Bachelor Thesis:
A Data Management System for P3D

BEP TI3806

Client: P3D
Client contact person: Jeroen Gross
TU Delft Coach: Asterios Katsifodimos

Zeger Mouw,
Abri Bharos,
Tim Pelser,
Erik Sennema and
Gijs Paardekooper

Foreword

This project was carried out as part of the Bachelor End Project (BEP) at the TU Delft. The BEP is part of the third year of the computer science bachelor. The project owner is P3D which is a sub-company of Promolding, located in South-Holland. Our contact person from P3D is Jeroen Gross who worked with us to create the requirements for the end project and guided us through the inner workings of P3D. We have had many meetings with Jeroen ranging from daily meetings near the end of the project to 2 or 3 meetings per week at the start. We are very grateful with Jeroen for investing all his time into this project and with his help along the way when problems arose. Jeroen has taken this problem very seriously and has helped us obtain all the resources we needed to make this project a success.
From the TU Delft we were assisted and guided by Asterios Katsifodimos who works at the Web Information Systems group of the EEMCS faculty of the TU Delft. Asterios helped us to think about the way the project could be implemented and gave us helpful tips on how to go about doing this project. Asterios his knowledge about the structure of web applications proved of great help during our project and has helped us to get going much sooner than we would have without his help. During our meetings with Asterios it was great to see his enthusiasm and involvement with this project. We would therefore also like to thank Asterios for his assistance and motivation throughout this project which helped us to get to our final product.

Contents

1 | Introduction

P3D is an injection moulding company that uses a technology called PRIM®(Printed Injection Mould) to create products for its customers. Different from traditional moulding companies, P3D's main business model resolves around an efficient workflow and fast delivery of products to its customers. At the moment P3D is able to quickly fabricate and deliver its products because of a small team of skilled designers and engineers who have lots of expertise in the field of injection moulding. Unfortunately, relying on the expertise of its employees, a lot of data about the company's workflow is memorised or written down. This poses a problem as P3D would like to grow in the future and will not be able to entirely rely on memorised or analogue data. The goal of this project is therefore to create a data management system (DMS) that digitises information about P3D its workflow and saves it in an easily accessible centralised system.

The big challenge about this project is the fact that the system created will be linked to every aspect of P3D's workflow. It is therefore required that before executing this project, one has a general understanding of this workflow and the company in general. Furthermore, P3D's processes are continuously evolving because of customer demands, the system implemented should therefore be highly modular and extensible to keep up with this rapid innovation.

1.1 Document structure

As stated earlier, before work could be done on implementing the DMS, research had to be done about the inner workings of the company and how the system was going to be designed and implemented. This research phase is documented in chapter 2. Chapter 3 touches upon how the system was finally designed and which changes were made throughout the project. Chapter 4 touches upon how this design was implemented and presents the proof-of-concept. A section is dedicated to how the final system has been tested, which can be seen in Chapter 5. Chapter 6 provides an evaluation of the final product and chapter 7 evaluates the process leading up to this product. Finally, recommendations on how this project should be continued in the future are given in chapter 9.

2 | Research Report

2.1 Overview

The first two weeks of the project were dedicated to doing research. In the sections below the findings and the important details relevant to the project will be discussed. Also the problem raised by the client will be thoroughly explained. This will be done in section 2.1.1, as well as the problem analysis.

2.1.1 Problem definition and analysis

As stated above, the following section focuses on detailing the problem raised by the client. In 2.1.2, the problem will be explained and detailed to give the reader a good overview of the challenge P3D faces. In section 2.1.3, the problem is further analysed and requirements for tackling this challenge are presented. Furthermore, in section 2.1.4, the problem is put into context through a top-down overview of P3Ds workflow. Lastly, section 2.1.5 elaborates on existing technologies that exist on the market and why they fail to solve the problem defined in section 2.1.2.

2.1.2 Problem Definition

The client's core business model revolves around technology that was created within Promolding. They saw a potential business idea and decided to further develop it through a sister company called P3D. P3D uses 3D printed plastic moulds to produce small series of products by injection moulding. This is a much faster process than its competitors. This is especially relevant in industries where certification and testing has to be done early in the product development process. Since P3D is a relatively small company with a lot of human expertise, communication between production processes is mainly done person-to-person and not thoroughly reflected within a broader system. This directly leads to the client's problem: workflow and process data inside of the company is distributed over several departments. This forms a problem as data is not easily accessible for every employee, administrative tasks have to be done multiple times and is in essence very prone to human error. In order for P3D to become more efficient and resilient, this problem has to be solved.

2.1.3 Problem analysis

P3D has the ambition to grow and optimise its processes in the future, so the aforementioned problem should be tackled. During this project a solution will be designed, implemented and deployed. In order to comply with the client's needs, the end result of this project should not only fix P3D's current problem but should also be aligned with their image of the future and long-term goals. To this end, a system will be created that acquires and utilises the company's data, while keeping the possibility of adding new functionalities. More details on this will be further discussed in the design goals in section 2.2.

The proposed solution ensures that the company's workflow data can be accessed from a central point, on top of which useful functionalities can be built. This will make sure that the company's internal workflow can be made more resilient, efficient and robust.

2.1.4 Workflow P3D

In order to have a better vision of P3D as a company, it is essential to have a clear overview of their production process. In this section the 4-part production process will be detailed to the reader, giving him a clear overview on how P3D serves their clients. Issues or bottlenecks that are present in the workflow will also be pointed out.

Incoming Client

As expected, the first part of P3D's process starts with an incoming business inquiry from a potential client. From this business inquiry, P3D analyses the clients needs and wishes. This often involves a 3D CAD File which will be checked by an experienced employee. This is a process which requires human involvement, something that could lead to limitations in the future. After all requirements and checks are completed, the project will be planned with approval from the client. This planning process is done manually and not reflected in a broader digital system. P3D considers this to be an issue as it is a very fault-prone and repetitive process at the moment.

Printing/Mould Fabrication

Once a project is successfully planned, the printing process is initiated. It is important to understand that P3D does not print its products directly. Instead, a plastic mould is 3D printed or in case this is not possible, a steel mould is ordered from a trusted supplier. The products are then fabricated using these moulds. This is explained in more detail in the next part.

Production

The third part of P3D's workflow involves the actual production of the clients order. P3D uses a process called PRIM® (PRinted Injection Mould) which creates the desired products using the mould fabricated earlier. This process involves analogue machines and manual labour as some parts have to be reworked and checked by hand. The analogue nature of the process does not form a problem for P3D but not being able to easily access the overall progress status of the production is an issue. Furthermore, acquiring data about the multiple sub-processes involved in production is also not possible at the moment which is a limitation.

Delivery

The last step in P3D's workflow is delivering the order made by the client. Products are boxed and the delivery is outsourced to another company. At the moment, this does not lead to any complications. Data is not acquired about this part of the workflow which could lead to limitations in the future.

2.1.5 Existing Technologies

A trivial element of the research phase of this project is to check out existing technologies that could potentially solve the stated problem in 2.1.2. It is of course undesirable to come up with a custom solution if proven technologies and services already exist on the market. In this section, we will detail these solutions to the reader and cover why they do not meet all functionalities or requirements the client needs.

ERP

At first glance, it is tempting to say that an ERP (Enterprise Resource Planning) system could solve much of the problems the client faces. In fact, P3D already uses a ERP system called Plan-de-CAMpagne made by Bemet. This system is at the moment mainly used for keeping track of potential clients, creating invoices and other administrative tasks. Even though Plan-de-CAMpagne is already specifically designed for manufacturing companies, it still has its limitations. P3D thinks Plan-de-CAMpagne as well as other existing solutions available on the market do not meet their requirements mainly due to the closed nature of a premade system and a lack of flexibility inside of the planning process. As one of the main focuses of the client is to deliver products quickly, a tailor made extensible system would better suit P3D's workflow and therefore optimise its overall production process.

Other products

Other software products exist on the market which would solve some of P3D's demands. An example of this could be solutions built by a company called AMFG. However this system as well as other similar products on the market are heavily focused on 3D printing companies which P3D is not (3D printing is a tool in their production process).

2.2 Design Goals

This section will elaborate on the design goals for this project. They are highly important and will be reflected in the design and implementation of (every requirement of) the new system. The main design goals are: extensibility, maintainability and flexibility in that order of importance. The implemented system will organise and manage the companies' workflow data. New features and systems will therefore be incorporated in its architecture, which is why it should be extensible. In order to keep the system functioning after the project, it should also be maintainable. And since the architecture will be developed further by other developers in the future, it should be highly flexible.

2.2.1 Extensibility

Extensibility is the most important design goal of this project. The system that will be created will have to be able to cope with P3D adding new systems to their company, effectively creating a new source of data that should be utilised. To be able to use this data in new and innovative ways, the system should allow for the creation of new features. These goals will be supported by two sub-goals: separation of data and functionality and extensive documentation.

Separation of Data and Functionality

By separating data and functionality, future developers that will work with the code can easily add new features, without having to change existing functionality or data structures. The same applies to adding new data or data structures to the system, the existing functionality does not have to be modified.

Extensive Documentation

Extensive code documentation not only makes it easier to work with the code for current developers, it also helps future developers get a good understanding of the code. Extensive architecture documentation, preferably with visual aids, also helps new developers in understanding the structure of the system. Both will help them create new functionalities and allow them to efficiently reuse existing ones.

2.2.2 Maintainability

Maintainability is the second most important design goal of the project. Future developers should be able to easily understand and work with the new code. In order to do this, certain sub-goals have been created: code quality, testability and modular independence.

Code Quality

The code should conform to the general code quality standards. This will help ensure the code is clean and consistent. High quality code makes it easier to read and understand the code, which will make it easier to maintain. All libraries and packages used should also adhere to these standards and be well maintained.

Testability

Improving testability will allow for more extensive testing. This in turn makes sure that the code functions properly, improves the understandability and saves developers time.

Modular Independence

Creating a system where functionalities are modular improves the aforementioned testability. This in turn helps developers pinpoint where an error occurs. Finally, it becomes easier to replace and update parts of the system without affecting the rest of the architecture.

2.2.3 Flexibility

In order to create a system that can be extended in the future, developers should not make too many assumptions about the code. In order to still create a well-functioning system which is able to adapt and be expanded in the future, the code base should be flexible. In order to make this happen, two sub-goals have been created: modular independence and extensive code communication.

Modular Independence

As mentioned before, a system that is modular makes it easier to update and replace parts of the system without affecting other parts of the code. This will give future developers a lot of flexibility when implementing new functionalities. It is important to make sure these modules do not become too large as this would defeat the purpose of modular independence.

Extensive Code Communication

Extensive code communication goes hand-in-hand with modular independence. To create new functionalities from existing ones in such a system, it is important to have support for good communication between different modules. This way, existing functionalities are accessible and approachable for future developers.

2.3 Requirements

This chapter outlines the functional and non-functional requirements that have been set up in cooperation with the client. The requirements have been divided into four categories using the MoSCoW method (Waters, 2009).
All of the requirements will be explained as to what they mean and how they can be defined as completed using success criteria.

2.3.1 Must Have

The first category of the MoSCoW method is the 'Must Have' category to which all requirements belong that are necessary to deliver a viable end-product. Therefore, the requirements in this section also have the highest priority in our project planning.

1. **Reading (existing) client data from the Enterprise Resource Planning (ERP) system**: it must be possible to extract client data from the ERP system currently in use by P3D and use and process that data within our system. The success criterion for this requirement can be defined by the presence of a data coupling between the system and the current ERP system.

2. **Project planning can be digitally entered by users**: it must be possible for users of the system to modify the project planning, that is, users must be able to assign employees to tasks and specify in what time period the corresponding employee will work on said task. The success criterion for this requirement is the functionality as described for this item implemented in a Graphical User Interface (GUI).

3. **Project planning can be digitally viewed by users**: apart from the ability to enter the project planning by users, the project planning must also be visualised within the system to give users a clear understanding of the project planning and during what time periods certain employees are available to work on certain tasks. This is completed when a GUI is present that shows an overview of the current week and boxes that correspond to employees working on a certain task at a certain time.

4. **Multiple projects can be digitally viewed at the same time**: The system should allow multiple project plannings to be viewed at this same time. This is completed when a GUI is present that shows an overview of the current week and the planning of projects scheduled for that week.

5. **The system should reflect the project status**: within the system there must be a visual representation of the project status for any order. The project status of an order within P3D can be anything from 'Product in Printer' to 'Product Completed'. This requirement is a success if there is a visual representation of the status of any order within P3D integrated into the GUI. Furthermore, the order status should correspond to the actual current order status, so users must be able to change the order status via the GUI as well.

6. **System (planning) should show the starting of drying material**: it must be possible to visualise different order statuses as explained in the previous requirement. Starting of drying material is one of those statuses and therefore it must be visualised as well. The success criterion for this requirement is the presence of a way to visualise the starting of drying material within the system.

