CSE2000 - Software Project Report

Group 11A

Lucian Negru
Radu Constantinescu
Kendra Sartori
Ahmed Ibrahim
Manar Al-Robayi

TA: Oskar Lorek
Coach: Thomas Overklift
Client: Unetiq BV

Delft University of Technology

June 2022

Preface

This report was written by a group of five computer science students at the Delft University of Technology. In the fourth quarter of our second year, we created software for processing user orders in online shops as part of the Software Project course.

This report is intended for readers with prior knowledge of both Frontend and Backend technologies of applications, as well as basic knowledge of programming terminologies.

Readers that are particularly interested in the topic of eCommerce, specifically online orders, and the conducted research can find all the information in Chapter 2. Users that have more interest in the implementation and outcome of our product may wish to read Chapter 6 and Chapter 7 respectively.

We would like to thank our project coach Thomas Overklift and our TA Oskar Lorek for their help and guidance throughout the course of the project. We would especially like to thank Stefan Reuther and William Bos at Unetiq BV for selecting us for this project, putting their trust in us as a team, and making the result possible.

Delft, 7 June 2022
Lucian Negru
Kendra Sartori
Radu Constantinescu
Ahmed Ibrahim
Manar Al-Robayi

Summary

Due to the global rise of eCommerce many businesses offer their customers the ability to shop online, so much so that the market is projected to have a compound annual growth rate of 14.7% from 2020 through 2027. Such a rapidly growing market is not without its problems, specifically in the field of manufacturing.

A large issue with ordering large quantities of products online is the process of searching for them and adding each one to the basket; in the case of manufacturing businesses these products are often similar in name and appearance, making the entire process long and tedious.

This report aims to showcase the solution our team developed to combat this problem, in the form of an Order Processing System (OPS). The team at Unetiq BV tasked us with designing this application from the ground up while allowing for future development to be easily continued by others in the future. This is an application which allows users to upload files of products they wish to order, then the products get automatically added to the basket. The research was conducted on the impact of eCommerce and similar systems to design an effective and value-sensitive product.

The clients then explained to us what they were looking for in this system over the course of multiple meetings. These were functional as well as non-functional requirements which we organised in order of importance using the MoSCoW method. The requirements were useful for determining the feasibility and risks of the project as well as getting started with the design.

We designed the project into Frontend, Backend and Database, and assigned ourselves various tasks in these categories. The Frontend was implemented using the React library of JavaScript, as per one of the requirements. The Backend was made using the Django framework in Python, allowing connection between the Frontend and the Database. Lastly, the database uses PostgreSQL, offering easy integration with Django.

The last step of the design was to look at the ethical values we have to consider, there were two main ones which apply to the project. The first is regarding the management and transparency of user data, specifically the files they upload into the system. These files get removed from the system after processing the order and the processing algorithm will be easily available to other developers wishing to implement it in their shops. The second value to consider was regarding artificial intelligence. The system uses a neural network to process the uploaded files before querying, there is a concern when it comes to accountability and verification when dealing with such technologies. That is why we offer the possibility for the user to edit any decision the processing makes in the basket, as well as document the implementation of the neural network.

The implementation allows users to perform the actions that the client wanted, delivering a satisfying product. The Frontend consists of three pages: the Home Page, the Order Page and the Basket Page. On the Home Page, the user may search for products directly or add files for processing, taking them to the Order Page. The Order Page shows the user what products were retrieved given the uploaded files, these can be reviewed and edited. Afterwards, the products can be added to the basket on the Basket Page. This is where all products from all processes - as well as manual searches - appear and can be edited.

The Backend features the algorithm for processing the uploaded files. The files are run through a trained neural network to label the fields. Since some files may not indicate what the data represents, we use the AI to accurately label it. Afterwards, the data forms are queried from the database and returned to the Frontend to be displayed.

The implementation contains full testing of the Frontend components as well as the Backend functions. There are also functions for testing the accuracy of the product given test data from the client.

We believe that the product delivered satisfies the requirements of the client as well as challenged us to work together as a team to implement something that we have never done before. The product demonstrated that the process of ordering online can be greatly improved by implementing a system which automatically fills the basket. In that regard, it has managed to meet the aim mentioned previously.

Contents

Chapter 1

Introduction: What is Order Processing?

Globalisation brings a vast array of advantages for businesses and markets worldwide. These benefits include cultural exchange, increased capital flow, and better-valued products, which all can determine the success of a business [1]. One main area of globalisation which suffers from some problems is eCommerce. Online shops are not usually designed to allow for large order lists, instead, they require the customer to manually search and add each article. Imagine having to order large quantities of small products, often with complicated names. This process can be extremely tedious and long for manufacturers, which often order small components in bulk. Unetiq [1] has started working on a project to address this issue, allowing customers of online shops to place their orders more efficiently by automatically determining which products must be placed in the basket. This will help the user by saving time with orders and help the online shop employees when processing the order. We have been assigned to expand on their work by introducing new functionality to the Furning [2] website.

This report aims to showcase how the Order Processing System (OPS) helps businesses have a better online shopping experience for their customers. Unetiq conducted some research to determine how customers of online shops frequently place their orders. One of the main ways customers store their orders is in Excel documents, therefore we were tasked with implementing a method for retrieving the files, processing them into ordered datasets and creating the basket for the user. Another type of document often used is PDF, usually in the form of invoices or lists. These PDFs needed to be processed with the use of machine learning to determine what the information dictates. We were given samples of orders compiled by the client, this helped us decide what we should research and implement. One challenging aspect of this feature was detecting the information stored in faulty data, such as mistyped words and missing fields. Lastly, the whole application had to be presented in an easy-to-use interface using React JS [3].

[1] https://www.unetiq.com/
[2] https://www.furning.com/
[3] https://reactjs.org/

The report will be presented in the following structure. Chapter 2 will delve into analysing the problem and discussions on the conducted research. Chapter 3 will include the requirements and use cases of the project. Chapter 4 will explore the feasibility, risk and architecture of the project. The result of the project can be found in Chapter 5, along with the specific implementations of each software component. An analysis of the results, such as speed, accuracy, and ease of use can be found in Chapter 6. Chapter 7 will summarise the findings and link them back to the initial research to answer the research question.

Chapter 2

Problem Analysis: Order Processing

To deliver a satisfying product which tackles the issue at hand, we had to first understand the problem. Ordering large quantities of products can be tedious and time-consuming for the customer as well as for those responsible for the processing. Manufacturing businesses are greatly affected by this, as they often order large quantities of small, unnamed and similar products. Because of this, businesses can spend a lot of time having manually processing these large orders. This chapter aims to elaborate on the problem to better understand what needed to be done.

Section 1.1 will describe the four main issues with Order Processing which we have tackled. Section 1.2 will go into detail about the use cases of the system and research the potential impact such a product could have on the market. In Section 1.3 we will discuss the already existing solutions for Order Processing, specifically what they accomplish and what their shortfalls are. Finally, in Section 1.4 we will state the goals of the project.

2.1 The Four Dimensions of Order Processing

The problem we were assigned to solve consists of the following four parts:

1. **Structuring the data from the Excel files**
   Many businesses keep their order lists in Excel documents only to have to manually add everything to the basket when the time comes. This wastes a lot of the time for the person filling out the order when dealing with large lists. The system accepts Excel files, in various levels of structure, and compiles the shopping basket automatically for the user. The challenge was structuring the data when there is faulty or missing information.

2. **Processing the data from the PDF files**
   The system also supports orders in the form of PDF documents as many companies prefer to send invoices of their order list directly to the shop for them to process. The PDF documents can be very unstructured and may have completely different formats from one another. Therefore, manual processing of the order is usually done on the shop's side, again wasting time. We have decided on a solution that can adapt to any of the formats they may have and accurately construct the shopping basket.

3. **Designing the user interface for the frontend of the online shop**
   Many online stores are not well optimised for devices such as phones or tablets, leading to a worse experience for customers which prefer those. For the customer to have a good experience, we have ensured that the interface is friendly and intuitive, as well as compatible with multiple sized displays.

4. **Satisfying the Stakeholders**
   As will be mentioned in the requirements section, the project had multiple stakeholders. One of the biggest priorities was that we satisfy all of them.

2.2 Market Impact of Order Processing

In this section, we evaluate if this was a viable and useful product by determining the factors and context in which it can be successfully used. Afterwards, we will look over the use cases of the system. We asked ourselves questions such as: Is there a demand for such a product in the market? Would this product have a meaningful impact? Is there proof that such a product would benefit the targeted stakeholders? This section will be backed up by empirical research. To accurately retrieve data on this issue, we have looked at the companies that implement a similar product and have conducted and published market research on this.

2.2.1 Research on Order Processing

One of the companies which have researched the problem and developed software for it is Conexiom [1]. Conexiom states that almost half of purchased orders are received as emailed documents and must be manually entered into a system, often this being done by an employee. In a study that Conexiom has conducted [2], they say that annually in the US, 17 trillion dollars are spent in manufacturing and distribution, out of which 50% of it is manually processed. In this study, they also point out additional interesting factors which say that per day a CSR spends on average 2-3 hours on re-entering mistyped orders and that another side effect of the manual transaction is that it is unable to scale when the business is growing.

[1] https://conexiom.com

From another perspective, we looked at another competitor, Palette Software [2], and one of their customer case studies. In this study, we can immediately see the huge impact that their product had on the customer's perspective. Emily Grantham, AP Supervisor at Landstar System inc., says "Before we deployed PaletteInvoice, our invoicing system was labor-intensive and time-consuming for staff. Automated processing has significantly increased our overall efficiency, shortened payment cycles, and helped to improve the manageability of transactions" [3]. This meant that customers do see immediate benefits from automation and that such products would have a positive impact on any organisation.

Moreover, we could see that automation in any part of a process can only provide benefits overall, most of them can be seen when talking about capital, by reducing costs, and time and increasing customer satisfaction by having a more efficient processing system. According to a 2016 Accenture report [3], 52% of the surveyed companies had changed their providers in that year due to poor customer satisfaction. Another big company, Capgemini, said that 69% of the organisations managed to decrease this issue of customer satisfaction by integrating intelligent automation [4], the rest still prefer human interaction or a mix of the two. This meant that automation could be the first step of implementation for any organisation that wishes to tackle this issue.

2.2.2 Use Case Analysis

For this analysis we decided to look over the use cases of the product for the client, this allowed us to give a technical description which takes advantage of the company's technical background.

