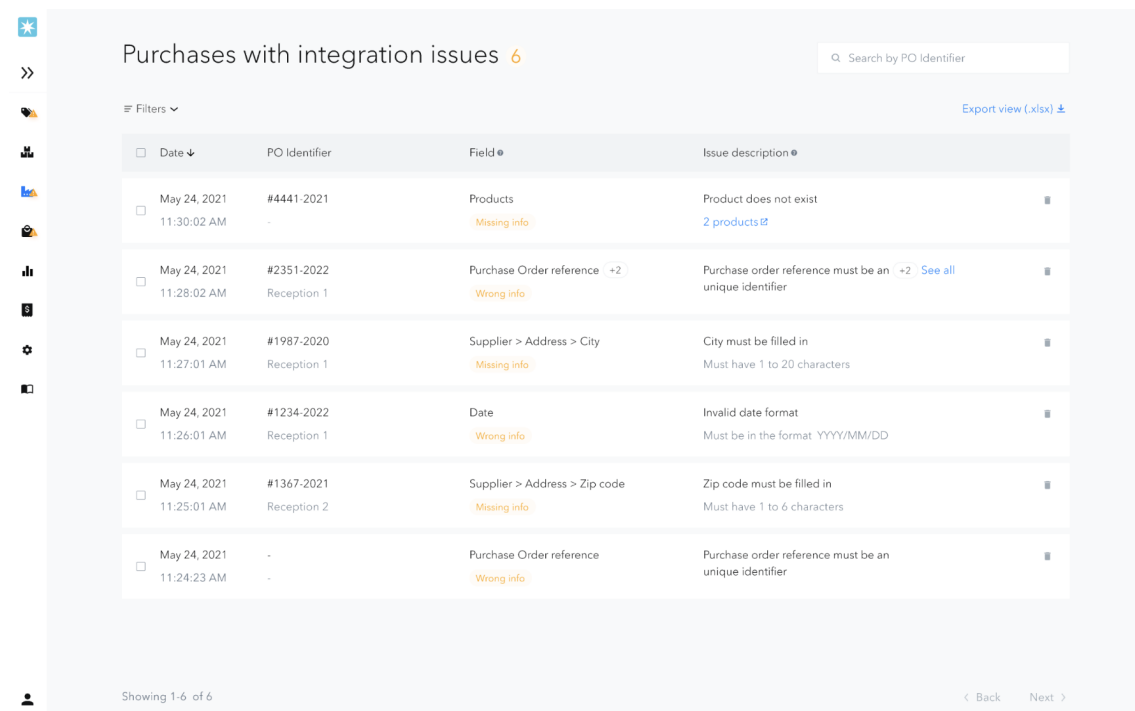
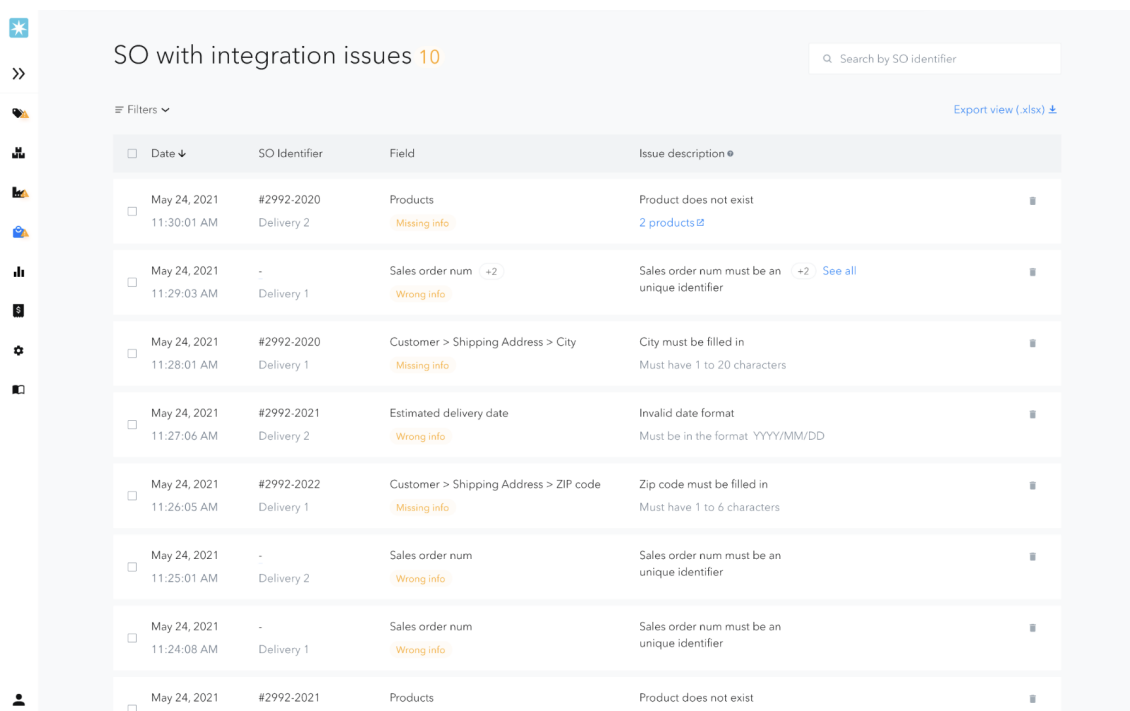


