PDEng THESIS REPORT

Model as a Service: Towards a Discovery Platform for Internet of Food

Hossain Muhammad Muctadir
10/2020
Department of Mathematics & Computer Science

PDEng SOFTWARE TECHNOLOGY

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Model as a Service: Towards a Discovery Platform for Internet of Food

Hossain Muhammad Muctadir

October 2020

Eindhoven University of Technology

Stan Ackermans Institute – Software Technology

PDEng Report: 2020/064

Confidentiality Status: Public

Partners	Sustainable Food Initiative and Internet of Food Consortium including: • Wageningen University & Research • Unilever Research & Development • NIZO Food Research BV • ISPT Foundation • Symrise AG	Eindhoven University of Technology
Steering Group (ordered alphabetically by first name)	Prof.dr. Jakob de Vlieg Dr. Michiel Gribnau Dr. Rogier Brussee Dr. Serguei Roubtsov Dr. Yanja Dajsuren, PDEng	
Date	October 2020	

Date	October, 2020
Contact address	Eindhoven University of Technology Department of Mathematics and Computer Science Software Technology MF 5.080 A P.O. Box 513 NL-5600 MB Eindhoven, The Netherlands +31 402744334
Published by	Eindhoven University of Technology
PDEng Report	2020/064

Abstract

The Internet of Food (INoF) consortium, which is part of Sustainable Food Initiative (SFI), aims to address the future food safety challenges using engineering solutions to make the production process more efficient and sustainable. Inter-organization collaboration can stimulate fast innovation and sustainable research processes by significantly reducing data loss as well as miscommunication. Such collaboration requires an appropriate digital infrastructure that can maintain interoperability among diverse data formats from different sources. This infrastructure should also be able to facilitate sharing of data and services without companies having to share IP (Intellectual Property) or replicate corresponding execution environments. As part of the INoF, this project aims to develop a prototype for such infrastructure and set up a baseline for building an effective model discovery platform. In this context, models are computational units that can provide insights into food products. Having access to results from more models, companies can make better decisions and speed up product development.

During this project, a microservice based architecture was designed and a prototype was developed that exploited the idea of Model as a Service (MaaS). It has the functionality to offer models in the form of web services allowing organizations other than the owner of the models to use them. For achieving interoperability among different data sources in the context of this project, functionalities, such as dynamic model parameter mapping and on-demand unit conversion, were implemented into this prototype. After execution, results from several models belonging to different organizations can also be viewed through this platform. One of the major goals of this project was to demonstrate the benefits and possibilities of sharing model results to attract further collaboration. Therefore, several INoF partners were closely involved in this project. The MaaS prototype was also demonstrated to all the INoF partners and earned quite a few appreciations.

Keywords	Internet of Food, Sustainable Food Initiative, Model Sharing, Data Fusion, Model as a Service, Integrative Data Science, Model Driven Decision Making
Preferred reference	Model as a Service: Towards A Discovery Platform for Internet of Food. Eindhoven University of Technology, PDEng Report 2020/064, October 2020.
Partnership	This project was supported by Eindhoven University of Technology, Internet of Food Consortium, and TKI Dit project ontvangt financiële steun van de Topsector Agri & Food. Binnen de Topsector werken bedrijfsleven, kennisinstellingen en de overheid samen aan innovaties voor veilig en gezond voedsel voor 9 miljard mensen in een veerkrachtige wereld.
Disclaimer Endorsement	Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the Eindhoven University of Technology and Company name. The views and opinions of authors expressed herein do not necessarily state or reflect those of the Eindhoven University of Technology, Internet of Food Consortium, and TKI, and shall not be used for advertising or product endorsement purposes.

Foreword

The Netherlands is the second-largest exporter of food in the world, and every year the sector generates ca. €50 billion of added value and accounts for more than €80 billion of exports. In comparison, the global agriculture revenue is approximately $4.8 trillion and only gets larger. If current trends continue, caloric demand will increase by 70 percent in 2050 and crop demand for human consumption and animal feed will increase by at least 100 percent. Finally, there is no way around it: food production must reduce its ecological footprint and become sustainable in the very near future! Meeting this demand won't be easy! According to the World Economic Forum a systemic transformation is needed to deal with the fast-growing demand for high quality food products, and the challenges imposed to meet sustainability goals.