7. **System (planning) should show the ending of drying material**: similar to the previous requirement, the system should also show when the drying of material is ending. The success criterion for this requirement is again the presence of a way to visualise the starting of drying material within the system.

8. **User can input the start of material drying**: it must be possible for the users of the system to input when the material drying process will start. The success criterion for this requirement is the presence of an input part in the GUI of the system that allows the user to put in this data.

9. **System (planning) should show the starting of production**: a visual status that must be displayed in the GUI of the system is when the production of a new part or mould starts. The success criterion for this requirement is the presence of a visual representation of this status within the GUI.
10. **System (planning) should show the ending of production**: just like the previous requirement was about the start of the production of a part or mould, the system should also show when the production has ended. The success criterion is the presence of a visual representation of this status within the GUI.
11. **User can input the start of production**: it must be possible for users to input when the start of production has begun. This requirement is completed when there is an option within the GUI of the system for users to input this data.
12. **User can input the end of production**: it must also be possible for users to input when production of a part or mould has ended. This requirement is completed when there is an option within the GUI of the system for users to input this data.
13. **Shipping status should be reflected**: the GUI of the system must be able to show whether an order has been shipped or is awaiting to be shipped. This requirement is met when there is an option within the GUI to show this order type.
14. **Invoice status should be reflected**: just like the shipping status there must also be a part in the GUI that shows what the invoice status is. Completion of this requirement is reached when there is a part of the GUI that shows the corresponding invoice status of an order.
15. **The user can access the system from anywhere**: to allow the system to be used from other places than just the P3D office, at home for example, it must be possible for users to log in from any place. This requirement is met when there is a possibility for users to access the system via a URL, allowing remote access.
16. **User can log in to the system**: when remote access is wanted there is a need for users to authenticate themselves by logging in to the system. This must be present for the system to work in a safe way. This requirement is completed when it is possible to login through the GUI of the system.
17. **User can log out off the system**: if users can log in to the system they must also be able to log off from the system via the GUI. This requirement is met when the GUI offers users a way to log off which requires them to authenticate themselves the next time by logging in again.
18. **System should reflect correct material stock throughout the process**: the system must show how much materials are left for the 3D printers and the other materials that are needed during the entire production process. This material stock must be shown in the GUI of the system. This requirement is reached when the system has a part of the GUI that shows the material stock.
19. **System (planning) should show the starting of rework**: when a product is finished it might need some rework which is the finishing touch stage of the creation of new products. The system should show when a product has started the rework phase. This requirement is met when a part of the GUI of the system is dedicated to showing when a product has entered the rework phase.

20. **System (planning) should show the ending of rework**: just like the previous requirement was about the system showing when a products has started the reworking phase there also must be a way to show a product has finished the reworking phase. This requirement is met when the system can show when a product has finished rework within the GUI of the system.

2.3.2 Should Have

The should have's within the MoSCoW method are requirements that offer significant value to the end-product, but the end-product does work without these requirements.

1. **System should reflect the printing status**: the system should be able to show the correct printing status in its GUI. This is completed when there is a part of the GUI that is dedicated to showing this printing status.

2. **System can suggest the overall time of production**: it should be possible for the system to make an overall suggestion for the production time based on parameters like the amount of products to print and their size. This requirement is met when the system shows a time suggestion in the GUI and calculates this suggestion automatically based on all the input data for a certain order.

3. **System should reflect the print cleaning status**: after a product has been printed it needs to be cleaned. Within the system the status of cleaning a product should be reflected. This is completed when there is a part in the GUI that shows the correct product cleaning status.

4. **System should reflect the status of making the sprue gate**: the sprue gate is used to inject the 3D print material into the mould. Within the system there should be a part of the GUI that correctly reflects the status of making a new sprue gate, this is also the success criterion for this requirement.

5. **Ordering status of steel moulds should be mentioned**: sometimes P3D uses steel moulds instead of 3D-printed moulds, when these are used the system should show the ordering status of these moulds. This requirement is completed when a part of the GUI of the system is dedicated to showing the ordering status of steel moulds.

6. **Quality control of moulds should be mentioned**: to ensure every product made by P3D is of the highest quality, P3D does quality control checks. This requirement is met when the system has a part within the GUI that shows the users whether or not an order is currently undergoing quality control.

7. **Historical project data should be saved for later use**: P3D would like to do analysis on past projects to gain knowledge about their system. This requirement is completed when data about old projects remains saved.

2.3.3 Could Have

The could have's represent requirements that only will be implemented when time within the project planning allows to do so.

1. **System should show invoice documents**: With every order an invoice is sent to P3D clients, these could be displayed within the GUI of the system. This requirement is completed when there is a part of the GUI in which users can view the invoices they have sent to clients.

2. **Reading Product Information from 3D File client**: whenever a new order is created a design is made based on a 3D file supplied by the client. It could be possible to create a functionality in the system that reads product information from this 3D file to save employees the hassle of having to manually enter the information. The success criterion for this requirement is the presence of the functionality to automatically read the 3D client file and process the file data.

3. **Project should be planned automatically by the system**: currently the project planning is done by hand, thus employees are planned and assigned to projects manually. This could be automated by having new orders automatically be assigned to employees that have time left at that moment. This would save the employees time and could therefore make the client, P3D, more efficient. This requirement is completed whenever an automatic project planning is present which also shows the planning in the system's GUI.

4. **The system sends alerts when the stock of material is getting low**: the system could have the ability to send automatic notifications to users upon noticing that the stock level of certain materials is getting low. This is already implemented in the ERP system used by P3D so the focus on implementing this requirement is rather low. Since the focus of this project is on centralising all different systems of P3D into one system, it is desirable to take this ERP functionality and put it into the new system. This requirement is met when the system has a built-in functionality that lets users set when certain stock levels of materials are low and sends a notifications to those users when these 'low levels' are reached.

5. **Historical project plannings can be viewed digitally inside of the system**: P3D would like to have insights on how past projects are planned and executed. This is completed when a GUI is present that shows an overview of the old project's planning and which employees worked on certain tasks at certain times.

2.3.4 Won't Have

The wont haves of the MoSCoW method are functionalities that are out of the scope of this project. These requirements will therefore not be implemented in this project.

1. **User can input finishing of material drying**: when a material is entering the drying part of the P3D process it will always be picked up after it has been dried. Therefore, it is not needed to manually enter that drying has finished after a user has picked up the material from drying. Thus this requirement won't be implemented in the system. However, in the future it could be automatically integrated into the system by coupling the drier to the system so that the drier can signal the system that drying has finished.

2. **The clients of P3D can login to the system and place new orders**: it won't be possible for clients of P3D to place new orders within the system by logging in. The system within the scope of this project is being designed to be used by employees within P3D. However, for future improvements the ability for clients of P3D to create accounts and place direct orders could be an enhancement to the system.

3. **The system automatically schedules and creates backups of its entire state**: during this project the ability to automatically schedule and create backups of the entire system won't be implemented because this would require an extensive amount of time. For future enhancements to the system this could however prove to be a requirement that makes the system more reliable to faults.

2.4 Implementation

This section is about technical and strategical decisions that are made before implementing the software. It contains all explanations for the choice of tools and technologies that will be used in this project as well as the architecture these tools will rely on.

2.4.1 Architecture

The structure of the system, also known as the software architecture, is the core of any software project. There are multiple software architectures to choose from, but two major architectures that are relevant for this project are the Monolithic architecture and the Microservice Architecture.

Monolithic architecture

In a monolithic architecture, all developers work on one application. This results in a fast build if the system is not too complex. However, a monolithic application has many disadvantages. A monolithic application is edited and deployed as a whole system. When a service must be updated, the whole system must be deployed and therefore all services must be deployed. This causes a bad experience for the current users. The same applies to a failure in one of the services. If a monolithic architecture is scaled, all the services will be scaled. This results in a larger consumption of server capacity (Villamizar et al., 2015).

Microservice architecture

A microservice architecture is a little bit more complex. Every service of the application is developed and deployed individually. Deploying the services individually has many advantages. Other services won't go down when a services is deployed. Services can be scaled individually when the demand of the service increases and if there is a new functionality, this can easily be added by adding a new microservice (Villamizar et al., 2015). The microservices communicate with each other via API calls. The most important design goal of P3D is extensibility. Microservices is therefore the best architecture for this project since this architecture is more accessible to extend. This was also recommended by the TU Delft coach that supervises this project.

2.4.2 Development Methodology

The development methodology is a set of practices, principles or a framework that provides guidance to plan, structure and control the processes during development. The chosen methodology has an impact on the goal of the project. Different types of methodology frameworks aim to achieve different interests of the software development and may change during a project. This section will detail to the reader the different methodologies that can or will be used.

Agile

Traditional and plan-based methods assume that problems are fully specifiable and that problems have an optimal or predictable solution. Waterfall development is a traditional method, where before starting the development all requirements should be specified and no adjustments are meant to be made on the way. Agile, in contrary to these plan-based and traditional methods, doesn't make this assumption.

Agile is based on the fundamental assumptions that projects are delivered using rapid feedback and change. The design is continuously improved during the project. This way the agile methodology tries to handle change and uncertainty during the development process. In this project requirements may be added or modified in the future, development must be quick due to the time constraint and work is done by a small team. Therefore, agile is the best choice for this project. (Dybå and Dingsøyr, 2008)

There are multiple development methods which have the same fundamental assumptions as the agile methodology. A few of these were considered before making the choice to manage the project using the scrum method. The agile methods considered are dynamic software development method (DSDM), feature-driven development, lean development, Rapid application development (RAD) and scrum. DSDM and lean are a better fit with bigger teams or multiple small teams with complex projects. Feature-driven and Rapid application development are more appropriate with critical systems or prototypes. At last a big part of the development team is already comfortable with the scrum principles. This concluded the choice for scrum. (Dybå and Dingsøyr, 2008)

DevOps

DevOps combines development and operations using automated deployment. This approach focuses on continuous operational feature deliveries. it 'helps deliver value faster and continuously, reducing problems due to miscommunication between team members and accelerating problem resolution.' (Ebert et al., 2016). Agile works well in combination with DevOps, the small iterations of the agile methodology can be supported by the continuous deliveries of the DevOps methodology. Some tools are needed to get this high degree of automation. One tool is the build tool, which handles compiling, dependency management, running tests, etc. The other is a continuous-integration tool that continuously runs the build. Another advantage of using DevOps is the possibility for the client to continuously evaluate new feature deliveries, this will be handy during this project.

2.4.3 Technologies/Tools

This section will explore possible tools and technologies that support the development process and that fit the architecture and methodology of the project.

Docker

The microservice talked about earlier must run on different machines. Instead of running each microservice on a virtual machine, there is a faster and more efficient method: the microservices can be containerised. A container doesn't have its own operation system and is therefore more lightweight. A useful tool to do this is Docker. Docker is a lightweight application which is specialised in containerising and deploying applications. Each microservice is transformed into an Docker image. From these images, Docker containers can be deployed (Turnbull, 2014).

Kubernetes

In the future, the demand on the system might increase. A single container might not be enough if it is used by many users. Kubernetes can wrap up containers in a new package called a pod. The Kubernetes software can distribute the workload and scale each pod as many times as needed (Vohra, 2016).

Gitlab

This project will be developed by five students. To make sure that the project will always contain stable code and that every student can share their code, there must exist some kind of continuous integration. GitLab is a development platform with built-in CI/CD. In GitLab there is also the possibility to register containers and use Kubernetes to cluster, scale and deploy those containers.

Programming language

The codebase of this project is primarily going to be written in Python. Python is currently popular and is actively developed (Fuchs et al., 2020). Because of these facts, by developing in Python future development teams that want to adjust or extend can still program in a relevant language. Languages as Java, PHP etc. where considered as well, but had less promising advantages. Java and PHP are less popular in the current development community.

Framework

By making the choice of the programming language Python, the available frameworks were reduced to Flask and Django. Where Flask is a lightweight web framework and Django a full-stack framework (Ashley, 2020). Since the Microservices architecture is decided to be used for this project, a full-stack framework for these services would be too much. The Flask framework could provide all the necessary features for the services while having the advantage of remaining lightweight and is therefore the chosen framework for this project.

GraphQL and REST API

GraphQL and REST API are two architectural concepts for calling an api and retrieving data. GraphQL uses its query language to tailor the request and retrieve the exact data needed, it operates over a single endpoint. The flexibility of these tailored requests could benefit our extensibility design goal, where new systems could tailor their requests to get the data they need.(Brito and Valente, 2020)

The REST API architectural concept provides robustness, simplicity and has become the industry standard for deploying APIs.(Brito and Valente, 2020) These elements benefit future development more than the benefits of GraphQL, which has a greater complexity and learning curve. Therefore the REST architectural concept is the better overall choice for this project.

2.4.4 Testing

This section will elaborate on the testing tools en methods. For each level of the development process a testing tool is chosen. Starting with unit testing, that supports the developer during programming. Ending with End to End testing, where the client can experience the full product.