Figure B.3: Mock-up of page displaying purchase order integration errors



SO with integration issues 10

Search by SO identifier

Filters

Export view (.xlsx)

☐ Date	SO Identifier	Field	Issue description
☐ May 24, 2021 11:30:01 AM	#2992-2020 Delivery 2	Products Missing info	Product does not exist 2 products
☐ May 24, 2021 11:29:03 AM	- Delivery 1	Sales order num +2 Wrong info	Sales order num must be an unique identifier +2 See all
☐ May 24, 2021 11:28:01 AM	#2992-2020 Delivery 1	Customer > Shipping Address > City Missing info	City must be filled in Must have 1 to 20 characters
☐ May 24, 2021 11:27:06 AM	#2992-2021 Delivery 2	Estimated delivery date Wrong info	Invalid date format Must be in the format YYYY/MM/DD
☐ May 24, 2021 11:26:05 AM	#2992-2022 Delivery 1	Customer > Shipping Address > ZIP code Missing info	Zip code must be filled in Must have 1 to 6 characters
☐ May 24, 2021 11:25:01 AM	- Delivery 2	Sales order num Wrong info	Sales order num must be an unique identifier
☐ May 24, 2021 11:24:08 AM	- Delivery 1	Sales order num Wrong info	Sales order num must be an unique identifier
☐ May 24, 2021 11:23:01 AM	#2992-2021 Delivery 1	Products Wrong info	Product does not exist 2 products

Figure B.4: Mock-up of page displaying sales order integration errors

Appendix C

Solution Algorithm Flowchart

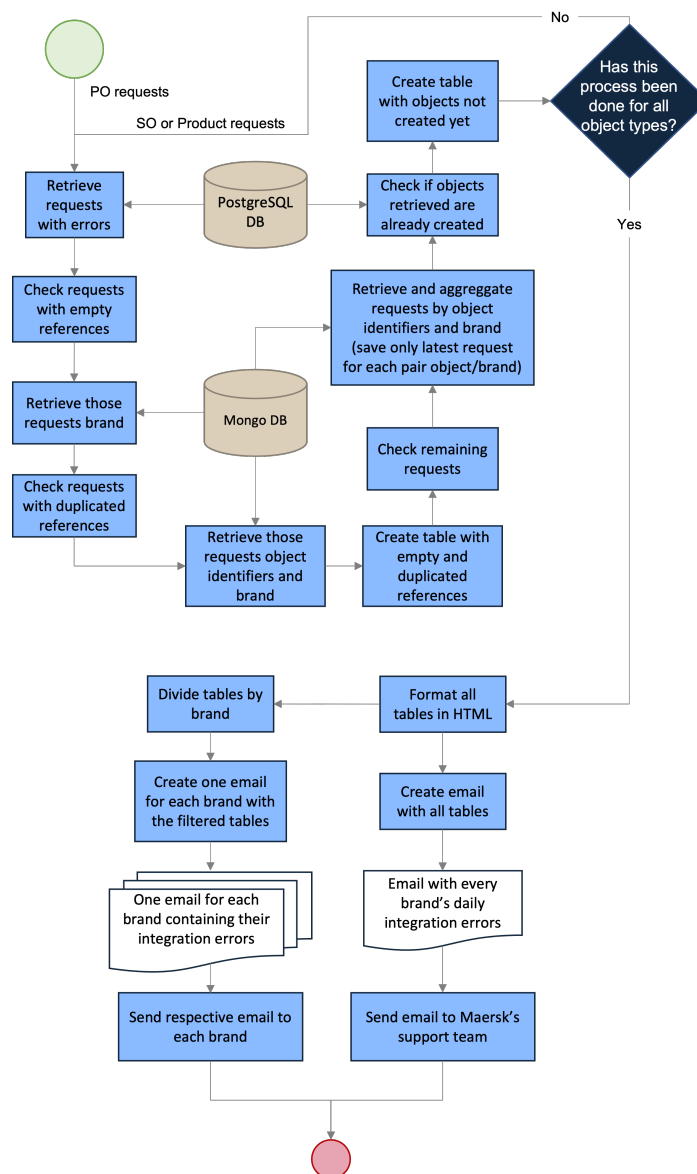


Figure C.1: Flowchart describing the solution algorithm