Two types of users can take advantage of the product, the online shop customer and the purchase order reviewer.

The online shop customer

- Customer wants to order a large list of products
- Customer wants to order an unordered list of products
- Customer wants to efficiently manage and order the set of wished products.
- Customer wants to order a list of products with (possibly) wrong data
- Customer wants to order a list of products with (possibly) empty data
- Customer wants to receive an ordered list from the unordered list of products
- Customer wants to upload the invoice of orders directly to the online shop
- Customer wants to upload multiple PDF formats to the online shop

The online shop employee

- The employee wants to receive an ordered list from the customer
- The employee wants to review only a subset of orders, the rest are automatic

[2] https://www.palettesoftware.com
[3] https://www.accenture.com/us-en

2.3 Already Existing Solutions for Improving Order Processing

The Automated Order Processing System is an eCommerce Automation product. To be able to compare the project to existing products in this category we first have to define what the category is and then look at existing solutions. In this section, we will describe what eCommerce Automation is and discuss three existing products for Order Processing.

2.3.1 eCommerce Automation

eCommerce Automation refers to "the software that helps your online store convert most or all of the manual, repetitive tasks into self-fulfilling, automated tasks." [4] This means that employees can shift their focus to other areas of the business and consumers can rely on a more efficient method of shopping.

Most automation can be described by a Trigger, Condition, Action workflow. Triggers are events initiated by the client directly, in the case of the system it would be the client uploading a file to the online shop interface. Then various conditions can be checked to determine the following action. In our case, the conditions would be the type of processing that should be used depending on the file type, whether the items on the list of orders are available, etc. Lastly, there is the Action stage where the automation process determines what should happen and makes it happen. In the case of the System, that would be the desired items being added to the basket.

eCommerce is one of the world's fastest-growing sectors, shown in Appendix A, with a consensus that it helps businesses thrive and improves customer experience [5]. Therefore, we find it important to compare those factors between the project and existing solutions.

2.3.2 Conexiom

Description: The conexiom software [5] offers processing of orders through the use of machine learning to save time. They do not disclose more information on the use of AI other than to identify data fields. It receives different file types of unstructured data and runs them through a processing algorithm to label, structure, and identify the content. Then it forwards the extracted data to the system for various tasks for every unique vendor.

Shortfalls: The software doesn't allow for files to be sent through the user interface, only via emails. The system aims to improve that by allowing customers to add the files to the online shop directly through the interface and keep things simpler.

[4] https://www.bigcommerce.com/ecommerce-answers/what-is-ecommerce-automation/
[5] https://conexiom.com/sales/

2.3.3 Tipalti Approve

Description: Tipalti Approve [6] is a platform that allows for PO (purchase order) generation by having users upload invoices in multiple formats and levels of structure, then processing the information and display it back to the user. The returned data can then be ordered or further analysed. The exact process of what happens in the background is not disclosed, unfortunately.

Shortfalls: The user has to upload the invoices to the app, which then sends an order. The implementation will streamline this by allowing the users to upload directly to any website using the system. Therefore, the platform has to support the different online shops and introduces a middle-man in the ordering process.

2.3.4 Palette Automated PO Software

Description: The Automated Po Software [7] is software that can be implemented into a business either stand-alone or in an ERP (Enterprise Resource Planning) system. The system receives the orders either through emails or scans and matches them to the recipient store. If a match is not met then it is forwarded through to a manual pipeline where an employee will handle it. The matches are then forwarded to the ERP where the data can be used for various actions, including ordering.

Shortfalls: The POs received have to be rather structured and the software does not make much use of machine learning for more complex and unstructured files. The processing consists of parsing rather than adapting, forcing the users to conform to a certain structure of invoice and compromising on user experience.

2.3.5 Research Summary

In conclusion, there exist products that implement similar functionality to the one proposed by the client. There are even products, such as Palette Automated PO Software, which contains most of the requirements. However, we have not found any to completely satisfy all of the requirements, this leads us to conclude that implementation of the Automated Order Processing System was necessary.

[6] https://www.approve.com/po-management-lp/
[7] https://www.palettesoftware.com/resources/purchase-order-automation

2.4 The Two Main Goals of The Project

The Automated Order Processing System was a project already started, in part, by Unetiq. The main goal was to implement new functionalities to the project in such a way that they can be integrated into the existing system and be easily expanded upon. Since we started the functionalities from scratch, we aimed to be efficient with our time and took great consideration in our choices as they would have affected the continuation of our's as well as Unetiq's work. The project goals were as follows:

1. **Develop the System which processes orders**
   Processing of orders from Excel documents must be implemented and functional by the end of the course. Processing from PDF documents is the next step of the project and is not expected to be finished by the end of the course, however, what we do should be well documented so that developers can work on it in the future.

2. **Scalable implementation**
   Everything we implemented has been well documented and all research made is readily available for future developers to continue to expand the project. We aimed for the features to be easily integrated with the project as a whole by the end of the course.

Chapter 3

Requirements Engineering for the Order Processing System

To fully understand the goal of the project and the wishes of the client we will use 'Requirements Engineering'. Requirements Engineering is the process used to recognise and express the software specifications that allow us to solve application problems [6]. It makes it possible to translate the client's needs and identify incomplete or inaccurate specifications that potential users may have. The following subsections discuss the various methods used to determine these specifications.

3.1 Stakeholders Involved

To identify all the requirements for the project, we first needed to know who the stakeholders are. Stakeholders are the parties that are affected by a development project [7]. This project involved three different parties, each with different requirements for the application. The following subsections discuss each stakeholder and their goals.

Unetiq BV

Unetiq BV were the main stakeholders and provided us with the necessary information to complete the project. Unetiq is a company that develops and customises AI software for various other companies to accommodate the automation of manual processes [8]. Their goal was to create a web application that allows customers of online shops to upload their orders through unstructured Excel and PDF files. The orders are processed in such a way that no human interaction is needed to complete an order. The application scans the uploaded file and extracts a structured order that can then be processed further.

Customers

There are two types of customers that we have identified as stakeholders. The first ones were the customers of Unetiq, they will be discussed in the next subsection. The second type of customer was the customers of online shops. The goal of these customers is to be able to place an order by uploading an Excel/PDF file, instead of going through online shopping and adding each product one by one into a virtual basket.

Online Shops

Online shops are the customers of Unetiq. In this report, we use the term 'online shops' as a term for identifying online shops that want to automate the process of handling orders through Excel and PDF files. The goal of this stakeholder was to allow their customers to upload Excel/PDF files to place their orders. This has been realised by integrating an option to upload Excel/PDF orders on their online shop.

3.2 Requirements Elicitation

There are several types and techniques for elicitation that allow developers to identify all requirements. The two types of elicitation techniques are:

- Direct Approach
- Indirect Approach

Indirect approaches are used to obtain information that could be challenging to extract and articulate. Direct approaches focus on understanding the problem that the client faces and the way they think this problem can be solved [9].

To collect the requirements for the project we have used a direct approach, namely the 'interview' elicitation technique, by meeting with Unetiq. These meetings have also provided us with a clear understanding of the requirements of the other stakeholders.

The interview was unstructured. The questions were formulated on spot during the interview to allow as much flexibility as possible.

The answers to these questions gave us a clear and adequate understanding of the requirements that had to be formed. The requirements were then split into two groups; functional and non-functional requirements. Functional requirements focus on the ability of the system to perform certain tasks, while non-functional requirements focus on the behavioural aspect of the system [10].

The list of all requirements can be found in Appendix D.

3.3 Definition of Requirements for the Order Processing System

The requirements in Appendix D are divided into four categories. The categorisation was done using the MoSCoW technique. This categorisation allowed the work to be sustainable and efficient. One of the key factors of MoSCoW categorisation is that it defines the priorities for each requirement [11].

The requirements are divided into four sections:

- **Must Haves:** The implementation of these requirements gave us a minimal viable product on top of which all other requirements are built.
- **Should Haves:** This category includes all requirements that were not vital for a system to work, but were of such importance that they need to be implemented.
- **Could Haves:** These were features that offered improvements to the product but were not necessary for the deliverable.
- **Won't Have:** These were requirements that were seen as infeasible to implement within the development team.

3.4 In-Depth Description of the Must-Haves

To understand the project better, it is good to take the requirements into account. The most important requirements were the Must-Haves as these requirements had to be completed. To highlight these must-haves a description is given. The Must-Haves of this project:

- As a customer I want to be able to search for a product on a search bar. This must-have improves the usability of the product. This must-have is important as without the search bar, a customer cannot search for a product and therefore may not be able to complete his order. For this must-have to be completed, a search bar must be implemented that is easy to use.
- As a customer I want to get a list of products that match my search keys. This must-have is important as not receiving a list of products for the customer makes the product less practical. This must-have improves the functionality of the product. For this must-have to be completed the product must be able to match products to certain search keys that are given.
- As a customer I want to be able to upload an excel file with an order. This must-have improves the usability of the product. This must-have is important because without the ability to upload an excel file with an order, the customer can not upload his order and the product becomes unusable. For this must-have to be completed a button is required on which the user can click to upload an Excel file that contains an order.

- As a customer I want to get an 'automatically created structured order' after uploading my order in an Excel file. This must-have improves the usability of the product. This must-have is important because otherwise, the user has to structure his order which is not practical. For this must-have to be completed the previous must-have about the button for uploading a file must firstly be finished. After having a button for uploading an Excel file, the product must be able to transform this order automatically to a structured order. This must-have can be completed by adding the functionality of transforming an Excel file into a structured order.

- The system will filter out unnecessary information from the excel orders. This must-have improves the functionality of the product. This is an important must-have as many orders contain unnecessary information that is not needed for the process. Filtering this information out ensures that less memory is needed and the process requires less time to be finished. For this must-have to be completed a function must be implemented that takes an order with necessary and unnecessary information as an input and gives an order with only necessary information as an output. The function must therefore recognize the difference between necessary and unnecessary information. It must also be able to filter unnecessary information.

- The system will map the column names for each column in an order. This must-have improves the functionality of the product. This must-have is important as it is an important part of processing the orders. For this must-have to be completed a function must be implemented that takes an order without mapping as an input and gives an order with this type of mapping as an output. To do this the function must be able to map the column's names and do this for each column in an order.

- The system will map the types for each column in an order. This must-have improves the functionality of the product. This must-have is important as it is an important part of processing the orders. For this must-have to be completed a function must be implemented that takes an order without mapping the types as an input and gives an order with this type of mapping as an output. The function must therefore be able to map the types for each column in an order.