Science and technology are increasingly playing a role to implement the required changes to produce enough healthy, tasty and affordable food. The food sector will have to work with more precision and control and enough data on every part of the process is needed, As a result, the agri-food sector has become extremely data intensive and has to benefit from new disruptive technology developments, e.g. in the field of life sciences, medicine, food sciences, information technology, artificial intelligence (A.I.), (multi-)sensing and computer vision to keep the sector future proof. This has resulted in a new science and research area currently under development: the interdisciplinary field of AgriFoodTech, i.e. crossovers between the agri & food domain and high-tech & digitization. In particular, there is a lot of potential value in data integration between the various steps and activities within the food value chain. Total integration of data and models throughout the entire chain, from seeds and fertilizers, towards feedstock, livestock, agriculture, food processing and retail, is clearly a moon-shot project and aimed to solve several long-term "Data Inside and Digitization" challenges in the food sector. Nevertheless, chain integration and consumer-driven chain reversal have become achievable by fast developments in A.I., digitization, (multi)sensing and computer vision and might even rewrite entire innovation and business models in the food sector.

This PDEng research project by Hossain Muctadir and this report, are part of the sustainable food initiative (SFI). In this initiative, companies and knowledge institutes teamed up to produce healthy and safe food products at the lowest possible environmental footprint. An important goal of the SFI program is to develop a shared roadmap for digitally enhanced discovery. This means speeding up food related R&D and the development of sustainable, flexible and precise food production processes by using modern data fusing, data analytics, modeling, computer simulation and other digital techniques. The goal of the digital roadmap to translate complex data into useful knowledge and to explore new ways of data-driven experimentation, e.g. in silico explorations of food combinations and food processing or multi-sensing-based decision making In this PDEng project, a first, but important step has been taken towards the development of a discovery informatics platform to connect cross-type data and models between ecosystem partners, each having their own models, their own data, their own software and ICT solutions and their own IP. Note that partners may, but need not be in different organizations. Such an informatics infrastructure speeds up product innovation by reducing the barrier for collaboration and increasing reuse of data and knowledge. The strategic starting point is to provide easy access to different models by gluing various big data layers together through high quality ontologies and meta-data ETL (Extract-Transform-Load).protocols A.I. and other data analytics algorithms usually needs access to heterogenous data obtained by e.g. (multi)sensing and data stored in multiple historical data bases. Reuse of existing analysis methods, and automated support for time consuming data transformations allow to focus on new analyses and data fusion that requires more extensive domain and data-science knowledge such as dealing with noisy and incomplete data. The developed computer science design is a layered microservice architecture based on FAIR (Findable, Accessible, Interoperable and Reusable) principles. The architectural design choices allow both, future use of Artificial Intelligence (A.I.) techniques (e.g. machine or reinforcement learning) or statistical models, and the use of visualizations or dashboards. This approach will be crucial to translate the complex data sets from the food value chain into new scientific insights and to improve sustainability of food production in the near future.

In a first prototype Hossain Muctadir showed that different types of data, ranging from ingredient data via product development and processing data to consumer data can be combined, adapted, and analyzed to answer relevant food-product development questions and to encourage inter-organization collaborations between companies and/or academic institutes. Although his work focused on demonstrating the possibility to exchange existing proprietary foods models between the involved SFI ecosystem partners based on a “share models without giving these away” basis, it is clear that the underlying computer and data science concepts developed in this project are generic enough to support other data-driven challenges in the food domain. Potential applications are e.g. in silico food product development, lights-out factories with sensor-based automated monitoring, and data-driven quality control using text mining and machine learning on vocabulary of sensory panels.

In short, in this PDEng project Hossain demonstrated successfully, that microservice architectures and using FAIR (Findable, Accessible, Interoperable and Reusable) principles to translate cross-type data allows to bring data & predictive models from different sources together in a single digital architecture. This greatly extends the potential to use computer models as actionable knowledge in the food industry.

Hossain, you have done a great job, especially considering the difficult circumstances you had to work in due to the corona measures. All through the project you have proved to be an exceedingly well organized, very hardworking, independent and problem-solving student. Supervising you is a great pleasure! Thank you very much for all your excellent work and achievements making this first phase of the SFI “Data Inside” moonshot project a success. I wish you the very best with the rest of your career!