Unit testing

Unit testing is used to test the functionality of small snippets of code, independent of other functionality. This type of testing ensures these units of code meet there design and behave as intended. Python has a library for unit testing, it provides methods to assert output of different functions. This makes unit testing the codebase easily possible.

Integration testing

Integration testing is the principle of making sure different components of a bigger system work well together. Since this project is based on a Microservice architecture, it is viable that different parts of our system communicate flawlessly. Integration testing is therefore needed in this project.

End to End testing
As the core features of this project do not necessary have a user interface, end to end testing is not considered to be a high priority. End to end testing will be therefore done through demos which is preferred over using an end to end testing tool, such as selenium. Selenium is a tool where a path through the user interface can be recorded and replayed as a test. This tool takes time to configure for new features and is only useful for testing repetitively, concluding that testing using regular demos is a better option.

3 | Design

3.1 Overview

This chapter contains the design of the software architecture. It describes the challenges that the team encountered while implementing the architecture. Furthermore, it contains the changes that the team made during the project. Finally, the final design is shown and described.

3.2 Challenges

This section will elaborate on the design challenges that were encountered during the project. The design was based on the design goals that were constructed in the research phase of the project. The main design goals were extensibility, maintainability and flexibility. These are general concepts but are good guidelines upon designing the system.

3.2.1 Extensibility

Extensibility as a design goal has implications on the integration with new systems and the introduction of new features within a system. One of the main challenges that was encountered was coupling the project with the Entreprise Resource Planning (ERP) system of P3D. The challenge here was not only a technical one but a communication one as well. On the technical side, it was a challenge to get familiar with the structure of the data of the ERP system and to design the system in a way that neatly connects the data of the ERP to the Data Management System (DMS). The communication challenge here was to explain the necessity of access to the database of ERP and getting information about the possibility of achieving this. The integration of the ERP system and thereby achieving the main design goal was the biggest challenge of the project.

The system is designed to have a separate microservice for each part of the system that manages different types of data or different functionalities. When needing new data sources or functionalities, one can add new microservices which can be called by existing microservices or user interfaces. In this way, the system is easy to extend. The challenge here is to design the endpoints of the microservices in a way that future developers can understand it without too much effort. Future developers will then be able to adjust the functionality or include already existing functionality in new systems.

Not only the API of the system is extendable, the front-end as well. The system is designed to be able to use multiple front-end interfaces. By using a Nginx(Reese, 2008) reverse proxy, new front-end interfaces can be added and linked together. When future developers want to build a new user interface (UI), it can be used next to the old system or a transition can be done to slowly migrate to a new front-end.

3.2.2 Maintainability

One of the other design goals was maintainability. To design a system with a high degree of maintainability it needs to have great code quality, high testability, and good separation of functionality. The challenge here was to keep the duplication of code and the use of resources to a minimum. With a microservice based architecture, as each service is an application that can run on its own, many of them need the same functions and resources. Here the design choice was made to create a custom package library that is used by all API microservices. This way the shared functions can be maintained in one place. Another challenge with maintainability was to be able to update the operating system (OS) on which the microservices run. Because the microservices are designed to run in a container, the image of the OS can be changed on each deployment, making the maintenance of the resources and packages easy.

By designing the endpoints of the microservices with an eye for simplicity and by extensively using the functionality of the custom package, the microservices have great code quality and a high degree of testability.

Another design challenge was to make sure the system is always available as the client is heavily going to rely on the system. This was achieved by using the container orchestration tool, the load balancer and the rollout functionality built into kubernetes (Burns et al., 2018).

The load balancer can increase the available resources on the server when the load is high and will decrease the capacity to keep the cost low. On a new software release, the rollout functionality makes sure that the new version of the application is started before the old one is stopped. This way requests during the rollout can still be handled by the old version and users will not notice a performance decrease.

3.2.3 Security

The last design challenge was security. Because the system is going to be used with company data the system must be secure. One of the requirements of the client was the availability of the system from anywhere, this collides with making the system as secure as possible. A more desired situation from a security standpoint would be, a system that is only available on the VPN of the company. Because this is not the case, there are several security design measures taken during this project. To protect the data, that is the databases, the access to it is limited to the internal IP address in the used Google Cloud environment. This way only the services that are on the cluster can access it. The cluster itself is protected by an ingress Nginx(Reese, 2008) controller which handles all traffic from outside the cluster. This is the single point of entrance, making the attack surface minimal. Another challenge of security is authentication. This part of the security is outsourced in the design. There are several good 3rd party authentication providers. Combining the authentication providers with an OAuth(Hardt et al., 2012) microservice on the cluster solves the problem of authentication.

3.3 Changes Made Throughout The Project

At the beginning of the project, some functionality goals were set with the client. The design goals were based on those agreements. However, some agreements turned out to be impossible or too hard to implement. Therefore, some changes were needed in the design during the project.

ERP Connection

One of the main functionalities of the product was making a coupling with the existing ERP system of P3D. Some microservices would connect to the external ERP database and retrieve the data that we could use in the system. However, the company that hosts the ERP database explained at the beginning of January, that an external connection with the ERP database was not possible due to security issues. Because of this issue the DMS got a big adjustment since P3D's project data was needed in the system. The solution was an extra database to replace the ERP database. This database contains all the information about P3D's projects that the system currently uses. Since project data is not automatically in this new database, the system needs to have an insert functionality. Project data can therefore now be inserted via a front-end into the replica of the ERP database. In the future, P3D wants to connect to a cloud-based ERP system. When this is done, the system can connect to the real ERP database and the project data is automatically available in the system.

Security Design

As mentioned in 3.2.3, the security design is very hard to implement. The first design was to have a homepage with a button to log in. Users were only allowed to visit the the homepage. For other URLs, the user had to be authenticated. The authentication check occurred in the Nginx(Reese, 2008) microservice and since the system uses a microservice architecture, every microservice needs to have a security check. The old design is shown in figure 3.1.

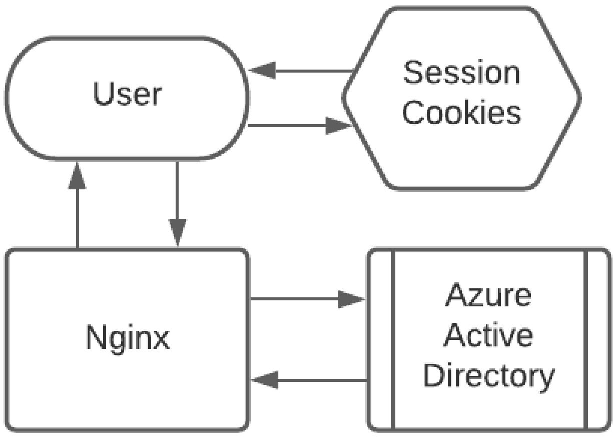


Figure 3.1: The old security design

However, it appeared easier to implement the security check at the entrance of the system. Now, every user has to be authenticated when visiting the front-end. If the user isn't authenticated, he will get redirected to the Azure login. The current log-in design is shown in figure 3.2

Activity Input System

Another main functionality was changing the statuses of processes in P3D's workflow. The client wants to keep track of the activities of a project. The first design was an input system that the user could use to change status, start and end date/time of an activity. However, after exploring the planning framework, it appeared to be unnecessary since this was easier to implement in the planning front-end. After double-clicking on an activity in the planning page, the activity can be marked as done or active. This solution gives a better overview for the user if he wants to change a date. Now, he sees immediately if an activity would overlap with another if he changes this date.

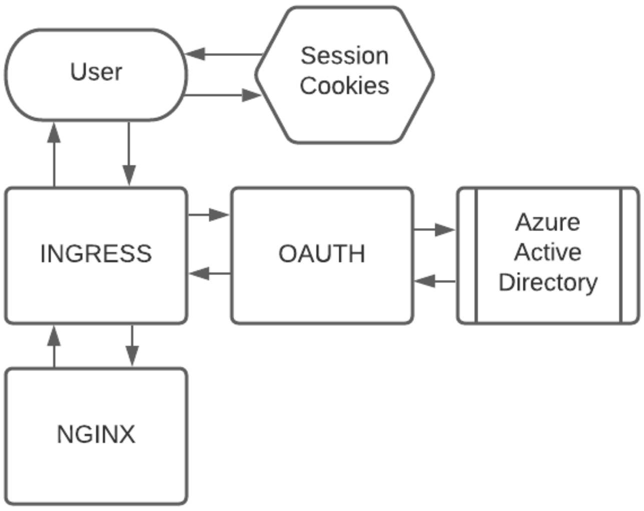


Figure 3.2: The new security design

Cert-Manager
In the start of the project, we knew that the website should have a secured connection but there was no design made for this problem. HTTPS is a secured version of an HTTP network. An HTTPS network encrypts the communications between the client and the system. Therefore, an attacker is unable to read the data sent over the network. In order to have an HTTPS connection, the system needs an SSL certificate. However, SSL certificates have an expiry date. This is needed to recheck the validation of the website. Since P3D doesn't have any IT employees, the SSL certificate would expire and the website would not be secure anymore. Therefore, a new microservice called Cert-Manager is installed. This microservice checks if the SSL certificate is still valid and will create a new certificate otherwise. This way, the website will always have an HTTPS connection.

3.4 Final Design Summary

Our final design is a microservice based architecture. This architecture makes sure that the extensibility and maintainability design goals are satisfied. Some microservices are connected to the (mocked) ERP database. When the ERP database changes, these microservices will be able to connect to the other database with minor changes. The other back-end microservices will connect to a planning database where all data present in the front-end will be saved. The ingress microservice serves as an open external IP. The OAuth (Hardt et al., 2012) microservice makes sure that every visiting user is authenticated and the react microservice is responsible for the front-end. When a user is logged in, it will be redirected to the Nginx (Reese, 2008) microservice. This microservice is a reverse-proxy, which will redirect the user to the correct microservice. The final design can be found in figure 3.3. More details about these microservices can be found in 4.2.1.

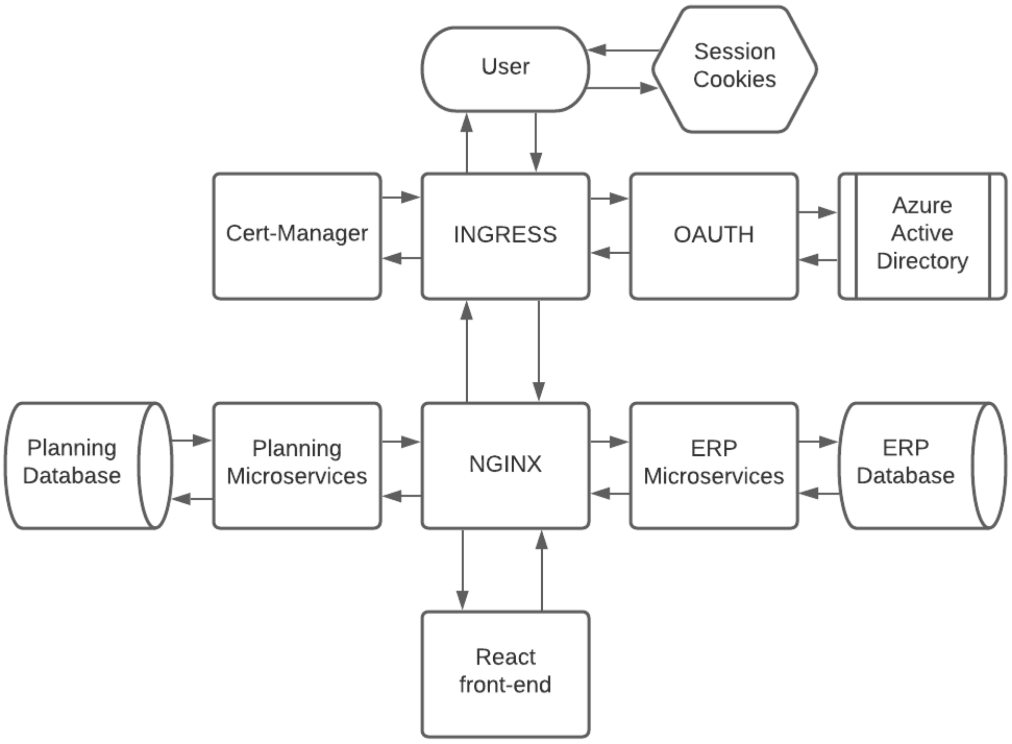


Figure 3.3: The final design

4 Implementation

4.1 Overview

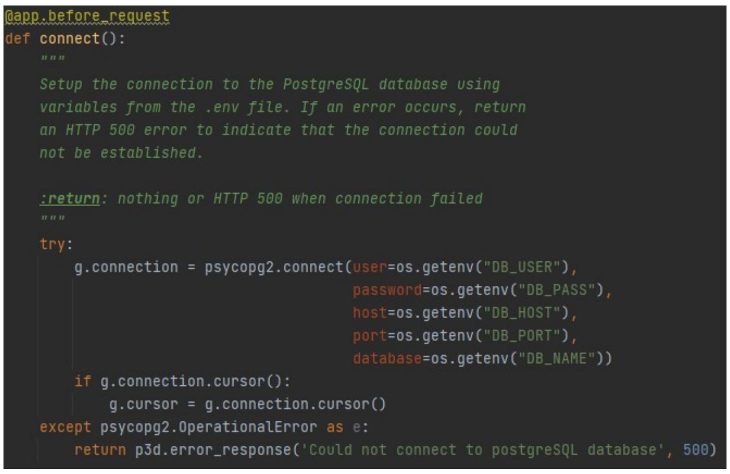
This chapter will cover the implementation of the final product. This implementation is based on the design of the system and can be divided into several parts: a front-end, a back-end and the deployment. The front-end is represented by a proof of concept and will be talked about in section 4.5. The back-end is represented by the microservices and will be talked about in section 4.2. Finally, the deployment will be covered in section 4.3.