- The system will convert a filtered order into a structured Python datatype. This must-have improves the functionality of the product. This must-have is important as a structured Python datatype is needed for further implementation. For this must-have to be completed a function must be implemented that takes a filtered order as an input and gives a structured Python datatype as an output.

Chapter 4

Design

In the following chapter, we will be looking over the design of the project. Designing is a step we have taken very seriously as it set the roadmap for the entire course duration. We have started by looking over the requirements stated in **Chapter 2** and discussing the overall feasibility in various categories. We have then analysed the risk given these requirements. Lastly, we have made an architectural design based on the knowledge previously stated.

In the following section, we will take a look at the various categories of feasibility, and how realistic we believe this project is. Then, we will take a look at the risks we have to consider when tackling this project. Finally, we will discuss the implementation for the frontend, backend and server, as well as the reasoning for all the choices we made.

4.1 Feasibility Analysis of Order Processing System

This study has been conducted after we had several meetings with the client, in which we discussed exactly the way and setting in which the final product will be used, to properly understand and assess the practicality of the proposed project. The final analysis has been done after considering multiple factors some of which are technical, operational and time-related.

In addition to those factors, we also looked at other factors that will guarantee that this project can be of use to multiple clients, and it can serve as a solution to the recurring problem that a lot of companies are facing as specified in **1.2.1 Research on Order Processing**.

In the following sections, we will talk about all the different types of feasibility components we should take into consideration. These will be technical, operational and time feasibility, as well as other factors. Finally, we will summarise everything to determine the overall feasibility of the project.

Technical Feasibility

In terms of technical feasibility, the team looked at the currently existing technology, and if it suffices our needs. Since this project was built from scratch, we considered that, after thorough research, several options existed in which we could combine currently existing technologies to achieve the final product.

Moreover, we considered that the team has the necessary skills in terms of technical knowledge to accomplish this project. In addition to that, the existence of products that achieve similar functionality to the project ensured that there would not be any technical barriers present.

Operational Feasibility

Regarding operational feasibility, this project matched the business objectives and goals of the company, since their team is also worked on it, however with slight differences in terms of requirements. In addition to this, the integration within the organisation's main focus, was easy since the company is already working on a similar product. Also, the company has provided us with a sample database, and orders, both in the Excel and PDF formats, that we have made use of to test and develop the product.

The difference between the two formats was that the data in Excel was more structured as some columns and rows are used. However, the data in the PDF was unstructured and consisted of many different formats.

Time Feasibility

Time was another constraint that this project was faced with. Given that the project was part of the course provided by the university, we - along with the stakeholders - were aware of the limited time frame in which this product had to be delivered. With this situation, we considered that a prioritisation technique was of utmost importance, therefore we decided to use the MoSCoW method, to set the targets straight.

We considered that, despite this constraint, the team will be able to accomplish this task and in the end at least deliver the minimal viable product; this has been established in the 'Should-Have' section of the MoSCoW analysis.

Other Factors

Other factors that have be taken into consideration were economic and legal feasibility. For the first one, we considered that given the setting in which the project is done, it fell outside of the scope, even if it can be recognised that the impact of such a project would be a positive one on a company. Regarding the legal feasibility, the project has been done under legal requirements since all laws and regulations are met.

Feasibility Summary

In conclusion, since all the aforementioned factors were met, this project was feasible. As a team,it was our responsibility to make sure we considered everything and respected all feasibility factors to deliver a viable product in time.

4.2 Risk Analysis of Order Processing System

It was of utmost importance to assess the risk associated with a project in terms of the different criteria that could influence the development process. Although, many aspects might be considered irrelevant when looking at the bigger picture, including them in the risk analysis has ensured that we took into account issues in an anticipative manner to prevent a failure to deliver a highly qualitative project.

In the following subsections, we will describe the different types of problems that we anticipated encountering throughout the past weeks and their relevance. Furthermore, we will explain the solutions that were feasible and suitable for all group members.

4.2.1 Schedule and communication capabilities

After making a thorough team analysis, also including Belbin's team role management theory [1], we have gained insight into how each individual in the team, but also the team composition might affect the way we work as a group.

The team members' prior experience consists of other projects which are part of the curriculum imposed by the study program. However, the Software Project we have been part of was far more permissive in terms of time allocation and resource management, as it was the first project in which the students had a connection to a real-world client. This could have posed a challenge, whereas the impression of freedom could easily affect keeping a proper schedule and making sure we are on track with the project.

On the other hand, the way that people interact has changed significantly in the past couple of years. In addition to the commodity everyone has working from home, online meetings feel like a much more effective way to discuss further developments, especially given that the group members live in different cities, which makes it harder to meet in person. This could have also affect productivity and morale, leaving people feeling rather lonely than part of a team.

We aimed to minimise the risk of running into these problems by having frequent meetings, at least three times a week while keeping in touch daily, to make sure we are all on track with the work that needs to be done. Furthermore, we planed on meeting at least once a week to work together in-person to create the general feeling of belonging to a team.

4.2.2 Possible growth in requirements

Requirement elicitation was the foundation on top of which the project is built. The requirements dictated all other software engineering processing which could have also influenced productivity, quality and risk [12]. For that reason, we had a clear overview of what we had to do and what the different stakeholders expected from us. However, some of the requirements were probably less obvious and there was always a chance that we could miss some of them, which might have negatively impacted the process and caused unnecessary delays.

[1] https://www.belbin.com/about/belbin-team-roles

In this regard, we believed it was important to have a good relationship with the client and constantly keep them updated regarding the development of the product, through the weekly meetings we have set. Furthermore, we have decided to follow the guidelines described by the agile development framework and structured the process into sprints that lasted one week each.

4.2.3 Lack of experience with React

Our client has expressed their preference for us to make use of React for building a user interface for the final deliverable. However, none of the team members had experience with this JavaScript library, which could have hindered and delayed the process by a significant amount of time, equivalent to the time required for us to have gotten acquainted with it and start making use of it.

We have overcome this lack of expertise by setting a clear deadline by which each team member needed to try getting acquainted with React. Additionally, we hoped that a great collaboration between the members would ensure we are all up for the challenge that creating this project poses.

4.2.4 Lack of knowledge about GDPR and other legal aspects

Given that we were developing a tool which will be, hopefully, widely used by different categories of stakeholders, it was our job to ensure that it was easy to use and did not have any negative impact on these stakeholders. In other words, it was our job to protect the users of the tool from the possible harm that it might do, especially in terms of privacy and security. In this regard, we needed to make sure that it was compliant with the guidelines described by the GDPR directive.

We avoided storing sensitive personal data retrieved from the customer (i.e. the address) anywhere in the database. However, in the eventuality that we needed to store the data to achieve a more accurate result, we would have made sure to comply with the GDPR, making sure that the customers are aware of how their data is being processed as well as deleting the stored data after a certain amount of time.

Given that we were not yet aware of all the legal implications of the project, we wished to carry out further research into the possible dangers that a user could be faced with and get more informed on these matters. Furthermore, we have asked the client for further information about things that we need to take into consideration regarding the legal aspects of the product.

4.3 Architecture of Order Processing System

In the following section, we will describe the research and design for the various components of the project. We will look over the frontend design, the backend design, and how we set up the database.

4.3.1 Frontend

In terms of frontend, we decided to use React [2] since it is one of the most popular frameworks and this can be useful for further implementations on this project. On top of this, using React was one of the non-functional requirements that the client provided us.

React is an open-source library that is built to simplify the building process of advanced user interfaces. The core of the React user interface consists of components which are parts of the user interface, that can be integrated to become a whole functioning user interface which simplifies the UI building process. There are several advantages of using react over regular JavaScript, first, it confines options of coding and so the code is easier to write and also becomes cleaner. Furthermore, the open-source aspect of React and its wide usage make it so that a great number of new tools are released regularly [13].

To have an overview of what the team decided that the implementation would look like, we created a low-fidelity prototype. This prototype served as a boilerplate on which we brainstormed with the client to achieve their desired implementation. The prototype can be found in Appendix C. A low fidelity prototype was helpful in the early stages of development, offering a visualization of alternative design solutions, which provokes innovation and improvement, alongside offering users a more comfortable way of making suggestions [14]. There were two pages created for the application, namely the Home page, the Order page. The Home page lets users upload a file and the order page makes it able for the user to review the processed order.

4.3.2 Backend

Python

For the backend, we agreed on using Python [3] as the main programming language. The choice of choosing Python was one connected to the non-functional requirements that we were provided by the client. There were a few advantages of using Python for the backend of this application.

The biggest advantage of using Python was that it is one of the easiest programming languages. This is caused by the simplicity of Python which not only makes it easier to understand but also makes it easier to improve your programming skills, and it allows for less coding. This simplicity also causes the programmers to focus on the product instead of having to spend a lot of time understanding the programming language.

Python also provided an extensive library which allowed for more possibilities in programming. Another advantage of Python was that it is a very well-known programming language which resulted in lots of tutorials, guides and documentation available online. This way the programmer has to spend less time being stuck and can get help easily online [15].

[2] https://reactjs.org/
[3] https://www.python.org/

Django

We agreed on using the Django [4] framework. The other option for the main framework – that wasn't chosen – was Flask. One of the reasons Django was chosen is that it is multi-threaded while Flask is single-threaded. Another reason for choosing Django is that it has more functionalities that are out-of-the-box and can be used. Further comparisons between the two that were useful to us can be found in Appendix D.

Django uses the Model View Template which is a software design pattern. There are a few advantages to using Django as the main framework. An advantage of Django is that it performs well on security as it offers protection against common security attacks. Django is also a production-ready framework which could be useful if the client decides to push this application to production. The inclusion of templates ensures that beginners, like ourselves, implement the best practices. Django is also a very well-known framework which results in lots of tutorials, guides and documentation available online. This way the programmer has to spend less time being stuck and can get help easily online [16, 17].

4.3.3 Database

In terms of databases, we first decided to use a serverless option, because of the small scale of the product and the additional benefits that it provided. An advantage of using this serverless option was the cost efficiency, with the option to host it on-demand. On-demand refers to paying only for the actual read and writes operations that the application would perform. Hosting the server can be very costly, but managing the server is costly as well, as there is licensing, maintenance, support etc. The scale potential was another reason for using the serverless database as the automatic scalability is based on the workload [18]. The database we chose firstly was DynamoDB, provided by Amazon, a non-relational database which is optimised for intensive read workloads. DynamoDB [5] was planned on as it does not require SQL, it is easier to read and it is not expensive [19].