Prof.dr. Jakob de Vlieg

Chair of the Applied Data Science (ADS) research group

Lead AgriFoodTech@TU/e and the JADS AgriFood&Data program line

September 2020

With ever faster and changing consumer demands, food industry needs to speed up its innovation and production process to become more efficient and sustainable. A clear example of this is the recent COVID-19 crises, which has led to big changes in consumer behavior and ingredient availability. On the one hand, consumers moving from out of home eating to in home meal making, which requires a change in the products offered. On the other hand, raw materials not being available due to transport issues, which require rapid replacement of these ingredients in the products on shelf .

A key requisite for tackling these changing consumer demands is a better usage and sharing of the available data and model results via a linked data infrastructure. Different types of data, ranging from ingredient data via product development and processing data to consumer understanding, also in combination with scientific data are stored in different places and databases with different structures at different companies, universities and other institutes and cannot be shared easily. This also holds for proprietary products and process property models, which cannot talk to each other.

As said above, the absence of common standards and tools to (conditionally) use each other's data and model results hampers the rate of innovation in the food industry.

It would be great, if one could create an infrastructure or platform which would allow the (conditional) sharing of data and results of models from different organizations and model owners as this will create a much more sustainable research process with a significant reduction in data loss and miscommunication.

In this report, Hossain Muctadir describes how he has developed an infrastructure for sharing the results of models of different companies and organizations via a Models as a Service (MaaS) approach. Together with his team of students he realized a first implementation of the infrastructure, which now can be demonstrated to users. For outsiders, it always remains impressive to see what a small, well led team can do on such a small-time period.

The availability of such a demo is key in several ways. First, it helps the senior managers at different organizations to understand what might be achieved, what the potential of such an approach might be, increasing the buy-in of senior stakeholders. Additionally, it helps to understand the academic community what challenges need to be solved to bring this to the next level. Examples of this are the challenges around the alignment of ingredient naming and ingredient properties in the Foods area and the challenge of exchanging the model information in a unique and well-defined way. Both aspects are essential for defining the direction and support of follow-up projects. In this the impact of the work of Hossain should not be underestimated as it is the first step of a long journey.

I would like to thank Hossain for all work he did. He operated in a very organized, transparent and focused way, always delivering what he had promised. When issues arose he put them on the table and openly discussed ways to handle them. This, next to his strong commitment, helped to make the project a big success.

In summary: Hossain, many thanks for all your work and lots of success in the next steps of your career.

Dr. Michiel Gribnau

Unilever R&D, Wageningen

August 2020

Preface

This report presents the activities and analysis performed during the execution of the project "Model as a Service: Toward a Discovery Platform for Internet of Food." This project was realized by Hossain Muhammad Muctadir, who is also the author of this report, as the partial fulfilment of the graduation for the Professional Doctorate in Engineering (PDEng) in Software Technology program. It is a two-year doctoral level technical designer program offered by the Stan Ackermans Institute at Eindhoven University of Technology.

This project was executed in the context of the Internet of Food (INoF) project that is a consortium of leading food production organizations and research institutions. During this project, the author developed a prototype infrastructure where one organization can share their computational models, which can calculate properties of food products, to other organizations as a service.

The intended reader of this report are both technical and non-technical. Chapters 1 and 2 explains the project context and identifies the problems that this project aimed to solve. Moreover, the project requirements and results are listed respectively in Chapters 4 and 7. The project management related activities are explained in Chapters 3 and 8. Above mentioned chapters contain very little technical details and are suitable for non-technical readers who want to understand what this project was about.

The design and architecture of the developed system is explained in Chapter 5, which should be understandable to people with basic knowledge of software engineering. Chapter 6 explains the implementation details, which are quite technical and require advanced knowledge of software development to fully understand.

This report is ordered in such a way that it starts with the project context and problem definition. These are followed by description of the implemented solution. The final chapters of this report are about project results and future possibilities.

Hossain Muhammad Muctadir

September 2020

Acknowledgments

I would like to express my gratitude to everyone who supported and guided me through this project. Their contributions were essential for successfully completing this project.

I want to specially thank my supervisor Jakob de Vlieg who shared his ideas and guided me through this project. Your feedbacks regarding the implementation and the report were essential for successfully completing this PDEng graduation project. Moreover, I absolutely enjoyed our discussions regarding various topics and it was a pleasure hearing your thoughts as well as being able to share mine.