4.2 Microservices

For this project a microservice architecture, as discussed in the design part of this project, was used. It was decided to implement this type of architecture after recommendation from the TU Delft coach and because it suits the usage of this system well. This section will give a general overview of the microservice implementations, discuss all microservices that have been created and how code duplication has been minimised by creating a custom Python package (P3D, 2021).

4.2.1 Microservice Implementation Overview

The microservices have been implemented using Flask, which is a framework specifically designed for microservices made in Python. The product uses six different microservices, which will be discussed in the upcoming sections. Each microservice contains its own routes which can be called upon. This then results in data being returned or entered into the database. Before a route is called, a database connection is made using the method shown in figure 4.1.



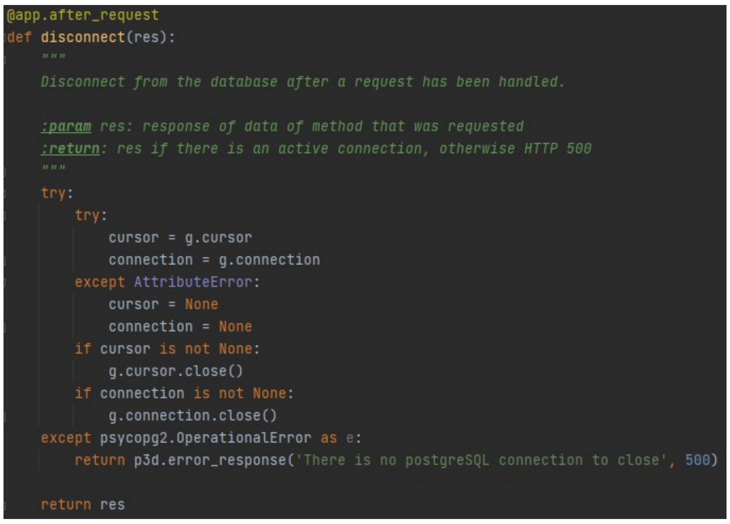
```python
@app.before_request
def connect():
    """
    Setup the connection to the PostgreSQL database using
    variables from the .env file. If an error occurs, return
    an HTTP 500 error to indicate that the connection could
    not be established.

    :return: nothing or HTTP 500 when connection failed
    """
    try:
        g.connection = psycopg2.connect(user=os.getenv("DB_USER"),
                                        password=os.getenv("DB_PASS"),
                                        host=os.getenv("DB_HOST"),
                                        port=os.getenv("DB_PORT"),
                                        database=os.getenv("DB_NAME"))
        if g.connection.cursor():
            g.cursor = g.connection.cursor()
    except psycopg2.OperationalError as e:
        return p3d.error_response('Could not connect to postgreSQL database', 500)
```

Figure 4.1: Microservice connect method for database connection

After a request to a route within a certain microservice the connection to the database is closed using the disconnect method shown in figure 4.2.

Both the connect and disconnect method are present in all of the microservices. The reason for this will be further explained in section 4.2.3.

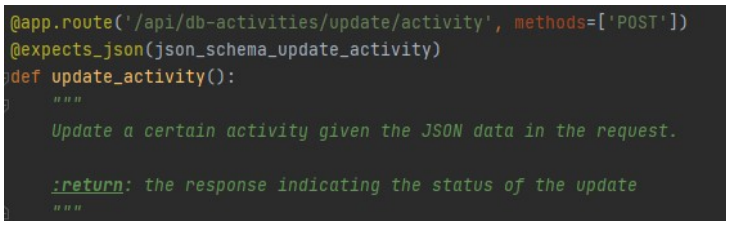
Another important aspect of the microservices is that data validation is being done inside each



```python
@app.after_request
def disconnect(res):
    """
    Disconnect from the database after a request has been handled.

    :param res: response of data of method that was requested
    :return: res if there is an active connection, otherwise HTTP 500
    """
    try:
        try:
            cursor = g.cursor
            connection = g.connection
        except AttributeError:
            cursor = None
            connection = None
        if cursor is not None:
            g.cursor.close()
        if connection is not None:
            g.connection.close()
    except psycopg2.OperationalError as e:
        return p3d.error_response('There is no postgreSQL connection to close', 500)

    return res
```

Figure 4.2: Microservice disconnect method for closing the database connection

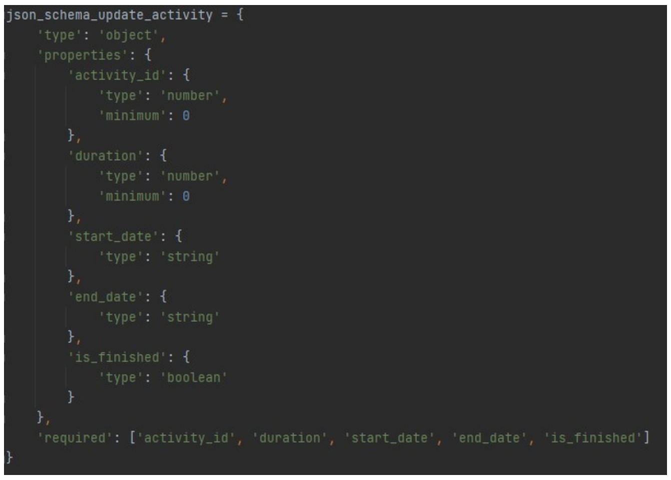
microservice. This project uses a Python data validation package to ensure this. This package, flask-expects-json, can be found here: `https://pypi.org/project/flask-expects-json/`. This package allows all request data to be validated before it is being processed within the method of the respective route. A route within Flask annotated with @*expects_json* is a route that uses the validation package, an example is given in figure 4.3.



```python
@app.route('/api/db-activities/update/activity', methods=['POST'])
@expects_json(json_schema_update_activity)
def update_activity():
    """
    Update a certain activity given the JSON data in the request.

    :return: the response indicating the status of the update
    """
```

Figure 4.3: Example Route That Uses The Validation Package

The variable within the @*expects_json* field of the route in figure 4.3 is a JSON schema that is used by the package to define what the structure of the payload must be in order to be valid. This schema is shown in figure 4.4.



```
json_schema_update_activity = {
    'type': 'object',
    'properties': {
        'activity_id': {
            'type': 'number',
            'minimum': 0
        },
        'duration': {
            'type': 'number',
            'minimum': 0
        },
        'start_date': {
            'type': 'string'
        },
        'end_date': {
            'type': 'string'
        },
        'is_finished': {
            'type': 'boolean'
        }
    },
    'required': ['activity_id', 'duration', 'start_date', 'end_date', 'is_finished']
}
```

Figure 4.4: JSON Schema Used For Validation

As can be seen in the schema of figure 4.4, the valid input for the route shown in figure 4.3 is a JSON with five fields: 'activity_id', 'duration', 'start_date', 'end_date' and 'is_finished'. For numbers like 'activity_id' it can be specified if the number must have a minimum or maximum value or whether it must lie between a certain interval. In this case since all IDs in the database used are positive values, the minimum value of 'activity_id' must be zero.

4.2.2 Created microservices

Our project uses 10 different microservices of which 7 have been coded by ourselves:

1. **db_activities**: this microservice is connected to the database of the created project and handles the deletion, creation and updating of tasks within projects.
2. **db_clients**: this microservice is connected to the (mocked) ERP database and handles the retrieval of clients from the database.
3. **db_material_stocks**: this microservice is connected to the (mocked) ERP database and retrieves the stock information of the materials used by P3D.
4. **db_projects**: this microservice is connected to the (mocked) ERP database and handles all data modifications that are done on projects in the user interface.
5. **planning**: this microservice is not connected to a database but calls other microservices to combine and process data to be shown in the planning part of the user interface.
6. **project_input**: this microservice is not connected to a database but calls other microservices to combine and process data to be shown in the project part of the user interface.
7. **Front-end (React)**: this microservice is used to generate the front-end of the DMS. More detail can be found in chapter 4.5.
8. **NGINX (Reese, 2008)**: This microservice routes every incoming request to the corresponding microservice.
9. **OAuth (Hardt et al., 2012)**: this microservice checks on every incoming request if the user is authenticated.
10. **Certificate Manager**: This microservice checks if the SSL certificate is still valid. Otherwise, it will generate a new one.

A full description of what all the endpoints inside each microservice do can be found in the API documentation found at this link: `https://documenter.getpostman.com/view/14222885/TVzYetMk`

4.2.3 Python Package

One recurring problem with the microservice architecture used during this projects was that a lot of code was duplicated. This caused problems at the beginning since multiple methods were copied to all six microservices. When a change was made to one method it had to be repeated in all microservices. After the first code upload the aim was therefore to reduce the amount of duplicated code, which eventually led to the decision to make a custom Python package.
Since this project used a GitLab repository that was hosted on the `gitlab.ewi.tudelft.nl` domain it was not possible to use the GitLab package management, and therefore it was decided to create a public Python package. This package was named 'bep-framework' and can be found via this link: `https://pypi.org/project/bep-framework/#description`. The usage of this package meant less code duplication occurred and the code was easier to maintain. One small problem was that the methods 'connect' and 'disconnect' as discussed in section 4.2.1 could not be placed in the package since this did not create a connection in the correct place, which is why each microservice still has those two methods. Because the Python package was designed and created in such a way that no sensitive information has been exposed in the code, uploading the package to PyPi does not create any security threats.

4.3 Deployment

To build and deploy all changes automatically to the server, the project uses continuous integration and deployment. This section describes the tools that were used to set this up.

4.3.1 Docker

Each microservice is contained in a docker container. A container is a shell or OS in which the application of the microservice runs. This container provides all the libraries and packages needed by the application.To get a container for the application one needs to create a Dockerfile. This Docker file contains some commands by which the docker application can build a docker image. This set of instructions contains the base image, what packages or modules need to be installed, and what instructions need to be executed when the image is converted to a container (when it starts to run).

The base images of the API microservices are python images and the front-end microservice uses a node-js alpine base image. These base images only have the required libraries and no other software on them, this contributes to a small container image. By having less software in the container the possibility of an exploit decreases.To even decrease the size of the containers further, a multi-stage building can be used. With multi-stage building, only the bare minimum remains in the last stage. By using this during the project the front-end image decreased from 900 MB to 125 MB in size.

The project uses two different docker files for each microservice excluding the Nginx (Reese, 2008), OAuth (Hardt et al., 2012), ingress controller (Burns et al., 2018), and cert-manager. These two are the production and development docker files. The development docker file is used for development by the software developer on its machine. The development docker file has more tools installed and the file system of the container is connected to the file system of the host machine by making use of docker volumes. When the developer changes some file on its machine the docker container can use this changed file directly, without needing to rebuild.

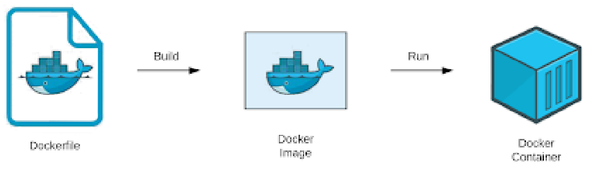


Figure 4.5: The stages of docker (neilkillen.com)

4.3.2 Kubernetes

Kubernetes is a container orchestration tool that manages the deployment, scaling, and networking of containers. The Google Kubernetes Engine that is used in this project is a cluster consisting of multiple nodes. The nodes are the workers of the cluster, they provide the CPU and memory. Each deployed container runs inside a Kubernetes pod, the pods run directly on the nodes. By using a Kubernetes service, multiple pods with the same container can be linked together, making the scaling of a deployed container possible without static IPs. Ingress is used to expose the cluster to the outside. Kubernetes is configured by YAML files, each container deployment, service and ingress has its own YAML file.

Some of the containers need production-specific environment variables and not all of them can be specified within the config file. Some of these variables are too sensitive to commit to the repository. These variables can be added to the secret config service of the Kubernetes engine. The secret environment variables are loaded when starting the deployed container.