The team tried to query the orders with DynamoDB, but the primary key was needed to process the queries, which in the case of the orders was not available all of the time. Another downside of using this database was that it then had to scan the whole order and it used pay per query, which would have been too expensive and not feasible. To resolve these complications, another database called Postgres was used. PostgreSQL [6], also known as Postgres, is a well-known relational open-source database. This database was used because of the higher performance in processing the queries. The advantages of Postgres were that it had a better combination with Python, it was well compatible with the framework and it did not require a pay per query. Postgres also performs well on scalability as it supports the technology for it and it performs well on security due to extensibility [20].

[4] https://www.djangoproject.com
[5] https://aws.amazon.com/dynamodb
[6] https://www.postgresql.org/

Chapter 5

Values to Take Into Consideration

As software developers for a project, it is easy for us to focus on the technical requirements and ideas of the project proposed by the client. However, this narrow view impedes us from considering the direct or indirect effect the work could have on other people. When we aim to be responsible and accountable for the work and its effects on others, we have to look at which *Values* we want to achieve, what norms have to be respected, and the technical requirements necessary [21]. One way of designing the product with these ideas in mind is through Value Sensitive Design (VSD).

VSD is "...a theoretically grounded approach to the design of technology that accounts for human values in a principled and comprehensive manner throughout the design process" [22]. In other words, it is the philosophy of considering values for your design and specifying them as technical requirements. This helps the technical requirements shine a light on ethical components and helps guide the developer to being more responsible and accountable.

The chapter aims to identify the ethical categories relevant to the project and explain how we incorporated VSD to mitigate them. The ethical categories will be **Ethics Regarding Data Management and Transparency** and **Ethics Regarding Artificial Intelligence**.

5.1 Ethics Regarding Data Management and Transparency

The software we have designed accepts files – in either Excel or PDF format – from the user and temporarily stores them while running an algorithm. Therefore, we thought it would be appropriate to consider the value of the user's privacy in the design, specifically how we manage data and transparency. The backend also contains data on the products offered through the Furning website, this database is solely queried and not accessible by the user.

The data stored from the user cannot be used to identify a person as it only contains products for an order. The system simply processes the data and compiles a shopping basket. The user manually adds personal information such as name and address when completing the order, this is out of the scope of the system so no such data will be stored. To respect the value of privacy for the users, we opted to not store any data they have submitted after the processing stage. Once the files have been processed and the basket returned, the data collected by the system will be discarded. This can be verified by the empty file list and conveys trust to the users.

Furthermore, to ensure transparency, the processing system will be made available for other developers to test and implement in their online shops. This includes the algorithm used to process the data as well as how the data is temporarily being stored.

5.2 Ethics Regarding Artificial Intelligence

Artificial Intelligence and Machine Learning are topics of controversy when it comes to making ethical designs. The GDPR [23] states that users must be informed of the use of AI in decision-making and must be offered the choice to opt-out of said decision-making. Therefore, we thought it would be appropriate to discuss the use of AI for the processing of the user data, and how we handle the GDPR conditions.

Our system uses AI to analyse the Excel and PDF files uploaded by the users and determine which products they are referring to. This process may include decisions on what label corresponds to the different fields of the data, correction of misspelt data and correction of missing data. The resulting basket is then shown to the user, who has the option to review and edit it. Furthermore, the availability and documentation of the project after its deployment ensures that the AI is explainable and accessible to others.

We believe the system does not raise any ethical concerns when it comes to the use of AI. The user can review the decision making and opt out of it, the user is informed of the use of AI, and the AI is explainable. These conditions correspond with the GDPR statements of an accountable AI and enforce the backwards-looking responsibility of the project developers.

Chapter 6

Implementation of the Order Processing System

In the following chapter, we will be going over the implementation of the order processing system. We started the process of implementing the application in the third week of the course after we completed the plan and had multiple meetings with the client. The project was implemented on the Tu Delft GitLab [1] where the TA and Coach, as well as the client, had access to our work and weekly issues.

We will be looking at the database and how we stored the Furning data, at the frontend and what changes we made from the initial plan, at the backend and what APIs we use, at the processing algorithm and how it works, and lastly at how we tested everything.

6.1 The Database of the System

Since the client wanted this project to be potentially used as an extension or an add on, on top of their current business Furning, they decided to provide an instance of the website's database. This was in form of an excel file, which the team would make use of throughout the project.

To gain more insight into the provided data, it would first be necessary to understand what Furning is used for. Furning is a website specialised in finding your desired furniture just by uploading a picture of your desired product. If the user would prefer their product to have a different colour or material, they can make use of the Furn Editor feature available which will redirect them to the closest possible match on the market based on their preferences. This application does not focus on selling own made products, but rather on connecting the customer to the brand that would suit his needs.

Having said all of the above, we could now understand the low level of structure the database provided had. The database had an estimated number of 450 thousand records, all of them being products from different websites. One instance of such a product would have a number of 20 properties namely: id, title, description_short, description_long, price_gross, price_gross_discount, price_net, delivery_cost, brand, delivery_time, availability, availability_items, gtin, merchant_shop_id, product_image, partnershop_name, furnlinen_category_seo_alias, furnilen_color, furnlien_color_hex, furnlien_delivery_days.

[1] https://gitlab.tudelft.nl

Looking at the structure of the data, the decision has been made to look for a NoSQL database, which would be optimized for intensive reading and scanning operations. The database would preferably be of a serverless type to reduce the amount of maintenance needed to be done by the host. Dynamo DB, an Amazon non-relational database, seemed to have been the best choice in this scenario.

In the end, as described in Section 4.3.3, after having several issues with querying and realizing that scanning the whole database could turn out to be expensive since the price would increase per query, it has been decided that it is an unfeasible solution and possibility expensive one.

The database used currently on this project is PostgreSQL, a well-known open-source relational database, that is known for its connection with python when used as a backend programming language in the technology stack.

An instance of the Product table can be seen in Appendix E with all of its properties.

6.2 The Frontend of the System

In this section, we will be discussing the implementation for the frontend of the application. The frontend is the area of the application with which the user interacts, therefore it must remain intuitive, informative, and aesthetic.

We will first be discussing how we made use of React and its features, and then describe the individual components of the frontend.

6.2.1 How we used React for the Frontend

As mentioned in Section 4.3, the React framework consists of many features which helped us implement the product. The most helpful feature was its modularity when it comes to the components. The components act as separate elements, such as buttons or inputs, which can be written a single time and later called from anywhere. These components also feature properties, or "props", which act as the means to customise the component to special needs. In the case of a button, that could be the function it has upon clicking it, thus developers can reuse the button component while assigning it different functions for different cases. This modularity allows for very fast development, saving the developer time from rewriting components. Furthermore, it allows for easy refactoring of the Frontend; if we decided to change the layout, we could have simply moved the component to the desired place while still keeping its initial functionality.

However, component-based modularity can bring some problems. Having many components with many properties means that those properties have to be declared somewhere. It is a best practice that they are declared on the top level of the architecture and passed down to the various components, to maintain consistency across the system. This can cause the main file of the application, in our case **App.js**, to become very cluttered. It can also cause the changes in the application to be less predictable and traceable. One way of preventing this would have been to use React Redux [2], which allows for the state of the application to be maintained without passing down properties but rather the use of "Stores" of session memory.

[2] https://react-redux.js.org/

Unfortunately, when trying to implement Redux later throughout the project we realised we would have had to rewrite everything. Since this would have caused us more problems than it would fix we decided to leave it as it is. It is important to mention, however, that for future projects in React and future iterations of the application we would like to make the change from the start, as will be discussed in Chapter 7.

6.2.2 The React components we used for the System

We will now be giving an overview of the main components of the application's frontend, discussing the structure and showing visuals from each.

Home Page and Pop-up Window

The Home Page, as the name implies, is the first thing the user sees when accessing the application. Therefore, it must consist of all the functionality and information they would want. As shown in the Low-Fidelity Prototype in Appendix C, we planned for the home page to have search functionality as well as a pop-up window where the user can add files to be processed. Bellow the search bar was to be displayed a list of the searched products. Not much styling was shown in the prototype, as it was merely a guide.

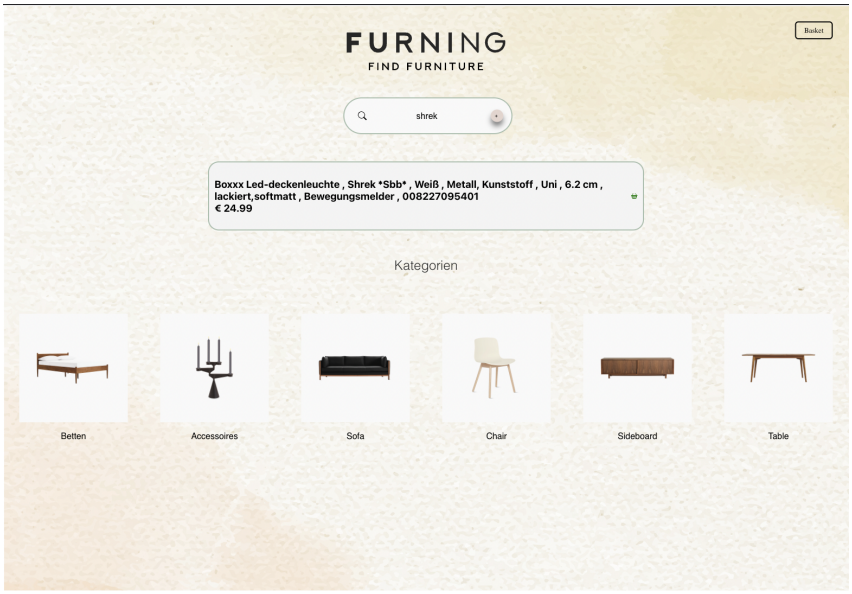


Figure 6.1: The Home Page of the application after a search was made.