I am very thankful to my supervisor Michiel Gribnau. As an expert in food production domain, you have tremendously helped me in understanding the domain concepts and guided me through this unfamiliar terrain. I really appreciate your feedbacks and support not just in the technical but also with the non-technical aspects of the project. Your collaboration was absolutely essential for successfully completing this project.

I would like to express my heartfelt gratitude to my academic supervisor and the program director of PDEng Software Technology Yanja Dajsuren. I learned many things from your guidance and feedbacks not just during this project but also throughout my PDEng years. I believe that all your advice will help me further in my career.

I am truly thankful to my second academic supervisor Serguei Roubtsov. I absolutely appreciate your critical attitude. You always asked the right questions that were essential to steer the project to the right direction. Specially, you feedbacks regarding the graduation report were crucial.

I am grateful to Rogier Brussee for all your contributions through this project. Your feedbacks and curious questions shaped certain aspects of this project as well as the final results. I really enjoyed and learned a lot from the discussions we had.

I want to thank Görkem Simsek-Senel from Wageningen University and Research for her contribution in developing related ontologies that were essential for developing one of the core components. As an ontology expert, you also provided valuable feedbacks and guidelines regarding various aspects of ontology based software development.

I want to acknowledge the contribution of the group of bachelors students from Eindhoven University of Technology who helped me with the development activities during this project. Your efforts were crucial for the success of this project and the quality of your work is quite high. I also want to thank all the partners of the Internet of Food (INoF) consortium who asked relevant questions and provided feedback during INoF meetings as well as demonstrations. I am thankful to all my colleagues from PDEng program for your feedbacks and shared experiences during the last two years. Together we solved many challenges and learned a lot. I would like to specially thank PDEng ST secretary Desiree van Oorschot for her logistical support throughout the PDEng program.

Last but not least, I want to express my deepest and most sincere gratitude to my wife Sanjida Hossain. Throughout my PDEng, you remained by my side and provided unconditional support, encouragement, joy, as well as love. Moreover, I am eternally grateful to my parents and my brother in Bangladesh for their continuous support.

Hossain Muhammad Muctadir

September 2020

Executive Summary

In recent years, the world has seen unprecedented growth of population that, according to United Nations, can increase by three billion by the year 2050 threatening the future food safety. The Sustainable Food Initiative (SFI) aims to address the future food safety challenges by combining fundamental research and fast innovation to produce new generation of sustainable food products. The Internet of Food (INoF) consortium, which is part of the SFI, intends to encourage inter-organization collaboration by creating a digital highway where organizations related to food production can collaborate. Being part of the INoF, this project aims to build a prototype for such collaboration platform.

In the context of the food production domain, models are computational units that can calculate various properties of food products based on the information regarding the conditions, processes, and ingredients necessary to produce them. These models play a key role in the modern food production chain by automating many of the manual processes, which reduces cost and makes the processes more sustainable. Typically, these models are developed in-house and used almost exclusively by their owners. Sharing these models or their results can instigate innovation and make a significant impact on the global food safety goals.

Sharing models or results from them requires a standardized platform that can interact with the models. Developing such a system is a complex task and involves solving several technical as well as non-technical challenges. One of the key technical challenges is dealing with the heterogeneous nature of the models as they are typically developed for in-house usage and therefore, do not follow any generally accepted standard. Another major challenge is about the distribution of the model artifacts (e.g., codes, executables). Sharing these artifacts with other companies is not feasible as the models are intellectual properties of their owners. Moreover, running these models often requires specialized setup that are typically expensive and therefore, defeats the sustainability goals. Consequently, a standardized platform is needed that can address these challenges and facilitate the sharing of model results among companies. Developing such a platform requires advanced usage of software engineering techniques and joint effort of key players from the food production domain. The goal of this project was to develop a prototype of a sharing platform for models and demonstrate its possibilities as well as capabilities to potential partners to encourage further collaboration.