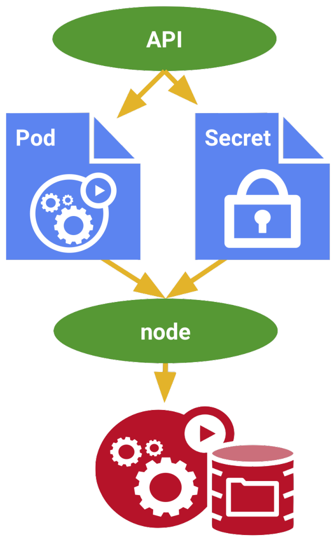


Figure 4.6: The stages of kubernetes (cloud.google.com)

4.3.3 Container registry

When deploying, the Kubernetes cluster needs to fetch the Docker containers from an online container registry. This is done using Gitlab. The containers are pushed to a container-registry and the Kubernetes cluster fetches the containers from that server.

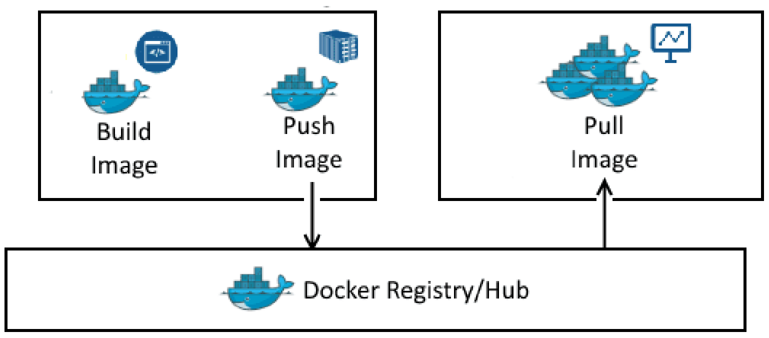


Figure 4.7: Container registry (https://www.itzgeek.com/)

4.3.4 Google Cloud

The product is hosted on the Google cloud environment. The choice was based on a couple of factors:

- The team members were familiar with the Google cloud environment.

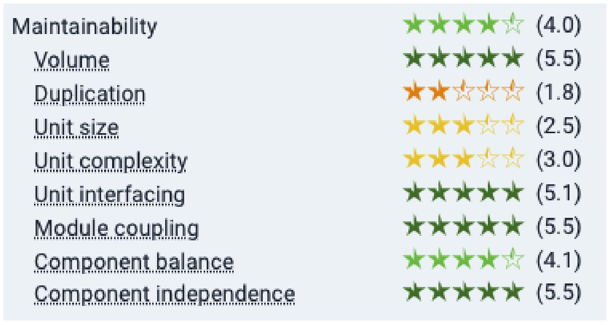
- Google cloud has a very structured Kubernetes management. The console has many functions, the most used functions in this project are:
  - The number of pods for each docker image.
  - The requested CPU of the nodes can be viewed
  - The statuses of the containers are displayed and if a container fails, the error logs are displayed well.
  - The secrets for the project are displayed.
- The google cloud console comes with a built-in terminal. In this terminal, the kubectl command is available for all Kubernetes functionalities.

4.3.5 Docker-Compose

Docker-compose is used by the developers to start all the microservices in a stack. This makes the integration of the different services easier as they all start by using one command, they can use the same virtual defined network and can have a single entry point. Nginx provides the single entry point and redirects the request to the correct microservice. During the project, the Kompose tool was used to convert the docker-compose configuration to the Kubernetes configuration.

4.4 Code feedback Software Improvement Group

As part of this project its evaluation process, a code analysis was performed by the Software Improvement Group (SIG). SIG tested the maintainability of the code based on 8 different factors: volume, duplication, unit size, unit complexity, unit interfacing, module coupling, component balance and component independence. Figure 4.8 shows the score for each of these factors after the first code submission.



Maintainability	★★★★☆	(4.0)
Volume	★★★★★	(5.5)
Duplication	★★☆☆☆	(1.8)
Unit size	★★★☆☆	(2.5)
Unit complexity	★★★☆☆	(3.0)
Unit interfacing	★★★★★	(5.1)
Module coupling	★★★★★	(5.5)
Component balance	★★★★☆	(4.1)
Component independence	★★★★★	(5.5)

Figure 4.8: Scores from the first SIG upload

As can be seen, 3 factors have a score below 4: duplication, unit size and unit complexity. It was necessary to improve these and they were therefore given priority during the continuation of this project. This section will now outline how the code was restructured to improve the low scores present in figure 4.8.

- A low duplication score occurs when the same code is used in different parts of the codebase. Code duplication mainly occurred due to the microservice architecture of this project. As stated in 4.2.3, this low score was improved by building a custom python package which could be called in each different microservice, avoiding duplication.

- A low unit size score occurs when different units (functions, constructors, etc.) inside the codebase are composed of too many lines of code. In this project, this occurred in the back-end as well as the front-end. In the front-end, helper classes were created to split up large components. In the back-end, unit size was reduced by using the custom python package and by paying special attention to the size of functions and split them if necessary.
- A low unit complexity score occurs when functions and methods have too many execution paths. This issue was solved similarly to the low unit size score by breaking up methods and functions in the back-end and front-end.

4.5 Proof of Concept

This project was heavily focused on designing and implementing an all-round data management system for the company P3D. To prove this has been achieved and the system could actually be used in real-life applications, a proof-of-concept was needed. This proof-of-concept is used to show to the client that the requirements are being met as well give an example of the possibilities of the system itself. In this section, an overview of the different parts of the proof of concept (POC) will be given followed by a more detailed explanation of these parts individually. This section will only focus on the user interface of the POC as other parts are already discussed in previous chapters (see 4.2 and 4.3).

4.5.1 Overview of the POC

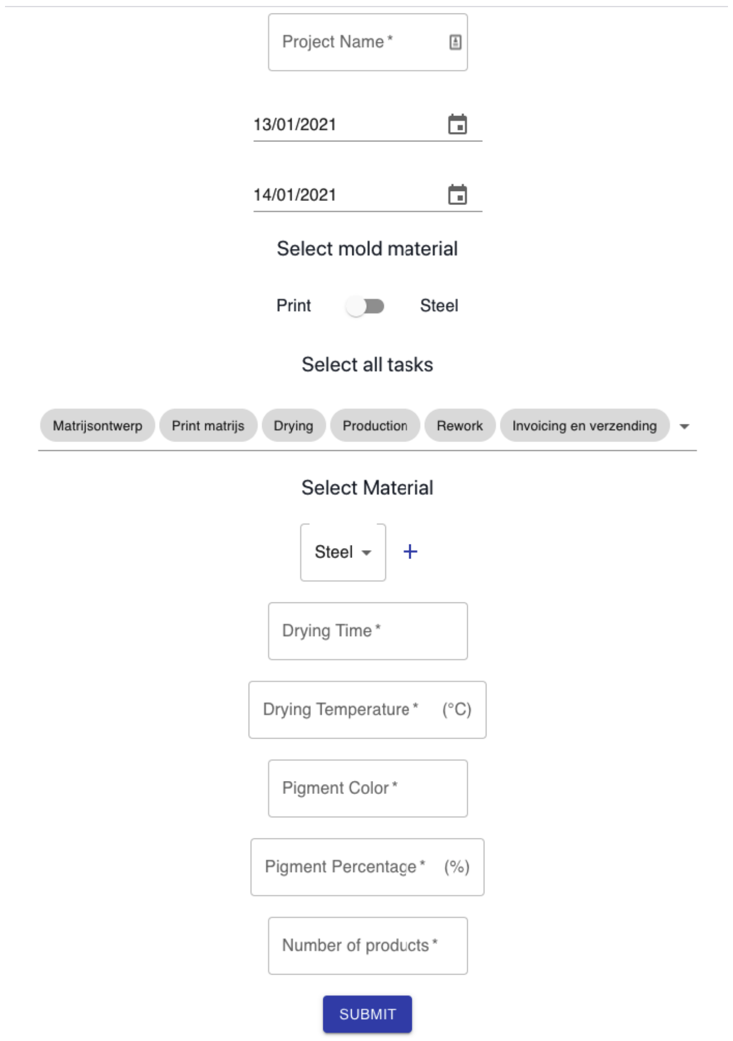
The POC comes in the form of a project planning tool designed specifically for P3D's internal workflow. As requested by the client, this planning tool shows projects that are currently running and gives the user the ability to insert, modify and delete projects as they so wish. Thanks to the microservice architecture and the way the system is designed, the insertion of projects could in the future be automated based on company data (by retrieving it from an ERP system for example). Currently, the POC provides a way for P3D's managers to visualise and modify their entire workflow planning from start to finish by manually inserting and editing projects.
On a more technical note, the POC makes use of different web development technologies namely React ((Aggarwal, 2018)) and Typescript ((Freeman, 2019)). After discussing with the client, it was decided that in order to facilitate the creation of the planning interface, a library would be used called DHTMLX Gantt (more information about this package can be found here: `https://dhtmlx.com/docs/products/dhtmlxGantt/`)

4.5.2 Project input

The first part of the proof of concept is the project input. The project input is a single page form that allows the user to input all relevant data of a project. Figure 4.9 shows how the form looks in the POC and which data can be entered.
Once a project is submitted, our back-end infrastructure takes care of calculating default values if needed and inserts the project to the database. The newly added project can then immediately be viewed inside of the project overview interface or the planning.

4.5.3 Project overview

The project overview section allows the user to view and make modifications to an existing project. This is needed when a project is already running but some information changed and needs to be reflected in all systems. The project overview is composed of three sections:


Project Name *

13/01/2021

14/01/2021

Select mold material

Print Steel

Select all tasks

Matrijsontwerp Print matrijs Drying Production Rework Invoicing en verzending

Select Material

Steel +

Drying Time *

Drying Temperature * (°C)

Pigment Color *

Pigment Percentage * (%)

Number of products *

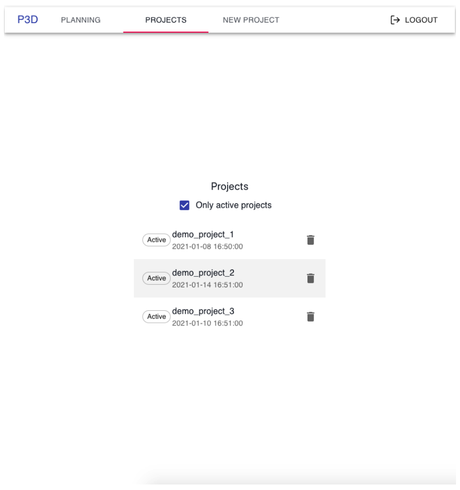
SUBMIT

Figure 4.9: Screenshot of the project input user interface

- *Project list and deletion (Figure 4.10)*: This section shows a list of all projects and gives the option to only show the currently active projects if wanted. It also enables the user to delete a project by clicking on the respective icon next to a project.
- *Project details (Figure 4.11)*: This section shows a single-page overview of all data belonging to a project. It also gives the user the ability to go to the project modification section (described below) which enables the modification of this data.
- *Project modification (Figure 4.12)*: This section enables the user to modify project or production data (depending on which button is pushed in the projects details section). It shows a basic form which can be submitted to save the newly update data.