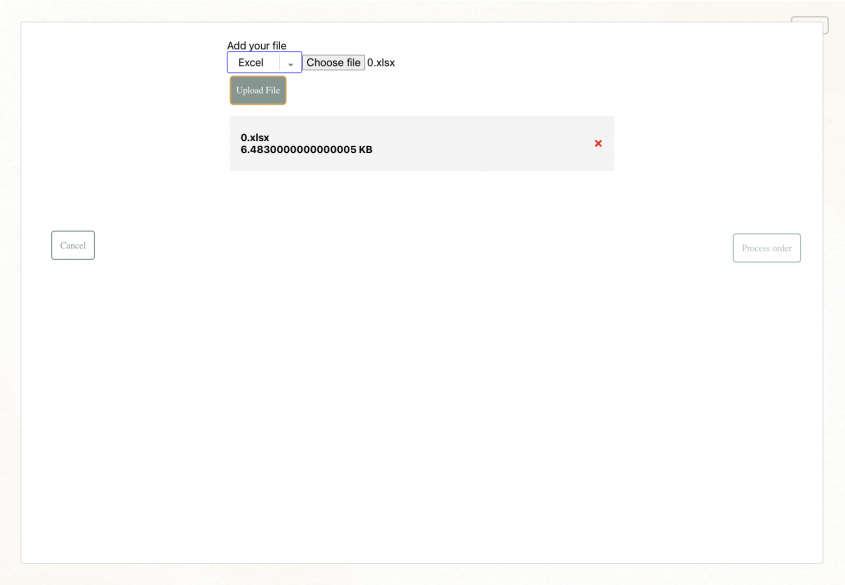


Figure 6.2: The pop-up form in which users can upload files for processing.

As can be seen in Figure 6.1, the elements of the home page were centred and styled using a consistent colour palette. The list for the searched products was replaced by individual elements in a list formation to allow for the "round" aesthetic of the entire application. A categories section was also added underneath to better imitate the Furning website we were implementing for. The home page also features a button to go directly to the basket page. Upon clicking the "+" button in the search bar, the pop-up window in Figure 6.2 will appear.

The pop-up window was changed to fit the entire screen, this allowed for the user to upload more files with longer names without having to scroll on a smaller window. It also made the pop-up clearer for users on a smaller screen, such as a phone. The user has to select what type of file they wish to upload and then are only allowed that type, this decision was made to be able to send all the files to the backend at once without having to worry about what processing method they require. The rest of the component is largely the same as the design.

Order Page and Products

The Order Page was designed (Appendix C) to be shown to the user after the files they uploaded have been processed. There they can edit them and delete them if necessary. The page was to contain a list of the products, featuring their name, quantity, price per unit, and expected delivery date. Lastly, the page would have had an "Add To Basket" button and a "Cancel" button.

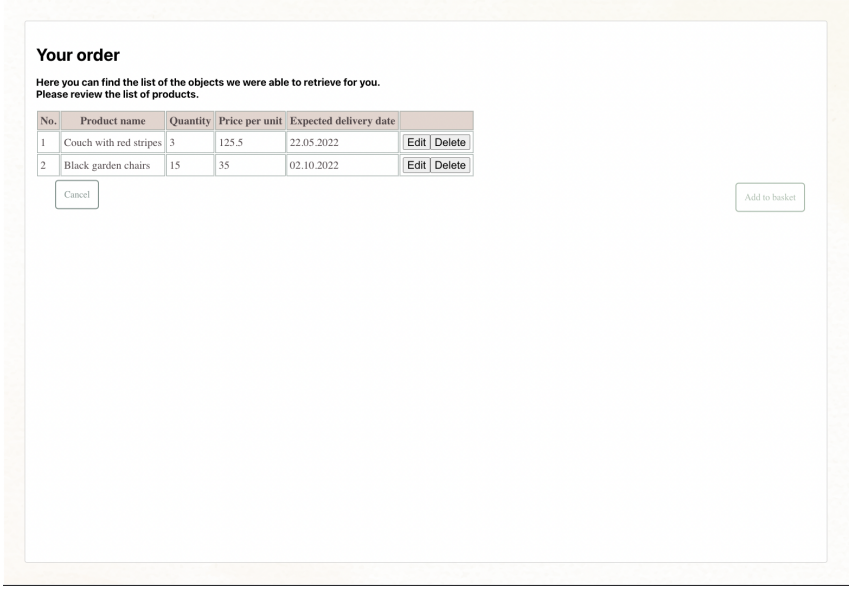


Figure 6.3: The Order Page of the system, featuring the Products component as a table.

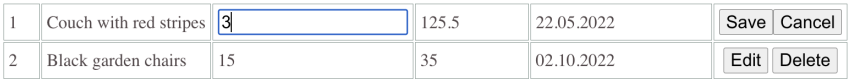


Figure 6.4: Example of an editable product (top) and a read-only product (bottom).

The implementation of the Order Page, seen in Figure 6.3, is nearly identical to that of the design. One difference is that the warning is now given by colouring the product row red for the products which have higher uncertainty. The other difference is that it is part of the same pop-up window as the file form, this is because it is the continuation of the last interface. The page features the Products component, which displays a table with rows of products. The products are read-only, but when the user selects the "Edit" button they become editable. An example of these two states of products can be seen in Figure 6.4. Upon clicking "Save" the product becomes read-only once more and the change is saved in a frontend JSON file before being sent to the basket.

Basket Page

In the original design, we did not account for a basket page, we planned for the order page to act as a basket page. This, however, introduced inconveniences when having multiple processes of orders. We wanted the user to be able to review every process as well as the final basket, therefore we created a Basket Page.

Figure 6.5: The Basket Page of the application with two products in the basket.

As shown in Figure 6.5, the basket page consists of all the products added to the basket from all processes as well as manual searches. Thus it resembles the traditional basket page of other online shops.

6.3 The Backend of the System

This section will discuss the design choices and implementation of the backend of the application with most focus on how the Django Framework has been integrated.

As mentioned in Section 4.3.2, Django is a framework based on the Python programming language, that uses the (MVT) Model View Template design pattern. In the Model View Template architecture we recall the Model as a python class that manages the data represented by the database, the View as the point of interaction for HTTP responses and requests and the Template as the front end layer which serves the dynamic HTML component of the Django application.

The interaction between the aforementioned components can be seen in the diagram below:

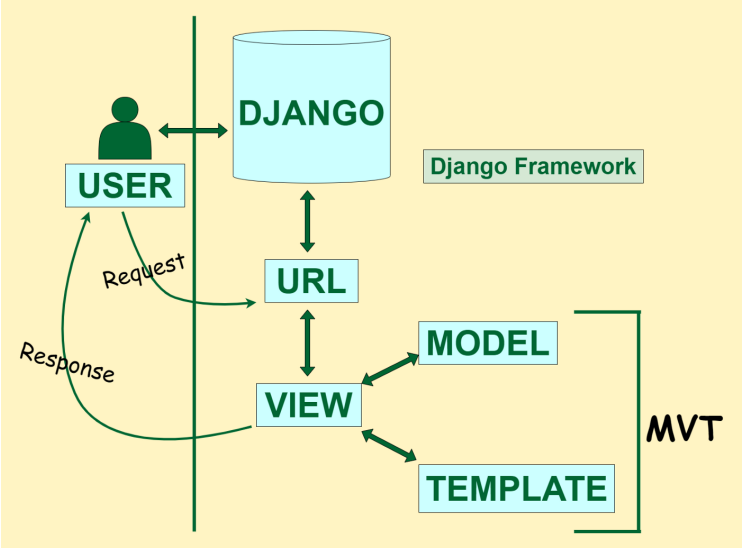


Figure 6.6: The interaction between the Model View Template, Django and the User.

In the case of this project, the Django backend has 2 main applications: order process and frontend. The first one contains all the logic that the backend uses when dealing with any type of request from the user. The frontend is served as a static index.html file which ensures the user of a responsive UI using the React framework.

The communication between the user and backend has been done by making use of API endpoints and the REST framework that is built-in Django.

As mentioned in the section talking about the database of the system, the application makes use of a SQL database, however with only one table, namely the Product table. Although the application focuses on the read functionality, to access the data that has been queried a serializer class has to be created for each table in the database. In this case, the only serializable table was the Product, which has been serialized to only display the id, title and price gross estimate, to fit with the needs of the frontend.

The way a user would interact with the application would be by making a request via the frontend or any external API tool, which would be processed in the backend by the Django framework which would eventually return a response.

A few common and relevant endpoints that are used across this project are: sending the uploaded files and getting all the items with a matching title.

The way a user would search for a certain product, i.e. sofa, would be by requesting to the localhost:/8000/search-title/<Sofa>. Then the backend would check if such a URL can be mapped to the list of possible URLs, redirect it to its corresponding view if found, where the function get_products_by_title(sofa) is applied. This is a function that will query the database to search for a product which contains "sofa" in its title description. The query would be done by using the psycopg2 library [3]. If more than 10 matches were found, the response to the frontend would be a list of the top 10 matches. The result can be later observed on the main screen of the application below the search bar.

One big setback that the team faced was sending multiples files from the frontend to the backend. To deal with that and cope with time constraints, there has been implemented a workaround, in which the list of files from the frontend are being transformed into a JSON file, then sent to the backend and later transformed back into the initial Excel/PDF using various popular Python libraries such as Pandas [4].

6.4 The Excel Processing Algorithm

The server of the application contains an algorithm that incorporates the automatic processing of Excel orders. Whenever a user makes use of the 'upload' function and presses 'process', the file uploaded is sent as a JSON to the server. The backend fully processes the file and returns a structured JSON. This section describes the various algorithms and functions that are used in this process.

6.4.1 Overview of the Processing Algorithm

Excel files are semi-structured data files. The data is divided into cells that belong to a certain column and row. To allow for fast and powerful accessing of the data in the Excel files, the Pandas library is used. The first part of processing an Excel file is converting the Excel file to a Pandas DataFrame. Afterwards, the dataframe is filtered to only contain information that is classified as important for further processing. The filtered dataframe is then used to map the correct label to each column. A label indicates what type of information can be found in the column. The last step consists of extracting the title, amount and the price of each product from the order. The last three steps will be described in the following subsections.

6.4.2 Filtering out Information from Excel Files

The application does not enforce any rules about the structure of the Excel files. Users may use any structure, as long as the structure is column-based (i.e. information about one product is spread over one row). Appendix F shows two examples taken from the data that has been provided by the client. These examples show how different orders may contain different information in a different structure. This made it complicated to retrieve information in a structured way.