During this project, a prototype sharing platform was developed that had the functionality to connect models from different sources and offer them as a service to external parties using modern web technologies. This concept was named Model as a Service (MaaS). It offered the functionality to define a food product that included the corresponding ingredients, processing steps, and packaging information. A food product definition and one or more models could be paired together to define a simulation that could be executed. While executing a simulation, corresponding models are run with values from the food product definition. To do that, the prototype implemented a mapping mechanism that mapped appropriate properties of food product definition to model input parameters. This feature was implemented using a computational model ontology that was developed in collaboration with

the Wageningen University and Research. Moreover, this ontology of measurement (OM2) was used to convert between units of measurements in case the food product definition and the corresponding model input parameter had a mismatch.

The MaaS prototype was designed and developed based on microservice architecture where different software components were independent microservices that communicated among themselves over REST endpoints. The prototype implements gateways that can connect heterogeneous and remotely located models to the MaaS infrastructure. A permission system was implemented that allowed companies other than the model owner to consume services from remote models. A hybrid storage infrastructure was implemented to store structured data as well as ontology related information. A web application was also developed that provided intuitive web-based user interface to consume the services offered by the MaaS infrastructure. Finally, all the microservices were made deployable using Docker containers, which made it versatile enough to run on almost all general purpose cloud infrastructure and local servers.

The final version of the MaaS prototype was demonstrated to the INoF partners and earned quite a few compliments. During the development of MaaS, several research possibilities and improvement opportunities were discovered. These points and the MaaS prototype itself will serve as a baseline for developing a production grade MaaS system.

Contents

List of Figures

List of Tables

Glossary

API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
INoF	Internet of Food
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
MaaS	Model as a Service
OEM	Original Equipment Manufacturer
OM2.0	Ontology of Units of Measure
PDEng	Professional Doctorate in Engineering
PID	Project Initiation Document
PSG	Project Steering Group
REST	Representational State Transfer
SFI	Sustainable Food Initiative
ST	Software Technology
TU/e	Eindhoven University of Technology
URI	Unified Resource Identifier
WP	Work Package
WUR	Wageningen University and Research

1 Introduction

In recent years, the world has faced several major global challenges due to the unprecedented population growth. Even after making significant advancements in the fields of agriculture and food production, future food safety remains one of the major issues. Moreover, with the population growth, the consumption of various global resources has also increased rendering the traditional way of food production insufficient to meet the predicted demand in the coming years.

This chapter briefly explains the future food safety and how it is connected to this Professional Doctorate in Engineering (PDEng) graduation project. Section 1.1 provides information about Sustainable Food Initiative (SFI) as well as Internet of Food (INoF) consortium and explains how they are connected to each other. Section 1.2 explains the context of this PDEng graduation project in connection to the INoF and briefly explains the major project deliverable. Finally, Section 1.3 presents an outline of this report.

1.1 World Population and Food Safety

The world population has grown at a much higher rate during the last few decades than ever before. Between the year 1960 and 2002, the number of humans has more than doubled [4]. Studies suggest that the world population can reach ten billions by the year 2050 [5], which is three billion more people compared to 2010. This also means there will be three billion more mouths to feed creating a 56% food gap considering the current rate of agricultural growth [5].

Studies also suggest that the world can see a significant economic growth in the coming years. As a result, the relative income gap between the developed and developing countries will be much less pronounced than it is today [6]. As incomes rise, people will start to consume more resources resulting in a 593 million hectare agricultural land gap and 11 gigaton greenhouse gas mitigation gap [5].

To ensure the sustainable food safety for the world population in the near future, preparations need to be taken as early as possible. Researchers have suggested various solutions for ensuring a sustainable food future. The majority of these solutions suggest increasing the production of food without expanding agricultural land as one of the key steps for ensuring food security in the coming decades [4][5][6][7].

1.1.1 Sustainable Food Initiative (SFI)

The Sustainable Food Initiative (SFI) is a mission driven community of leading food production experts and academics. The goal is to combine fundamental research and fast innovation to produce new generation of food products that are safe, healthy, sustainable, as well as climate resilient [8]. SFI also aims to create a community of digital food experts by organizing workshops on modern approaches of food production. The whole initiative is widely supported by companies, knowledge/academic institutions, and the Dutch Government [8].

SFI facilitates different activities to encourage research and innovation in order to be able to meet future demand for sustainable, healthy, and safe food. One of these activities is called Field Labs. These are practical hands-on trials in which companies and knowledge institutions purposefully develop, test, as well as implement smart solutions for the future. They also form an environment in which people learn to apply these solutions [8]. SFI also conducts various projects where the goal is to bring technical and engineering solutions into the food production domain.