4.5.4 Planning overview

The main component and most interesting part of the POC is the planning overview. It gives the client a direct overview of all running processes in P3D its workflow and enables him to individually plan and manage each process. It also notifies the user when certain processes are not finished even though their deadline has expired and does some simple resource management

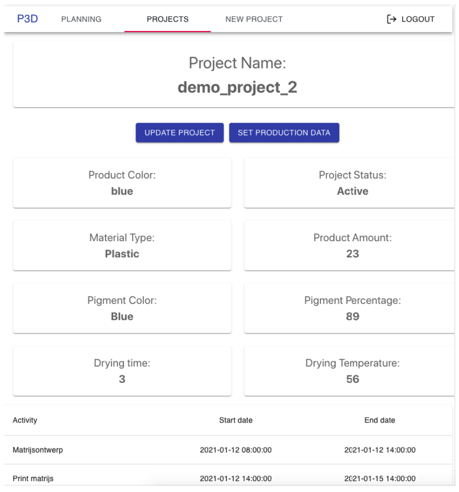


Projects

☑ Only active projects

Active	demo_project_1 2021-01-08 16:50:00	🗑
Active	demo_project_2 2021-01-14 16:51:00	🗑
Active	demo_project_3 2021-01-10 16:51:00	🗑

Figure 4.10: Screenshot of the project list and deletion user interface



Project Name:
demo_project_2

Product Color: **blue**

Project Status: **Active**

Material Type: **Plastic**

Product Amount: **23**

Pigment Color: **Blue**

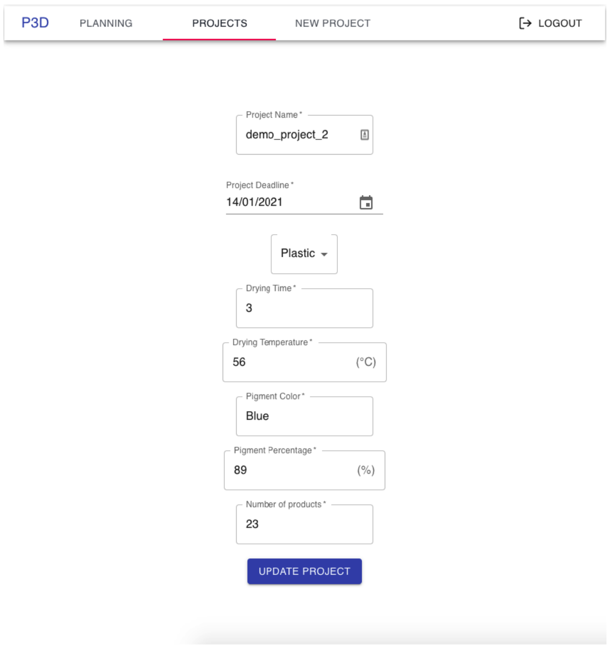
Pigment Percentage: **89**

Drying time: **3**

Drying Temperature: **56**

Activity	Start date	End date
Matrijsontwerp	2021-01-12 08:00:00	2021-01-12 14:00:00
Print matrijs	2021-01-12 14:00:00	2021-01-15 14:00:00

Figure 4.11: Screenshot of the project details user interface



Project Name *: demo_project_2

Project Deadline *: 14/01/2021

Plastic

Drying Time *: 3

Drying Temperature *: 56 (°C)

Pigment Color *: Blue

Pigment Percentage *: 89 (%)

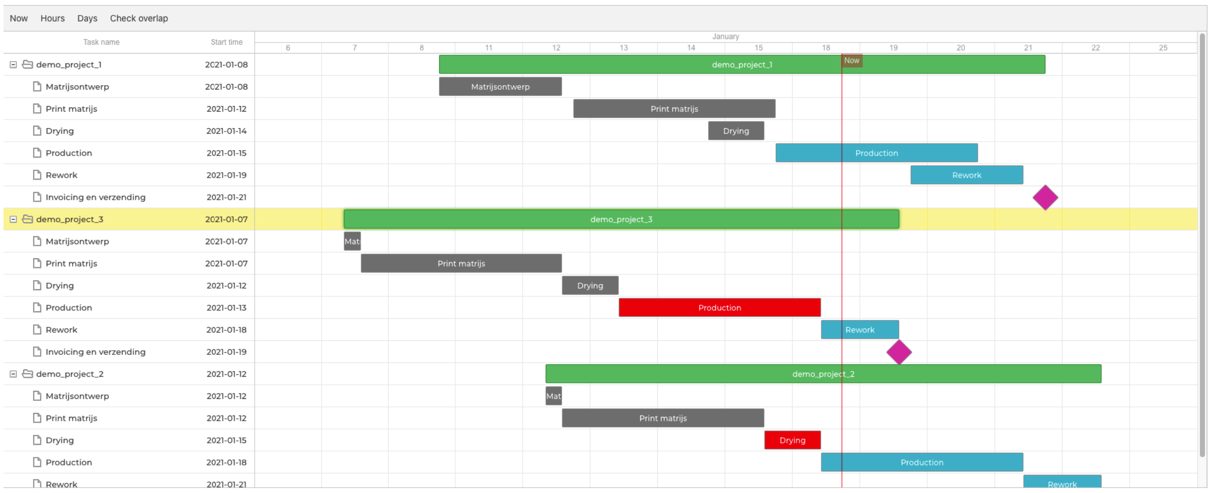
Number of products *: 23

Figure 4.12: Screenshot of the project modification user interface

by checking for possible overlaps of the same processes in the timeline.
As it is difficult to correctly represent such an interactive application on paper, this section will only briefly describe the different functionalities of the planning component in order to provide the reader with a general overview of what the application can offer.

4.5.5 Timeline

The planning component has a timeline which shows each process of P3D's workflow individually. Individual processes can be dragged and dropped to easily modify their start and end time or change their order inside of a project. The client requested to have the timeline accurately reflect the possibilities and limitations of P3D's workflow, in practice this meant the application has automatic constraints on the processes displayed, mainly dealing with ordering. An example of this functionality can be seen in figure 4.13.



Task name	Start time
demo_project_1	2021-01-08
Matrijsontwerp	2021-01-08
Print matrijs	2021-01-12
Drying	2021-01-14
Production	2021-01-15
Rework	2021-01-19
Invoicing en verzending	2021-01-21
demo_project_3	2021-01-07
Matrijsontwerp	2021-01-07
Print matrijs	2021-01-07
Drying	2021-01-12
Production	2021-01-13
Rework	2021-01-18
Invoicing en verzending	2021-01-19
demo_project_2	2021-01-12
Matrijsontwerp	2021-01-12
Print matrijs	2021-01-12
Drying	2021-01-15
Production	2021-01-18
Rework	2021-01-21

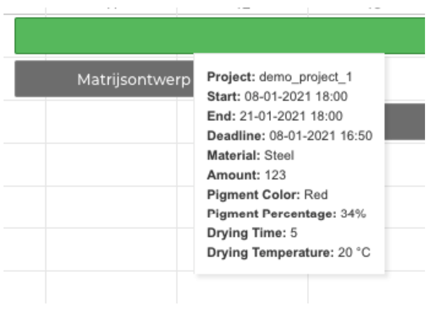
Figure 4.13: Screenshot of the planning timeline

4.5.6 Tool-tip information

The client requested to have a quick way to show all relevant information of a project. This has been satisfied by implementing a tool-tip which shows the information manipulated by the project input (4.5.2) and overview (4.5.3) parts of the POC. An example of this functionality can be seen in figure 4.14.

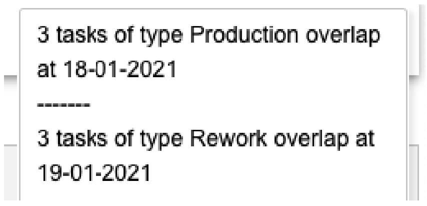
4.5.7 Visual cues

The planning implements multiple visual cues in order to show the user the different state of each process. These visual cues enable managers that work at P3D to immediately see which processes require the most attention and need to be handled straight away. Regular processes can take 3 colours : a process with a grey colour has been finished. A process with a blue colour is not finished yet but its deadline is still in the future. Finally, a process with a red colour has not been finished but its deadline is already overdue. Processes that are purple are treated as final deadlines and are used to reflect when a product needs to be shipped. An example of this functionality can be seen in figure 4.16.



Project: demo_project_1
Start: 08-01-2021 18:00
End: 21-01-2021 18:00
Deadline: 08-01-2021 16:50
Material: Steel
Amount: 123
Pigment Color: Red
Pigment Percentage: 34%
Drying Time: 5
Drying Temperature: 20 °C

Figure 4.14: Screenshot of the tool-tip functionality



3 tasks of type Production overlap at 18-01-2021
----
3 tasks of type Rework overlap at 19-01-2021

Figure 4.15: Screenshot of the overlap functionality

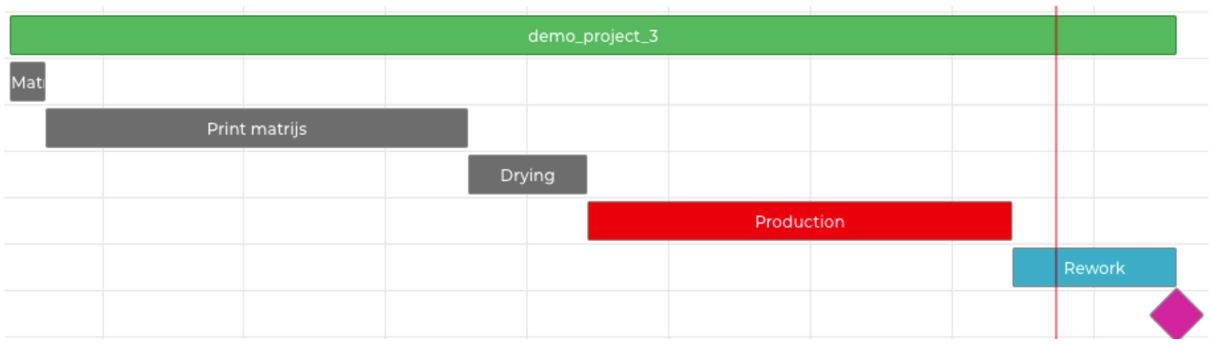


Figure 4.16: Screenshot of the visual cues functionality

4.5.8 Overlap detection

The client asked for the functionality to be able to demand an overview of which processes are currently overlapping given a specific threshold. The grid can show an overview of all vertically overlapping tasks (which are retrieved form the back-end) in a simple message. This allows the user to quickly see if any mistakes have been made regarding resource allocation of processes for example. An example of this functionality can be seen in figure 4.15.

4.5.9 Conclusions

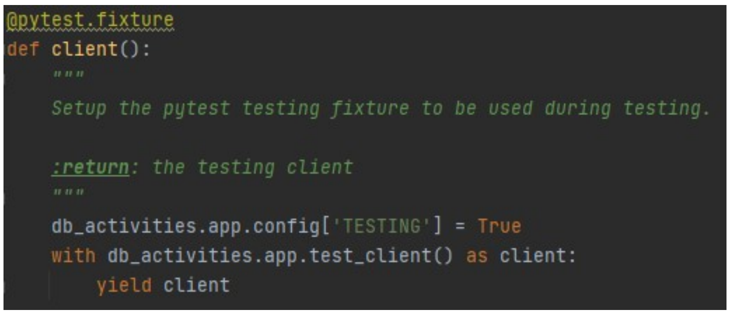
The POC created for this project clearly shows that the architecture implemented works as expected but is also a fully featured and usable product on its own. In fact, the POC will be used by P3D to digitise their planning system. The application has a lot of potential and in conjunction with the microservice based architecture, can be easily modified to allow extra functionalities in the future. Chapter 9 will provide more detail about this but one could already see that an auto-scheduling feature is easy to implement given the current user interfaces and architecture. As a conclusion, the POC provides a way for the client to have a general overview of its company's workflow while simultaneously showing that the microservice architecture backing it is working as expected.

5 | Testing

This chapter discusses how testing has been handled throughout this project and what code coverage has been achieved. As stated in the research report, end-to-end testing, or front-end testing, has not been handled directly. Instead of front-end testing this project used many demos to the client to test and verify the system's behaviour. Therefore this chapter does not contain a specific section for end-to-end testing. The most important part of this project is the correct functioning of the microservices and database which is why the main focus has been back-end testing.

5.1 Back-end Testing

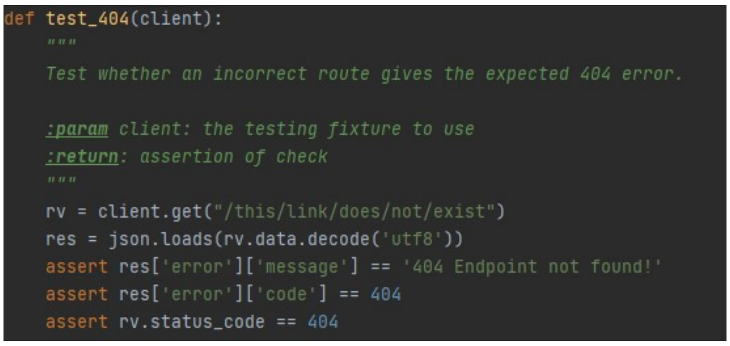
For back-end testing within this project, the Pytest library has been used to test the behaviour of microservices. This library allows the usage of GET and POST which enabled testing the routes within each microservice. Since the complexity and amount of routes within each microservice varies strongly, the amount of tests per microservice also differs. Before a test can be executed, a so-called Pytest 'fixture' is created that gives the ability to use the methods and routes of the to-be-tested microservice. This fixture is setup and created as shown in figure 5.1.



```python
@pytest.fixture
def client():
    """
    Setup the pytest testing fixture to be used during testing.

    :return: the testing client
    """
    db_activities.app.config['TESTING'] = True
    with db_activities.app.test_client() as client:
        yield client
```

Figure 5.1: Setting Up The Testing Fixture

The created 'client' is passed as a parameter to all testing methods and allows the tests to make GET and POST requests. An example test for testing the 404 route of a microservices is given in figure 5.2.



```python
def test_404(client):
    """
    Test whether an incorrect route gives the expected 404 error.

    :param client: the testing fixture to use
    :return: assertion of check
    """
    rv = client.get("/this/link/does/not/exist")
    res = json.loads(rv.data.decode('utf8'))
    assert res['error']['message'] == '404 Endpoint not found!'
    assert res['error']['code'] == 404
    assert rv.status_code == 404
```

Figure 5.2: Example Test For The 404 Route

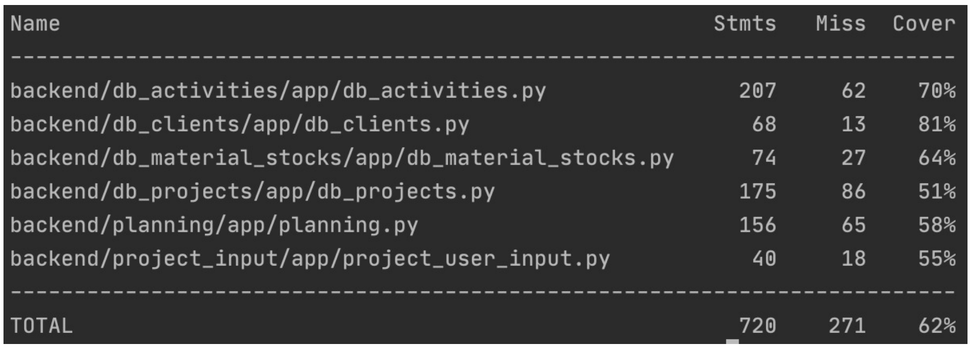
In the test in figure 5.2 a call is made to a route that does not exist upon which the corresponding microservice response with an HTTP 404 code and error data in a JSON format.