[3] https://pypi.org/project/psycopg2/
[4] https://pandas.pydata.org/

To ease the task of extracting information, multiple algorithms are used to exclude and remove unnecessary columns and rows from the converted Excel file. Firstly, the dataframe is passed to a function that checks and removes the first row if it's used as a 'header' row. This is done by matching the content of the first row to a 'Labels' object that contains various labels that can be found in the Excel orders. After checking and removing the header row, the dataframe is passed to a function that checks and removes 'order columns'. Order columns are columns that are used to indicate the number of each row in an Excel file. The last filtering function removes any empty rows and columns.

6.4.3 Mapping Columns to the Correct Labels

After applying the previously mentioned methods that filter a converted Excel file, the dataframe columns are mapped with the correct labels. Labels indicate the kind of information that can be found in a column. This is done by first extracting all possible labels that can be found in the excel orders that the client has provided. In total there are 11 labels, but only three are needed to extract a structured order; product_name, quantity and price. A column mapped to the 'product_name' label contains the different names of the products that a user wants to order.

To map the columns to the correct labels we have made use of an Artificial Neural Network. The Skicit-Learn library has been used to create and use the neural network [5]. The client has provided the team with a file that contains the correct labels for each excel order. Each cell in each order has been used to either train or test the neural network. Since neural networks only accept integers as input, all characters had to be converted into integers. For this, all characters have been encoded into ASCII characters. Regular neural networks only accept fixed-length input data. The length of each input data point has been set to 20 ASCII characters to accomplish this. If a data point consisted of fewer characters, the data point was padded to 20 characters by appending the ASCII encoding of the character the following character: "'". For data points that contained more than 20 characters, only the first 20 characters were used. Each data point was then mapped to a different label. The labels have been encoded from 0 to 10. This was also the corresponding output for the neural network.

The neural network has been tested, validated and optimized to get the highest accuracy. The client made 500 Excel orders available for learning and testing. Eighty per cent of the data has been used to train the neural network, while twenty per cent was reserved for testing. The Skicit-Learn library provides the option to optimize a neural network by providing hyperparameters. Each combination of hyperparameters is then tested and validated using the K-Fold Cross-Validation Algorithm. The result of the optimization was a neural network with 5 hidden layers. Each hidden layer consisted of 100 neurons. The neural network achieved an accuracy of 82% on the test data.

The cells in the filtered dataframe are encoded in the same way as the training data to map the correct label to each column. For each column, all cells are provided as input data to the neural network. Each cell is then mapped to a corresponding label. The column is then mapped to the most frequently occurring label. This achieved great results as the columns are mapped to the corresponding label with 99% accuracy.

[5] https://scikit-learn.org/

6.4.4 Extracting Product Information

After mapping each column to the correct label, an order is created by passing the labelled dataframe to a function that extracts information about the products and mentioned quantity in the Excel order. To extract the products, the function makes use of the StringGrouper library [6]. String grouper matches the product names to the most similar item in a group of items. To extract a product the function tries to match the mentioned product name to the most similar product name from the database.

After the matching completes, an object of type Product is created for each product that is mentioned in the Excel order. This object contains all information that is needed to process and view the structured order for the user. The quantity, mentioned price (in the Excel order) and the actual price (from the database) are provided in addition to an extra field that identifies the frontend system whether the product exists (a match has been found) in the database or not. All objects of the corresponding products are added to a list and sent as a JSON to the frontend.

To measure the accuracy of the products the extracted products were matched with the products that the client has provided. However, not all products existed in the provided database. About 40% of all products that were mentioned were found to exist in the database. This has caused some complications with calculating the accuracy of extracting the products, i.e. the product accuracy. To overcome this issue, the client agreed to only count products that can be found in the database towards product accuracy. This allowed for a product accuracy of about 98%.

6.5 Testing the System

The application has been tested to ensure the client receives a satisfying product according to their requirements, and the user has a reliable tool for their needs. Unfortunately, we started with the testing rather late in the process, something that will be touched upon in Chapter 7, meaning we had many classes to cover. We did manage to cover all classes and components in both the frontend and the backend.

This chapter will talk about the testing that was made on the components in the frontend, the testing in the python backend, and the testing for the processing and querying files.

6.5.1 Testing the Frontend of the System

The frontend features full component testing as well as testing of their functions. The component testing consists of the component rendering without crashing and it rendering with the correct properties and contents. This type of testing uses component mocks and function mocks from the Jest library [7] to simulate the behaviour of the program compared to the expected outcome. Using this technology, the frontend has full component and method coverage in all classes.

[6] https://pypi.org/project/string-grouper/
[7] https://jestjs.io/

6.5.2 Testing the Backend of the System

For testing the product, it is not only important to test the frontend of the system, but also to test the backend of the system. The backend of the system consists of the classes related to the heuristics. To test the functionality of these functions related to heuristics, it is important whether the functions actually act the way they should act. This is tested by giving a certain input and asserting the expected output with the actual output. The software testing framework used for testing the Python code is called Pytest. Using this framework, it is possible to write easy and efficient tests.

In addition to the classes built to deal with the Heuristic approach, several other functionalities in the backend had to be tested. One of the main ones was regarding the endpoints created and making sure that they work across different users. To do that, several tests were created using python to mimic the way a request would be done from the frontend. In addition to that, during implementation [8] POSTMAN has been used as an API testing tool to ensure proper functionality.

Since the main functionality of our application is based on interacting with a black-box algorithm, namely Neural networks, it was of utmost importance to not use all the provided data to train it, but also to test it and validate it afterwards.

The above-mentioned tests have been added to the pipeline which besides usability tests, also ensures that the code has been written into a pre-defined style, using the Checkstyle extension.

[8] https://www.postman.com

Chapter 7

Product Discussion and Future Recommendations

As in any project, it is important at the end to have a clear overview of what the product that has been designed is capable of doing and how this can further be improved. During the past weeks, an Order Processing System has been developed with the aim of automating the currently manually done processing of large orders. This automation is beneficial from multiple points of view, from the decrease in human-made errors to the increase in the speed at which orders can be fulfilled [24]. Although the product developed is quite a complex one, offering a wide range of functionalities, it goes without saying that there are several improvements which could enhance the reliability and usefulness of our product for its users. Therefore, in this chapter, a clear overview of the functionalities of the system is given. Additionally, the last sub section discusses future improvements that can be made.

7.1 Functionality Offered by the Order Processing System

To understand fully the different functionalities that that system needs to offer, the MoSCoW method has been followed and the different requirements were categorised accordingly. The goal was that at the end of the ten weeks to have a system which has the capabilities described by the Must Have requirements implemented and some of the Should Have requirements, in accordance with the time limit. Having followed this approach has helped prioritise the different tasks that were being carried out, giving precedence to those functionalities that were described in the two discussed categories and making sure that a product is reliable, in the sense that it performs exactly what the user expects it to, is delivered.

Having discussed with the client the time constraints and feasibility of implementing the automation for both Excel and PDF files, a decision has been made such that the focus should be to have the processing of the Excel files fully functional and, if time permits, get a start with the processing of PDF files, categorised rather as a should have and not considered part of the minimum viable product.

The product offers a wide range of functionalities, all with the aim of making online ordering a more pleasant experience for the customers. The main functionality of processing large orders made through PDF files is accompanied by the option to search for products through the catalogue of products or modify the basket to a user's preference before checkout.

The Excel upload and processing functionality can be easily used by any customer. On the home page of the web application, a user can see the search bar, which alongside the search button, has a button marked with a "+" sign. Once a user clicks this button, he is prompted with a pop-up where he can select to upload their preferred files. After uploading the files the user needs to press the "Process Order" button, which will send the files he has uploaded into the backend for the processing phase. The processing phase consists of an algorithm which recognizes the columns from the Excel files and maps them to specific names and types for each column. Afterwards, the algorithm filters out unnecessary information and turns the final product into a structured Python data type to be sent back to the frontend. The resulting list of extracted products from the database is displayed on the next page. Here the user can still modify the quantities of the products he has decided to order and review whether their order was correctly processed from the file uploaded. Products that were incorrectly found in the database or differ to some extent from the products mentioned in the file with the other (i.e. prices differ) will be highlighter, offering the user the possibility to remove these and therefore assuring an as transparent as possible process. This processing functionality currently works for uploading Excel files as intended and as discussed with our client. After reviewing the list of products retrieved from the database for him, the user can add these products to his shopping basket and continue shopping.

Another one of the main functionalities that a user can benefit from when using this product is that they can also manually search for singular products. The system offers a search bar with the capability of processing human input and querying the database in an attempt to find the most suitable match for what the user is looking for. Usually, the top 10 matches are returned for the user to see, from which he can choose and add their preferred ones to the basket.

Lastly, the user can access their basket at any time during his shopping session, he can view the products that he has added to his basket and can modify their quantities or delete them at any time. Furthermore, once a user decides to terminate their shopping session they can go to the checkout and follow the appropriate steps to place the order.

The product was designed following the client's idea that once implemented and fully working, the main functionality of automating order processing of PDF and Excel files will be integrated into other websites, for the companies which will be open to using our product. Therefore, the designed web application serves more as a high-fidelity prototype to showcase and support the processing functionality. In this sense, there were not many other important capabilities that the web application can do, whereas the focus should remain on the way that the processing works.

An overview of the functionalities that the system offers can be found in Appendix G, where the list of requirements only contains those requirements that have been fulfilled.

7.2 Future Improvements for the Order Processing System

The challenge that came with implementing such a product is twofold. First, the system makes use of different technologies and programming languages, for some of which the knowledge on behalf of the team was limited. Second, the short amount of time for implementing the product played a major part in the decisions that were taken, starting with the requirements engineering step.

The first and most urgent matter that needs to be taken care of with regards to this product is to enable the accurate processing of PDF files. Although the user can now upload PDF files, these are not correctly processed in the backend, and although we have agreed that this is not a Must-Have for our application, the initial wishes of our client were for our web application to include such functionality. However, rather than an improvement, this can be seen as an extension of the current status of the product.

The system makes use of a PostgreSQL database which contains roughly 450 thousand products, each with a large number of properties. Furthermore, given that the database of the website from which the information for the database was acquired makes use of input extracted from a high number of different websites, the data that is contained in the database is highly unstructured. The database that is being used is a well-known, open-source relational database which is very efficient when working with Python as a programming language. However, given the nature of the data, a NoSQL database can be a better option, which is optimal for reading and scanning operations, the majority of the operations used by the system being of that type.