1.1.2 Internet of Food (INoF)

Internet of Food (INoF) is a consortium of leading food production and research organizations. It is part of the SFI. The goal of INoF is to speed up innovation to create a more efficient and sustainable food production process. Increasing the collaboration among related organizations is one of the key steps to achieve this goal. Therefore, INoF aims to create a digital highway where organizations related to food production can collaborate based on the FAIR (Findable, Accessible, Interoperable and Reusable) principles to be able to create a sustainable research process, reduce loss of data, and avoid miscommunications among organizations as well as individuals. The INoF consortium also envisions the development of better sensing technologies and connecting smart sensors with the previously mentioned digital highway to increase the operational efficiency and process optimization.

Nowadays, companies use various data intensive software systems for process automation. These software components and related data are stored using proprietary databases and other storage techniques. Moreover, these artifacts themselves are heterogeneous and typically owned by their producers. Therefore, sharing these private, sophisticated, and diversified information based on the FAIR principle is very complex and currently available systems are far from optimal or nonexistent. The INoF consortium aims to address these issues by dividing its activities into four work packages.

This PDEng project was part of Work Package 1 (WP1) of the INoF consortium. Developing an infrastructure to facilitate collaboration for data-driven research projects was the main focus of this work package. As part of this infrastructure, INoF aimed to develop a reliable and flexible database system that would allow the automatic discovery and integration of various data and software artifacts. To develop such infrastructure, WP1 identified the necessity of developing related ontology and aimed to develop it interactively. In the context of this work package, all the developments were performed in steps where each step had a development and evaluation phase.

The other work packages of the INoF consortium aimed to address the other aspects that are related to efficient and optimal food production. The goal of Work Package 2 was discovering and controlling high impact variables to make food production more efficient. To do that, it combined various product properties with related sensors to find out most important attributes. The Work Package 3 aimed at competence development in the area of digital food technology through organizing workshops on selected specialized topics. The overall project management of the INoF was the main concern of Work Package 4 that included ensuring optimal alignment among ongoing activities, maintaining an inventory of similar projects, identifying opportunities of collaboration, preventing unwanted duplication, and organizing stakeholder alignment meeting on a regular interval.

1.2 Project Context

This report represents and was written as part of the graduation project for the PDEng in Software Technology program. It was executed under the WP1 of the INoF (explained in Section 1.1.2) consortium in close collaboration with Unilever, Wageningen University and Research (WUR), Nizo, and Eindhoven University of Technology (TU/e).

Unilever and Nizo are two of the most prominent names in the field of food production and research. As global leaders, they think there are myriad opportunities for innovation and optimization in the food production process. They understand the potential of digital technology and collaboration. As a result, Unilever and Nizo have become key partners of the INoF consortium along with TU/e as well as WUR. This PDEng graduation project took place in the context of this collaboration where the author of this report, as a representative of the TU/e, contributed technical expertise, Unilever and Nizo provided the majority of the domain knowledge, and WUR contributed in developing related ontologies.

In the food production domain, a majority of the work that was previously done manually (e.g., sensory taste testing) has been replaced by computer simulation and computation. This has made a significant impact by decreasing the cost, increasing accuracy, and reducing food waste. The computational units that perform these computations are known as models. Each of these units can perform a certain operation or set of operations. One example of such model is the taste model that can calculate various sensory taste perceptions based on the ingredients of certain food products.

Typically, the production of one food product involves several models, e.g., bacteriological model, cost model, and viscosity model. Companies create these models for their own use in the context of the food products that they produce. However, models created by one company have the potential to be reused in different companies or other departments within the same company for similar food products. If achieved, this kind of collaboration can allow companies to access more data and therefore, help them take better informed decisions. This can make a global impact by ensuring optimal usage of available resources and bringing the world one step closer to the sustainable food future.

Developing an infrastructure that allows sharing of models is a very complex task that requires time and resources. Moreover, the heterogeneity of the models adds even more complexities to the existing ones. However, the INoF consortium brings together several leading organizations from both the food production and technical domain, which provides a very nice opportunity for building an experimental system that can demonstrate the usefulness of a platform for sharing model results. This project aims to build a prototype that can act as a baseline for developing such platform.