5.1.1 Testing Approach

The main testing approach throughout this project has been to write both tests that call routes using the correct data payload and using requests that do not adhere to the routes rules and data schemes. This results in multiple tests per route to ensure the data validation works correctly and the route only functions when it has received all the correct data.

5.2 Code Coverage

The code coverage of this project is at the moment this report was written 62%, but the aim is to increase this to 75% when the code deadline is due. Figure 5.3 shows the current code coverage for each microservice and the total code coverage.



```
Name                                                      Stmts   Miss  Cover
-----------------------------------------------------------------------------
backend/db_activities/app/db_activities.py                  207     62    70%
backend/db_clients/app/db_clients.py                         68     13    81%
backend/db_material_stocks/app/db_material_stocks.py         74     27    64%
backend/db_projects/app/db_projects.py                      175     86    51%
backend/planning/app/planning.py                            156     65    58%
backend/project_input/app/project_user_input.py              40     18    55%
-----------------------------------------------------------------------------
TOTAL                                                       720    271    62%
```

Figure 5.3: Code Coverage Of The Project

One major reason why some microservices have a rather low code coverage percentage is that many statements that were not tested are error and exception handling statements that could only be reached by breaking a connection to the database or corrupting data in the .env files. Time has been spent to figure out how .env files could be changed for the testing environment but since no solution worked in the project environment, it was decided to ignore these missing statements. Rather, effort and time was spent on testing the routes and methods that were used by the user interface. Another thing to note from figure 5.3 is that both the planning and project__user__input microservices have rather low code coverage, this is caused by the fact that those microservices call routes from other microservices and combine those data-sets. Since the routes in those microservices are already tested in their own test packages it was decided to spend less time on testing the aforementioned microservices leading to a lower code coverage rating.

6 Product Evaluation

6.1 Overview

In this chapter, the final product will be evaluated. In order to successfully do this, this chapter will look at different benchmarks to measure the success of the project and its resulting product. In section 6.2, an evaluation will be done on the design goals (stated in section 2.2). It will explain exactly how the design goals have been integrated into the system. In section 6.3, the requirements (stated in section 2.3) of the project will be evaluated. It will go over every requirement and explain for each one if it has or has not been implemented.

6.2 Evaluation Design Goals

The three design goals of the project are extensibility, maintainability and flexibility, each with their own sub-goals. They will be evaluated in this section. For each of them, a brief description will be given. After that, it will be checked how many of its sub-goals have been satisfied and how its integration with the product is being reflected by the final product. From this will be concluded if the design goal has been achieved.

6.2.1 Extensibility

Extensibility is the most important design goal of the system. Future developers should be able to freely add new data sources and features to the system.

Sub-goals

This design goal has two sub-goals. The first one is 'separation of data and functionality'. Due to the use of the microservice architecture, this sub-goal has been satisfied. The second sub-goal is 'extensive documentation'. As all of the code has been documented, this sub-goal has also been satisfied.

Integration

The integration with the product has been a success; all of the code has been documented and developers are able to add new data sources and functionality to the system without having to change a lot of the existing code.

Because the sub-goals have been satisfied and integration has been successful, this design goal is considered 'achieved'.

6.2.2 Maintainability

Maintainability is the second most important design goal of the project. Future developers should be able to easily understand and work with the new code.

Sub-goals

This design goal has three sub-goals. The first one is 'code quality'. The code conforms to general code quality standards, is understandable and easy to read. Therefore the first sub-goal has been satisfied. The second sub-goal is 'testability'. The code of the system is modular and therefore highly testable, which is shown by the 75% coverage. Therefore the second sub-goal has been satisfied. The third and final sub-goal is 'modular independence'. Because of the use of the microservice architecture, the code is modular independent and changes in one module will change little to nothing in other modules. Therefore, the third sub-goal has been satisfied.

Integration
The integration with the product has been a success; future developers have many tools to their disposal to quickly understand and work with the code. On top of that, they have a lot of liberty in adjusting existing code when necessary. This will allow them to work more easily with the code, which supports maintainability.

Because the sub-goals have been satisfied and integration has been successful, this design goal is considered 'achieved'.

6.2.3 Flexibility

Flexibility is the third-most important design goal of the project. In order to create a well-functioning system that is able to adapt and be expanded in the future, the code base should be flexible.

Sub-goals
This design goal has two sub-goals. The first one, 'modular independence', was also mentioned in section 6.2.2 and, as described there, has been satisfied. The second sub-goal is 'extensive code communication'. The way the code base is set up, there is a lot of communication between the different microservices. This sub-goal has therefore been satisfied.

Integration
The integration with the product has been a success; the code base does not make predictions about possible future features that might or might not be added to the project, which makes it very flexible.

Because the sub-goals have been satisfied and integration has been successful, this design goal is considered 'achieved'.

6.3 Evaluation requirements

In this section, the requirements listed in section 2.3 will be evaluated. In table [t1] the requirements are listed, along with their priority, their status, success, and an explanation. Their priority refers to their category according to the MoSCoW model. Their status can be either *implemented* or *not implemented*. If the status is *implemented*, the explanation will describe how this has been done. If the status is *not implemented*, the explanation will explain why this is the case. Success refers to their success criteria mentioned in the research section 2. If its entry is *true*, it means the success criteria has been met, otherwise it will be *false*.

No	Category	Name	Status	Explanation	Success
1	Must Have	Reading client data from the ERP system	not implemented	Team did not get access to the ERP database, which made this requirement not feasible within the timeframe of the project. Can still be implemented in the future.	NO
2	Must Have	Project planning can be digitally entered by users	implemented	Planning data can be entered through the UI.	YES
3	Must Have	Project planning can be digitally viewed by users	implemented	Planning can be viewed through the UI	YES
4	Must Have	Multiple projects can be digitally viewed at the same time	implemented	Planning can be viewed through the UI	YES
5	Must Have	The system should reflect the project status	implemented	Project status can be viewed through the UI.	YES
6	Must Have	System should show the starting of drying material	implemented	Will be shown on a timeline in the planning as a task for a specific project.	YES
7	Must Have	System should show the ending of drying material	implemented	Is shown on a timeline in the planning as a task for a specific project.	YES
8	Must Have	User can input the start of material drying	implemented	Planning data can be entered through the UI.	YES

9	Must Have	System should show the starting of production	implemented	Is shown on a timeline in the planning as a task for a specific project.	YES
10	Must Have	System should show the ending of production	implemented	Will be shown on a timeline in the planning as a task for a specific project.	YES
11	Must Have	User can input the start of production	implemented	Planning data can be entered through the UI.	YES
12	Must Have	User can input the end of production	implemented	Planning data can be entered through the UI.	YES
13	Must Have	Shipping status should be reflected	implemented	Will be shown on a timeline in the planning as a task for a specific project.	YES
14	Must Have	Invoice status should be reflected	implemented	Will be shown on a timeline in the planning as a task for a specific project.	YES
15	Must Have	The user can access the system from anywhere	implemented	System has been deployed on Google Cloud.	YES
16	Must Have	User can log in to the system	implemented	Login system has been implemented.	YES
17	Must Have	User can log out of the system	implemented	Login system has been implemented.	YES
18	Must Have	System should reflect correct material stock throughout the process	not implemented	Team did not get access to the ERP database, which made this requirement not feasible within the timeframe of the project. Can still be implemented in the future.	NO
19	Must Have	System should show the starting of rework	implemented	Will be shown on a timeline in the planning as a task for a specific project.	YES

20	Must Have	System planning should show the ending of rework	implemented	Will be shown on a timeline in the planning as a task for a specific project.	YES
21	Should Have	System should reflect the printing status	implemented	Will be shown on a timeline in the planning as a task for a specific project.	YES
22	Should Have	System can suggest the overall time of production	implemented	System provides standard values for durations during entering planning data.	YES
23	Should Have	System should reflect the print cleaning status	not implemented	Discussed with the client. Prioritized implementing other features over this one. Can still be implemented in the future.	NO
24	Should Have	System should reflect the status of making the sprue gate	not implemented	Prioritized implementing other features over this one. Can still be implemented in the future.	NO
25	Should Have	Ordering status of steel moulds should be mentioned	implemented	Will be shown on a timeline in the planning as a task for a specific project.	YES
26	Should Have	Quality control of moulds should be mentioned	not implemented	Prioritized implementing other features over this one. Can still be implemented in the future.	NO
27	Should Have	Historical project data should be saved for later use	implemented	Unless explicitly deleted, the data will remain in the database.	YES
28	Could Have	System should show invoice documents	not implemented	No data present that was required to implement this feature. Also not feasible within the timespan of the project. Can still be implemented in the future.	NO

29	Could Have	Reading product information from 3D file client	not implemented	No data present that was required to implement this feature. Also not feasible within the timeframe of the project. Can still be implemented in the future.	NO
30	Could Have	Project should be planned automatically by the system	implemented	System provides standard values for durations during entering planning data.	YES
31	Could Have	The system sends alerts when the stock of material is getting low	not implemented	Team did not get access to the ERP database, which made this requirement not feasible within the timeframe of the project. Can still be implemented in the future.	NO
32	Could Have	Historical project plannings can be viewed digitally inside of the system	implemented	Planning shows all projects with a deadline later than two weeks ago.	YES
33	Won't Have	User can input finishing of material drying	implemented	Did not seem like a good idea to add the feature during the Research Phase, which is why it is under Won't Have. Discussed with the client Eventually decided to add.	YES
34	Won't Have	The clients of P3D can login to the system and place new orders	not implemented	Not feasible within the timeframe of the project.	NO
35	Won't Have	The system automatically schedules and creates backups of its entire state	not implemented	Not feasible within the timeframe of the project.	NO

6.4 Summary

The final product integrates all of the design goals into the system. Also, most of the requirements have been implemented and all of the requirements that have not been implemented have a valid reason. Therefore, the product is evaluated to be a success.

7 | Process Evaluation

During the project, the team encountered obstacles on its way, but also things that went better than expected. This chapter contains the evaluation of the process.

Scrum
During this project, the team used the scrum methodology. This choice is described in chapter 2.4.2. Each sprint had a duration of 2 weeks. To keep track of the sprints, the team used the Boards functionality on GitLab. The Boards functionality keeps track of all open issues where each issue is assigned to an 'Epic' issue (main functionality of the system) and a milestone (sprint). Every morning on a workday there was a stand-up with the team and two times per week there was a meeting with the client.

Coronavirus
Due to the coronavirus, every meeting was online. These meetings went fine as Jeroen Gross was able to answer our questions thoroughly. However, it would have been easier and more efficient to learn about P3D's workflow if more on site visits could be have been organised. Unfortunately, this was impractical due to the imposed COVID-19 regulations.

Client
The communication with the client went very well during this project. Before the project started, the team and the client had a few meetings already. When the project started, the teams visited, in groups of two, the company in The Hague to have an overview of the workflow of the company. The meetings with the client were always very informative and the team enjoyed these meetings. The client has always been very helpful for the team.

Communication hosting company
The ERP database is hosted by a 3rd party company. The communication with this company did not go as expected. It took a couple of weeks until the team got a response. After this response, the company told the team that they could not help the team because they did not want to be responsible for any security flaws in the system. The new database creation and its input systems had to be built in the last weeks.

Parallelisation
At the beginning of the project, the team was divided into 2 groups. The first group started with continuous integration and continuous deployment (CI/CD). While the other group worked on the first microservices. In this way, the team could work in parallel on the different components of the project. This turned out to be a good approach and meant a well-working development framework was setup while not delaying the implementation of the desired features of the client.

8 | Conclusions

This chapter discusses the findings and conclusions found during the duration of this project. The main focus throughout this project has been to develop a data management system for the company P3D. Before the actual development could start, time was spent on researching different architectures for the web application which lead to the current design. P3D asked for the system to be easily extendable and maintainable which is why a microservice architecture hosted on a Kubernetes (Burns et al., 2018) environment was chosen. Although this required more work to create, it achieves an easy to expand and maintainable framework. After the entire framework was set up, the group concluded that creating the specified architecture allowed for easy expansion of the system and allowed microservices to be automatically replaced when they went down. The usage of microservices has allowed the project to be clearly divided in separate Flask projects, each with its own sole purpose. This way it is clear what each microservices does, and more importantly, when one microservices goes down other microservices are unaffected. The user interface of the product has been made using React which made the development of the front-end faster because of re-usable components and open-source components. One part of the user interface, the planning, was made using a third party library called DHTMLX which provided a pre-made solution to the planning interface and allowed easy modification.