Another aspect which could be improved is the time at which the processing is done. Currently, a user needs to upload the desired times and then press another button which will send a request to the backend component application and trigger the processing of the uploaded files. However, there would be a possibility to process excel files on the go, the moment that a user uploads the file, creating the request to the backend on every upload of a file. This can improve user experience, whereas the time waiting for a response and thus the list of extracted products can be significantly reduced. However, there might be other dimensions to this aspect, such as ethical ones, which need to be considered, especially when processing data provided by the users, which fall outside of the scope of this report and would need to be analysed in the eventuality in which this improvement will be considered for future implementation. Currently, the request sent after the user uploads the desired files contains a workaround which might not be the most effective one. The files are decomposed into a list of JSON files structure and sent to the backend where they are reconstructed into Excel files, due to some limitations that were encountered during the implementation. This so-called workaround can cause lossy conversions which lead to some of the lines in the Excel file being lost. This can usually lead to performance issues and incorrect results, heavily affecting user satisfaction and trust in this system. Therefore, further improvements in the communication between the frontend and the backend could increase accuracy and ensure user satisfaction.

Lastly, there are two improvements from which the Order Processing System implementation would benefit, which also serve as a good lesson to learn for future projects. The first one is the use of Redux or another state management framework. Due to the limited amount of time and knowledge about React JS, the frontend does not use any state management tool, which makes the code harder to understand and maintain by future developers. This is however usually a decision that needs to be taken before starting the code implementation, whereas refactoring the code to make use of a tool such as Redux would be a timely operation which was not feasible to do within these 10 weeks. Furthermore, considering that the web application only serves as a prototype for displaying the actual processing functionality, it was not considered as being a necessary step to spend a lot of time on. The second one is the need for testing throughout the implementation phase of the project. It is important to write tests for the different components of the project at the same time at which the code is being written, to ensure that it works as intended and that its quality meets the standards.

Chapter 8

Conclusion

The Order Processing System is a tool meant for the automatization of a process that has been manually done for many years now and is aimed at reducing the time that it takes for orders to be fulfilled once the clients have expressed their wishes. The importance of the existence of such a tool has been studied in the second chapter of this report.

At the beginning of the 10 weeks dedicated to the Software Project course, the two main goals of the project have been outlined as being developing a system to process an order and having a scalable implementation. Seeing that now the system has turned into a viable product which can be used by large amounts of clients, this section aims to draw conclusions on the extent to which the product meets the initial goals that have been set.

On the one hand, the main goal of the project was to be able to develop a product which offers as the main functionality the possibility for users to upload files and receive back a list of the products that they want to order. This is indeed accomplished through the project, whereas the functionality for uploading Excel files and receiving a structured order out of them is now fully working. There are still improvements to be made on the PDF processing part, which due to the time limit is not yet a functionality which can reliably work.

Furthermore, the focus lied on creating a scalable implementation which allows other software developers who will take over the project to continue working on it and further improve it. This goal has been achieved through regular quality checks of the code that was being written, which is now good enough to be passed on to the next team of people who will work on it. The one aspect that could use improvement is the way the states are managed on the frontend, which is currently not of the desired quality.

Lastly, the team process has been a really good one, as the team as a whole has evolved ever since the beginning of the project. A tight schedule was kept to make sure that the final requirements are met, as well as regular meetings for feedback rounds and help have been organized weekly. Each member of the team has worked weekly on their task and there has never been a situation in which someone did not stick to what they were supposed to do.

The automation of processes has proved to be a very useful one, especially as a support for smoother collaborations within supply chains, but not only. The product developed throughout the last 10 weeks has proven to be an effective way of supporting this and is a good start for a system that can be extended to accept multiple types of files and increase customer trust and reliability in such systems.

Bibliography

[1] G. Partners, "Top 5 benefits of globalization," 2020. [Online]. Available: https://www.globalization-partners.com/blog/top-5-benefits-of-globalization/

[2] Conexiom. (2021) Don't Wait to Automate. [Online]. Available: https://conexiom.com/resources-category/infographics/

[3] Palette. (2021) Landstar benefits from Automated Invoice Processing with PaletteSoftware. [Online]. Available: https://www.rillion.com/case/landstar/

[4] Capgemini. (2018) Consumers are embracing AI and will reward organizations that offer more human-like AI experiences. [Online]. Available: https://www.capgemini.com/us-en/news/consumers-are-embracing-ai-and-will-reward-organizations-that-offer-more-human-like-ai-experiences/

[5] A. Maier. (2020) Expert insights into Customer Experience. [Online]. Available: https://codecoda.com/en/blog/entry/expert-insights-into-customer-experience

[6] Z. Jin, "Chapter 3 - importance of interactive environment," in *Environment Modeling-Based Requirements Engineering for Software Intensive Systems*, Z. Jin, Ed. Oxford: Morgan Kaufmann, 2018, pp. 29–39. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128019542000030

[7] Concepta. (2018) How to Define Stakeholders for Your Software Development Project. [Online]. Available: https://www.conceptatech.com/blog/how-to-define-stakeholders-for-your-software-development-project

[8] Unetiq. (2022) Unetiq - Your Agency for AI and Data Science Services. [Online]. Available: https://www.unetiq.com

[9] S. Khan, A. B. Dulloo, and M. Verma, *International Journal of Information and Computation Technology*. International Research Publications House, 2014.

[10] M. Glinz, "On non-functional requirements," in *15th IEEE International Requirements Engineering Conference (RE 2007)*, 2007, pp. 21–26.

[11] Komodo. (2022) How to Use MoSCoW Prioritisation to Help Deliver Better Software Products. [Online]. Available: https://www.komododigital.co.uk/insights/how-to-use-the-moscow-prioritisation-technique-to-execute-software-projects

[12] N. T. More, B. S. Sapre, and P. M. Chawan, "An insight into the importance of requirements engineering," *International Journal of Computer and Communication Technology*, vol. 8, no. 1, pp. 29–31, Jan. 2017.

[13] U. Pisuwala, "The benefits of reactjs and reasons to choose it for your project," Mar 2022. [Online]. Available: https://www.peerbits.com/blog/reasons-to-choose-reactjs-for-your-web-development-project.html

[14] A. Affairs, "Prototyping | usability.gov," 2022. [Online]. Available: https://www.usability.gov/how-to-and-tools/methods/prototyping.html

[15] "Advantages and disadvantages of python - how it is dominating programming world," Aug 2021. [Online]. Available: https://data-flair.training/blogs/advantages-and-disadvantages-of-python/

[16] A. Joy, "11 advantages of django: Why you should use it," 2022. [Online]. Available: https://pythonistaplanet.com/advantages-of-django/

[17] V. Singh, "Flask vs django in 2022: Which framework to choose?" 2022. [Online]. Available: https://hackr.io/blog/flask-vs-django

[18] A. Arkhipov, "Serverless databases: Pros & cons, benefits, examples on the market," 2021. [Online]. Available: https://www.techmagic.co/blog/serverless-databases/

[19] A. Newbies, "AWS DynamoDB: Non-Relational Database," 2022. [Online]. Available: https://awsnewbies.com/dynamodb/

[20] "Advantages of PostgreSQL," 2022. [Online]. Available: https://www.cybertec-postgresql.com/en/postgresql-overview/advantages-of-postgresql/

[21] I. Van de Poel and L. Royakkers, *Ethics Technology & Engineering An Intro*, 1st ed. John Wiley & Sons, Ltd., 2011.

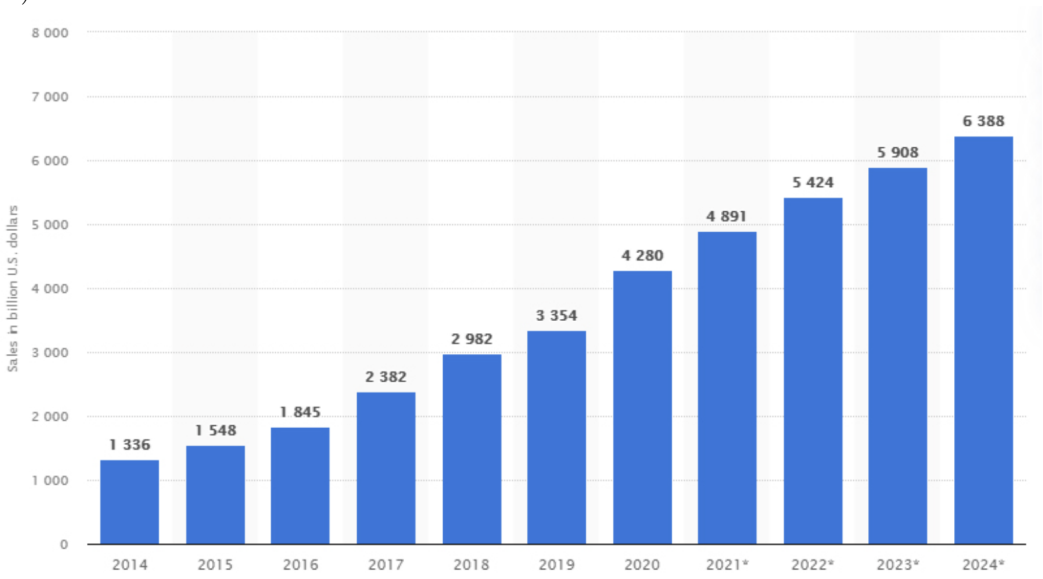
[22] B. Friedman, P. H. Kahn, and A. Borning, *Value Sensitive Design and Information Systems*. John Wiley & Sons, Inc., 2008, p. 69–101.

[23] (2018) 2018 reform of eu data protection rules. European Commission. [Online]. Available: https://ec.europa.eu/info/

[24] S. Callarman, "How automated order processing accelerates your ecommerce business," 2017. [Online]. Available: https://www.shipbob.com/blog/automated-order-processing/

Appendix A

Graph displaying the worldwide growth of the eCommerce market (note: * signifies prediction).



Source: statista.com

Appendix B

The full list of requirements for our project compiled after multiple meetings with the client.

Non-Functional Requirements:

- The system will be scalable to the extent that it can also process orders of other file-types.
- The system will make use of a server-less database.
- The system will process at least 70% of the excel orders in a correct way.
- The system will run on a web application with React Framework.
- The system will have a Python back-end.

Must Haves:

- As a customer I want to be able to search for a product on a search bar.
- As a customer I want to get a list of products that match my search keys.
- As a customer I want to be able to upload an excel file with an order.
- As a customer I want to get an 'automatically created structured order' after uploading my order in an Excel file.
- The system will filter out unnecessary information from the excel orders.
- The system will map the column names for each column in an order.
- The system will map the types for each column in an order.
- The system will convert a filtered order into a structured Python datatype.