1.3 Report Outline

The following chapters of this report describe all the steps performed to achieve the project goals. There are ten chapters in this report including the current one. The second chapter explains the problems that this project intended to solve. Chapter 3 talks about the stakeholders that were associated with the project. The results of the requirement analysis are explained in Chapter 4.

After the requirement and problem analysis phase, the major design challenges became explicit. To overcome these challenges a system architecture was designed, which was updated on several occasions as implicit challenges become visible. The final architecture is presented in Chapter 5. The implementation of the model sharing platform is described in Chapter 6. It is worth noting that Chapters 5 and 6 are complimentary to each other as they both describe the same system from different viewpoints and level of abstractions. Moreover, the implemented system was verified based on the elicited requirements. The findings of this step are presented in Chapter 7.

This project solves very complex cross domain problems. Therefore, for successful completion, it is essential to have proper project and process management in place, which are explained in Chapter 8.

As explained in Section 1.2, one of the main goals of this project is to develop a prototype platform for sharing model results and while doing so, identify major technical challenges for developing a similar production level system. Chapter 9 presents these findings and proposes possible future improvements.

The concluding part of this report is Chapter 10. It summarizes this report and discusses the project in retrospect.

2 Problem Analysis

The idea of a sharing infrastructure is introduced in Chapter 1. This idea is explored in more details throughout the following sections of the current chapter. First, Section 2.1 takes a closer look at the domain and explains what a model is and how are they being used. The later sections discuss the problems this PDEng project aimed to solve.

2.1 Domain Analysis

During the initial phases of this project, the main focus was on understanding the context and identifying related artifacts. Formally, these activities are known as domain analysis. It is a process by which information used in developing software systems is identified, captured, and organized with the purpose of making it reusable when creating new systems [9].

As explained in Section 1.2, this project was executed in the context of the Internet of Food (INoF) consortium. Unilever and Nizo were two of the major partners of the INoF and the key stakeholders (details in Section 3.1.2) of this PDEng project. Therefore, the majority of the domain knowledge came from them. Moreover, other INoF partners also provided information that were found to be crucial in the later stages of this project. This section only presents the domain concepts that are relevant and useful for the rest of the report.

2.1.1 Food Product Definition

A food product definition defines how the corresponding item is produced. This definition includes information regarding the required ingredients, processing steps, and packaging methods. The definition can also include certain properties of the corresponding product (e.g., expected pH of a tomato soup). The ingredients and processing steps are together called a recipe.

A recipe contains a list of necessary ingredients and their quantities for producing a certain food product. The Table 2.1 shows an example list of ingredients for producing 100 grams of tomato soup. One important thing to note in this table is the *Ingredient Code* column. Typically, there are hundreds of variants of one ingredient. These codes uniquely identify each of the ingredients and their variant. This means a code (e.g., tomato012) not only identifies an ingredient (e.g., Tomato) but also the variant of the corresponding ingredient (e.g., red tomato acquired from a certain farmer at a certain time). As the same ingredient acquired from diverse sources can have different properties (e.g., color, taste), the ingredient code makes them uniquely identifiable and provides the possibility to adjust the recipe to ensure a consistent end result.

Another aspect of using the unique code is that it enables unambiguous communication among differ-

Table 2.1: Example ingredient list for producing 100 grams of tomato soup

Ingredient Name	Ingredient Code	Amount	Unit
Salt	salt001	4	gram
Sugar	sugar002	2	gram
Vinegar	vin005	2	gram
Water	water	74	gram
Tomato	tomato012	16	gram

ent software systems. As a result, Unilever uses a similar identifier system not just in the ingredient list but also in other places while defining a food product. Moreover, the company sometimes uses one additional identifier to facilitate external communication.

As mentioned earlier, processing steps are also part of a recipe. Each of these steps are sets of mechanical and/or chemical operations performed to the ingredients of the corresponding food product. For specifying processing, Unilever follows the ISA-88 [10] standard, which is a set of guidelines and terminologies for batch processing. It also defines a consistent set of standards for physical model, procedures, and recipes.

Table 2.2: Example of a processing step represented in a tabular format

Processing Step	Equipment	Property	Value	Unit
Mixing	Mixer	Speed	500	rpm
		Duration	90	second

After careful analysis and discussion with the stakeholders, it was found that a simpler representation of the ISA-88 was sufficient for this project. Afterwards, this representation was restructured into a tabular format. An example of such a processing step is shown in Table 2.2. The important attributes of a processing step are name, one or more associated equipment, and one or more properties with their corresponding values and units.