Aside from the technical conclusions, there are also conclusions to be drawn from the communication with the client. The first two weeks were spent on getting to know the client's company, P3D, and what they do. These lead to the creation of a list of requirements that were used to setup milestones for the entire duration of the project. Along the way some obstacles or challenges showed up that caused the team to re-evaluate the goals and requirements with the client, what can be concluded from this process is that a clear explanation of certain problems during a project towards the product owner or client causes the client to be pro-actively involved in finding a solution to this problem. One such example was that the client stated that a connection to their ERP system was a requirement, however the contact with the third party company that manages the ERP system of P3D was quite slow, stagnating the entire project process. Therefore, it was decided, after having contacted the client about this problem, to mock the ERP database which saved time and allowed the product development to continue. Furthermore, it can be concluded that having multiple meetings per week with the client to give updates on the progress of the project allows the client to give more feedback which speeds up the development time.

9 | Recommendations

As the total duration of this project was around 3 months, it is of course inevitable that not all features the client desires could be implemented in the final product. Moreover, given the nature of this project, the system will continue to evolve in the future to match P3D's ever-improving workflow. This section will therefore outline the recommended features that could be implemented in the future, starting with the ones that will be the most beneficial to the client. Also, this section will talk briefly about what is required for the project to be maintained and extended by other people in the future.

9.1 Possible system extensions

This section outlines the possible features that could be implemented in the future to the benefit of the client. Note that this section does not list all features that are recommended to be added but merely the ones that are believed to be the most significant in the improvement of P3D's workflow. The should-haves (2.3.2) and could-haves (2.3.3) sections of the research chapter (2) give more examples of possible additions.

9.1.1 Connection to a cloud-based ERP system

As outlined in chapter 3, complications arose when coupling this project with P3D's existing ERP system. This was eventually solved by creating a manual input interface for projects (4.5.2) but this is not ideal. Coupling this project to P3D's ERP system means more data is available to be used (budgets, employee data and inventory data for example) and no data duplication occurs as projects do not have to be entered twice. For these reasons, making this connection is recommended to improve the data management system as a whole. The choice for a cloud-based ERP system (like Microsoft dynamics 365 ®) is proffered as this limits security issues when integrating with the already cloud-based data management system.

9.1.2 Automatic planning of projects

When a new project is created in the project input interface (4.5.2), the system currently uses static default values to calculate start and end times for all processes. This is of course not ideal as it does not take in to account vacations, resource availability and other constraints of P3D its workflow. In the future, it would be preferred to automatically schedule processes based on data retrieved from an ERP system or other sources. The library DHTMLX Gantt used in the planning component (4.5.4) already has support for an auto-scheduling functionality, it is therefore only required to collect the necessary data and constraints to implement this feature into the system. This feature is believed to be very beneficial as it would significantly reduce the time managers of P3D spend on planning different projects and, if implemented well-enough, would result in more efficient workflow planning.

9.1.3 Addition of incremental progress and more types of processes

In the current implementation of the system, the incremental process is not saved nor displayed. Processes can either be finished or not but no visual indicator is given on how far a process is in its completion. In the future, it would give P3D its managers a better overview if the progress of processes can be viewed as a percentage of their estimated finishing time. This is directly linked to another functionality, namely adding more precise types of processes. Currently the system supports 6 different types of processes. These are the main processes in P3D's workflow but could be further subdivided into more precise tasks. Alongside the incremental progress functionality, this could give a very detailed overview of P3D's workflow state. Both of these features are already supported in the planning interface and therefore only require the necessary data for these features to be acquired and added to the overall DMS.

9.1.4 Data analysis of previous projects

The current implementation of the DMS saves historical data about projects that have been finished in the past. This offers an opportunity as this data can give valuable insides into P3D's planning of projects and workflow behaviour. P3D could analyse this data in order to optimise and better predict the behaviour of projects in the future. Bottlenecks and mistakes in its current workflow could also be found using this analysis. As the data required for this functionality is already available, adding this functionality simply consists of starting the analysis.

9.2 Maintainability and extensibility in the future

The features presented in the previous section are believed to be most beneficial to P3D and would improve the planning of P3D's workflow significantly. During the entire course of this project, special attention has been given on making sure these features could be added with little effort in the future. It is therefore believed that adding these aspects will not create significant change to the overall design but will only require the feature to be implemented and connected to the already existing infrastructure.
It is however required that people with the right set of skills are hired to maintain and extend this project. P3D is advised to hire developers that have experience in the following fields and technologies:

- Continuous integration and development using Docker (Turnbull, 2014) and Kubernetes (Burns et al., 2018)
- Front-end development using React (Aggarwal, 2018) and TypeScript (Freeman, 2019)
- Back-end development using Python (Rossum, 1995) and Flask (Aslam et al., 2015)

10 | Discussion On Ethical Implications

This chapter will discuss the ethical implications of this project and what these implications mean for the way the product should be maintained and handled.

10.1 Privacy of Data

In the system, client data is being processed to enable the end users to see what projects belong to what clients. The data in these projects is stored in a Google Cloud PostgreSQL database. This database is located in Europe and therefore it follows the European GDPR rules regarding privacy, this takes away the major concern of privacy of data. Other data that is being entered into the system via the user interface is not as sensitive and therefore no ethical concerns apply to that data. P3D will notify their clients about the location of their data. In case they do not agree with this, P3D will offer the option to not use this system for their project. If in the future, more clients report having an issue with the hosting location of their data, it is possible to modify the system to make use of alternative database hosting solutions with minimum effort.

10.2 Security

Although the data used within the project is stored in a database that follows strict regulations from the EU, the way the data is being processed within the system can also be seen as an ethical discussion point. However, since all data in the project is being sent over secure connections, and authentication is needed to access the system the security has been taken care of in this project. It is recommended that the system will be maintained by other developers upon delivery to make sure that the latest security fixes have been implemented and no possible security issues arise.

A | Original Project Definition

P3D (part of Promolding Holding) has developed the PRIM® (PRinted Injection Mould) technology: injection moulding of plastic products in 3D printed plastic moulds. P3D has all the expertise and machines for 3D printing and injection moulding available in-house. This way we are able to get from product design to production in the shortest way possible. With this service we provide a substantial part of product development and production start-up for our clients in a way that was not possible before.
In short: automate all steps in our workflow by building (parts of) a management system.

P3D is a young company and a sister company of injection moulder Promolding. P3D produces small series injection moulded products for a wide variety of clients. We have found developed an affordable way to let injection moulding have very short lead-times for making small series injection moulded parts by applying 3D printed moulds.
At the moment we are very much a company mainly focused on making physical products. And although this will remain the core of our activities, we need to improve our processes in all areas to make them leaner, better controlled, quicker, of higher quality, etc.
In this processes improvements we see digitisation as the way forward.
This goes for all steps in our processes. Sales, project management, planning, mould design and making, materials management and handling, supplier management, production, finance, etc, etc. Are you the ones to help us out with our next steps?

The project will consist of:

- Getting to know the processes of P3D; why, what and how?
- What can be automated and done smarter?
- Get better grip on project management and planning: this will be the focal point of the project.
- What is needed and how can it be realised?
- Whatever we build: how will it be sustainable for future implementations of automation?

First you will get to know the processes of what we do from A to Z. How do we work, what do we do, who are involved, what is injection moulding, etc, etc. With this information a rough picture can be painted of what, where and how digitisation/automation can be implemented. Data will be obtained from current software being CAD-software (Autodesk Fusion 360), ERP (Bemet Plan de Campagne), Excel and webforms.
The biggest part of the assignment will be: how can project management and planning be automated, optimised and better controlled? From the first contact with a client, his project should be (visibly) planned and scheduled with a simple push of a button (so to speak). At the start of a client project, most steps are still pending and preliminary. During the project steps become done, fixed, adjusted, etc. This should be clear to all stakeholders, including our co-workers, suppliers and clients. Visual, flexible and accessible with people focusing on decision making rather than data entry.
So, your student project will be to develop a planning module for P3D that has the flexibility to grow and be implemented in the bigger picture of our company.

Bachelor Thesis: A Data Management System for P3D

Client organization: P3D
26-01-2021

P3D is an injection moulding company that uses a technology called PRIM®(Printed Injection Mould) to create products for its customers. The data of the workflow of P3D was not digitised and was therefore prone to human errors. The company needed a data management system to keep track of the workflow. The system needed to connect to multiple databases and was therefore developed with a microservice architecture. The big challange during the research phase was understanding how microservices work, how docker containers are created and how to deploy those containers on a kubernetes server. It is hosted on the Google Kubernetes Engine. The final product became a web application that was tested with back-end tests and front-end client confirmation. The product will be used and further developed by P3D.

Members:

- Zeger Mouw (Project Data, Deployment) zegermouw@gmail.com
- Gijs Paardekooper (Microservices, Databases and testing) gijs.paardekooper@gmail.com
- Abri Bharos (Back-end Development, specifically planning tool) a.r.j.bharos@gmail.com
- Tim Pelser (Front-end Development) timpelser@outlook.com
- Erik Sennema (ScrumMaster, Deployment) eriksennema@outlook.com

All team members contributed to preparing the report and the final project presentation

Client: Jeroen Gross (Business Unit Manager)
TU Coach: Asterios Katsifodimos (Faculty of Engineering, Mathematics and Computer Science Tu Delft, Web Information Systems)

The final report for this project can be found at: http://repository.tudelft.nl

Bibliography

Aggarwal, Sanchit (2018). “Modern Web-Development Using ReactJS | Document Object Model | Model–View–Controller”. In: *International Journal of Recent Research Aspects* 5.1, pp. 133–137. URL: https://www.scribd.com/document/379709841/Modern-Web-Development-Using-ReactJS#download.

Ashley, David (2020). “Comparing CGI, SSI, Flask, and Django”. In: *Foundation Dynamic Web Pages with Python*. Springer, pp. 201–208.

Aslam, Fankar Armash, Hawa Nabeel Mohammed, Jummal Musab Mohd. Munir, and Murade Aaraf Gulamgaus (2015). “Efficient Way of Web Development Using Python And Flask”. In: *International Journal of Advanced Research in ComputerScience* 6.2, pp. 54–57. URL: https://core.ac.uk/download/pdf/55305148.pdf.

Brito, G and M T Valente (2020). “REST vs GraphQL: A Controlled Experiment”. In: *2020 IEEE International Conference on Software Architecture (ICSA)*, pp. 81–91.

Burns, Brendan, Joe Beda, and Kelsey Hightower (2018). *Kubernetes*. Dpunkt.

Dybå, Tore and Torgeir Dingsøyr (2008). “Empirical studies of agile software development: A systematic review”. In: *Information and Software Technology* 50.9-10, pp. 833–859. ISSN: 09505849.

Ebert, C, G Gallardo, J Hernantes, and N Serrano (2016). “DevOps”. In: *IEEE Software* 33.3, pp. 94–100. ISSN: 1937-4194.

Freeman, Adam (2019). “Understanding TypeScript”. In: *Essential TypeScript: From Beginner to Pro*. Berkeley, CA: Apress, pp. 35–40. ISBN: 978-1-4842-4979-6. URL: https://doi.org/10.1007/978-1-4842-4979-6_2.

Fuchs, C, S Spolaor, M S Nobile, and U Kaymak (2020). “pyFUME: a Python Package for Fuzzy Model Estimation”. In: *2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pp. 1–8.

Hardt, Dick et al. (2012). *The OAuth 2.0 authorization framework*. Tech. rep. RFC 6749, October.

P3D, BEP (2021). *Bep-framework*. URL: https://pypi.org/project/bep-framework/.

Reese, Will (2008). “Nginx: the high-performance web server and reverse proxy”. In: *Linux Journal* 2008.173, p. 2.

Rossum, Guido (1995). *Python Reference Manual*. Tech. rep. NLD.

Turnbull, J (2014). *The Docker Book*. James Turnbull. ISBN: 9780988820234. URL: https://books.google.nl/books?id=CtMEBwAAQBAJ.

Villamizar, Mario, Oscar Garces, Harold Castro, Mauricio Verano, Lorena Salamanca, Rubby Casallas, and Santiago Gil (2015). “Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud”. In: *2015 10th Colombian Computing Conference, 10CCC 2015* October, pp. 583–590.

Vohra, Deepak (2016). *Kubernetes Microservices with Docker*. 1st. White Rock: Apress, pp. 39–42.

Waters, Kelly (2009). “Prioritization using MoSCoW”. In: *Agile Planning* 12, p. 31.