Should Haves:

- As a customer I want to be able to download a structured CSV file with the automatically created order from the Excel file that I have uploaded.
- As a customer I want the system to notify me if my order contains an unavailable product.
- As a customer I want the system to notify me if my order contains a product that does not exist.
- As a customer I want the system to notify me in case my order can not be processed.
- As a customer I want to be notified if an actual price differs from my expected price.

Could Haves:

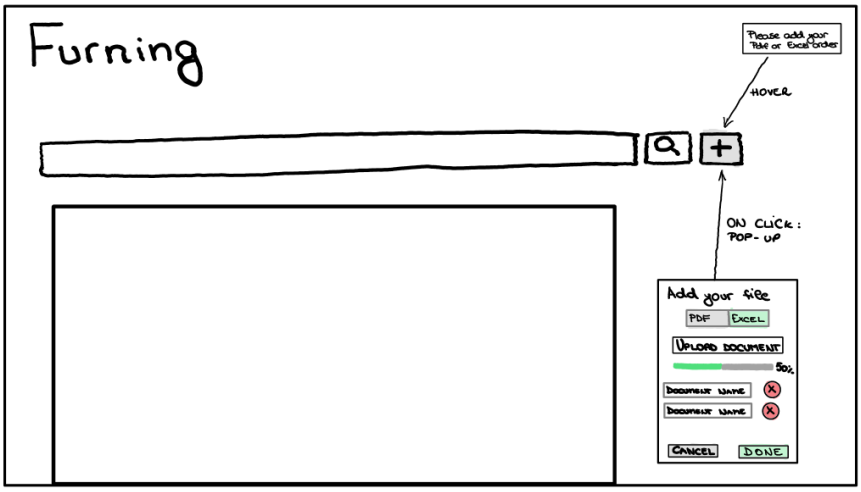
- As a customer I want to be able to manually add products to the automatically created order.
- As a customer I want to be able to change an automatically created order.
- As a customer I want to be able to delete items in an automatically created order.
- As a customer I want to be able to let the system know whether my order was processed correctly
- As a customer I want to be able to upload a PDF file with an order.
- As a customer I want to be able to upload multiple files at the same time.

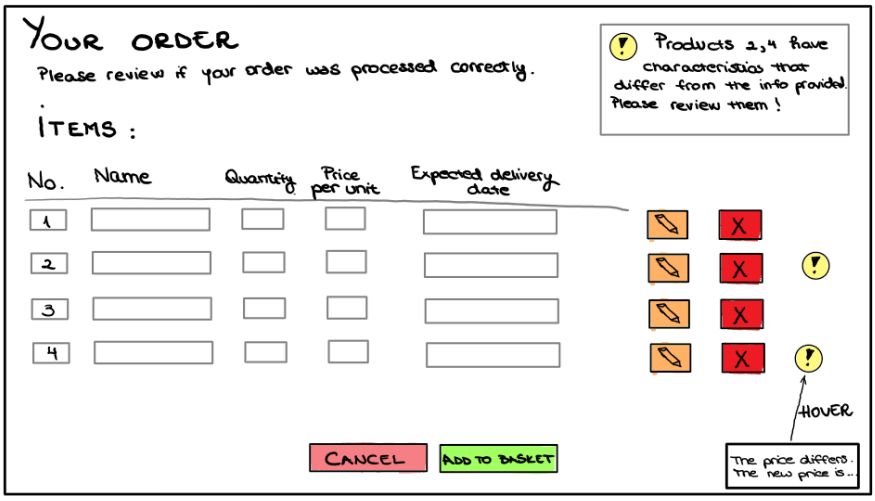
Won't Haves:

There are no requirements that we will certainly not do.

Appendix C

Low-Fidelity Prototype showcasing the Home Page and Basket Page respectively.





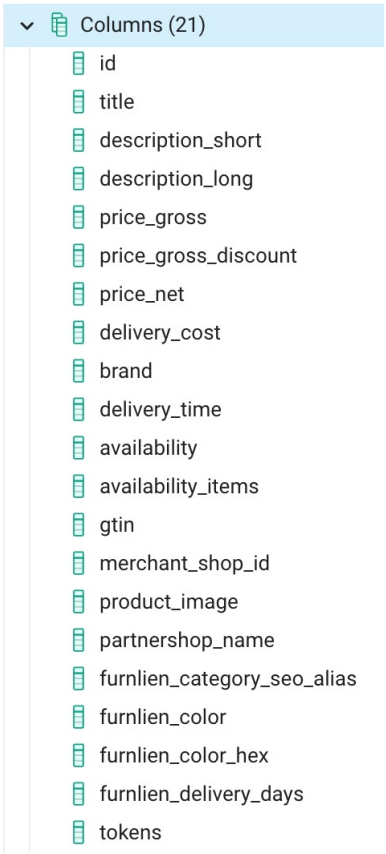
Appendix D

Table of useful comparisons between Django and Flask that helped us decide what is best for our project.

Parameter	**Django**	**Flask**
Type of framework	Django is a full-stack web framework that enables ready to use solutions with its batteries-included approach.	Flask is a lightweight framework that gives abundant features without external libraries and minimalist features.
Working of Framework/Data Model	Django follows an object-oriented approach that enables object-relational mapping (linking databases and tables with classes)	Flask works on a modular approach that enables working through outsourced libraries and extensions.
Project Layout	Django is suitable for multiple page applications.	Flask is suitable for only single-page applications.
Bootstrapping Tool	-Django-admin is the in-built bootstrapping tool of Django that allows the creation of web applications without any external input.	Flask does not come with an in-built bootstrapping tool.
Database Support	Django supports the most popular relational database management systems like MySQL, Oracle etc.	Flask does not support the basic database management system and uses SQLAlchemy for database requirements.
Routing and Views	Django framework supports the mapping of URL to views through a request.	Flask web framework allows mapping of URL to class-based view with Werkzeug.
Structure	Django framework structure is more conventional.	Flask web framework structure is random.
HTML	Django supports dynamic HTML pages	Flask framework does not support dynamic HTML pages
Usage	Django is suitable for high-end technology companies like Instagram, Udemy, Coursera etc.	Flask is suitable for companies and projects that want experimentation with the module, architecture of the framework like Netflix, Reddit, Airbnb, etc.

Appendix E

An instance of the Product table from the database, with all its properties.



Appendix F

Examples of Excel orders. These examples illustrate the different structures that are used.

	Anbieter	Artikelnum	amount	recipient	shop	rodukt-Tit	Currency	Datum	Preis Brutt	Address		
0		42514485(	no21	Muzaffer k	Havatex	Premium S	€	2022-06-0	45,90	Schwitalplatz	78206193	Pegnitz
1												
2												
3												
4		AB384077	17	Muzaffer k	Fun-Möbe	Boxspringk	Euro	2022-06-0	988,00	Schwitalplatz	78206193	Pegnitz
5		42505614	15	Muzaffer k	Havatex	Küchentep	€	2022-06-0	163,90	Schwitalplatz	78206193	Pegnitz
6		1195880	21	Muzaffer k	Zurbrügger	Z2 Kommode	JUTZLEI	2022-06-0	525,00	Schwitalplatz	78206193	Pegnitz
7		42505614	22piece	Muzaffer k	havatex	susal teppi	€	2022-06-0	72,90	Schwitalplatz	78206193	Pegnitz
8		AB717285	10	Muzaffer k	fun-mobel	boxspeingl	Euro	2022-06-0	729,00	Schwitalplatz	78206193	Pegnitz
9		1259177	6 Stücke	Muzaffer k	Zurbrügger	Pelipal Badblock TREI		2022-06-0	879,00	Schwitalplatz	78206193	Pegnitz
10		KA112526	qty 11	Muzaffer k	mobel-styl	kqwola es	EUR	2022-06-0	2.605,00	Schwitalplatz	78206193	Pegnitz

Figure E1: Example of a Labeled Excel Order.

Havatex	4,25E+12		2022-07-0	53,90		€	42510799	Kräusek Ve	Zobelallee	3Stücke
havatex	4,25E+12		2022-07-0	113,90		€	42508272	siwal teppi	Zobelallee	16
ichliebedesign			2022-07-0	33,50			A108886-0	filzunterse	Zobelallee	5pcs
havatex	4,25E+12		2022-07-0	597,90		€	42514485(	schmutzfa	Zobelallee	units20
Zurbrügger	4,05E+12		2022-07-0	285,00			1067524	Valnaturq	Zobelallee	5
Deko Welt	7,39E+12		2022-07-0	22,90		EUR	5836	Lichterbog	Zobelallee	6
ichliebede	4021224986945		2022-07-0	569,00			AJK0340	klapptisch	Zobelallee	1pcs
ichliebedesign			2022-07-0	458,00			AJK0026-0	outsoor tis	Zobelallee	19
Zurbrügger	4,01E+12		2022-07-0	39,99			1302882	Zassenhau	Zobelallee	9Stücke
Zurbrügger	0		2022-07-0	579,00			1157460	Zurbrügger	Zobelallee	16units
zurbrugger	4,01E+12		2022-07-0	9,99			1266605	emsa mikr	Zobelallee	no 8

Figure E2: Example of an Excel Order without Labels.

Appendix G

The following requirements are the ones that are currently present in the implementation of the Order Processing System.

Non-Functional Requirements:

- The system will process at least 70% of the excel orders in a correct way.
- The system will run on a web application with React Framework.
- The system will have a Python back-end.

Must Haves:

- As a customer I want to be able to search for a product on a search bar.
- As a customer I want to get a list of products that match my search keys.
- As a customer I want to be able to upload an excel file with an order.
- As a customer I want to get an 'automatically created structured order' after uploading my order in an Excel file.
- The system will filter out unnecessary information from the excel orders.
- The system will map the column names for each column in an order.
- The system will map the types for each column in an order.
- The system will convert a filtered order into a structured Python datatype.

Should Haves:

- As a customer I want the system to notify me if my order contains an unavailable product.
- As a customer I want the system to notify me if my order contains a product that does not exist.
- As a customer I want the system to notify me in case my order can not be processed.
- As a customer I want to be notified if an actual price differs from my expected price.

Could Haves:

- As a customer I want to be able to change an automatically created order.
- As a customer I want to be able to delete items in an automatically created order.
- As a customer I want to be able to upload a PDF file with an order.
- As a customer I want to be able to upload multiple files at the same time.