2.1.2 Computational Model

A model is a computational unit that takes a food product definition as input and outputs certain properties of it. For example, a model can calculate the sensory taste perception of tomato soup based on its recipe. While running, a model can use all or a subset of attributes from a food product definition.

Computational models play an important role in optimizing the food production process as they reduce the experimental efforts for developing food products, which would take more time and resources otherwise. Typically, food production companies develop models for in-house usage. Each company develops, deploys, and manages their models based on their needs and available technical expertise. Therefore, models are diverse not just in nature but also in the kind of software technologies they are developed with.

In the context of Unilever, the development and deployment of models is a multistep process. Initially, models are developed by Unilever researchers and food science experts using experimental/scientific tools like Matlab. Once the models are finalized, the corresponding calculations/algorithms are documented in a model specification document. Afterwards, these specifications are used to re-implement the models that can run in a production environment. Typically, general purpose programming languages (e.g., Java, Python) are used here. Finally, these re-implemented models are made available via a service endpoint or as a library that can be used by other Unilever employees.

2.1.3 Simulation

Typically, models are run as part of a simulation. A simulation is defined by specifying one food product and one or more models. When a simulation is executed, all models in that simulation are run using the corresponding food product definition as the parameter.

2.2 Problem Definition

With consumer demand growing faster than ever, food producing companies need to speed up innovation and optimize the production processes for better yield. Models can provide important insights into the production process that can help companies make informed decisions. Access to more and better models can ensure access to more information that can translate to more optimized and sustainable food production. This is beneficial not just to the companies but also in dealing with increasing global demand.

As mentioned in Section 2.1.2, companies develop models for in-house use. This means, whenever a company needs a new model, they would typically develop it from scratch. As a result, different companies end up developing similar models that were developed separately. Since model development is a time and resource intensive task, doing it repeatedly is not a sustainable solution. As part of the Sustainable Food Initiative (SFI), this project aims to find a suitable solution for this problem.

One proposed way to avoid this repeated task and optimize the resource usage is by sharing the complete model and related artifacts (e.g., executables, codes) with other companies. The sharing can happen based on a financial agreement benefiting both model developer and buyer. However, this approach has several major disadvantages:

- Models are considered trade secrets that require resources and intensive R&D efforts to develop. If companies start selling these models, they might end up in the hands of competing organizations, which can result in serious financial consequences. Therefore, it is very unlikely that any company would want to sell its models and related artifacts.

- As mentioned in Section 2.1.2, models are typically developed for in-house use that often requires specific infrastructure to run. Most of the time, these infrastructures are proprietary and the setup is very expensive. If a company wants to buy a model, they will have to pay for the model as well as the required infrastructure. For an average size company, the total cost would be too high compared to the added value.

- Each company has its own conventions for storing and formatting data. Since models are primarily developed for in-house use, they also follow the same conventions specially for the input and output. Therefore, models from one company are very unlikely to work out of the box in the context of another company.

- Models are complex software components. Therefore, like any other pieces of software, they may have bugs. Typically, bug fixes and improvements are delivered to the customers using a continuous delivery pipeline. However, in the context of food production companies, this adds another level of complexity and even more cost.
- Sharing a model and its artifacts with another company gives them the opportunity to reverse engineer the original model and create another similar one. Afterwards, the company can claim this duplicate model as their own and sell it to even other companies. Preventing such incidents requires lots of legal complexities.

Due to these disadvantages, sharing the complete model with related artifacts is not a feasible solution. While searching for a better solution, the following three questions were identified. This project aims to prove that a software system that can answer these questions can also address above mentioned issues.

1. How can models and/or results from them be shared with other organizations without replicating the original execution environment and sharing related artifacts?
2. How can results from several computational models belonging to different organizations be made available under a unified infrastructure?
3. How can one company use a model from other companies without reformatting data to match the model inputs (e.g., different units of measurements)?

2.3 Project Goal

This project aimed to develop a prototype software platform that can address the questions mentioned in Section 2.2 and brings models from various sources under a standardized infrastructure. The idea was, this would allow one company to use models from several other companies without facing the problems listed in Section 2.2.

The concept of such an infrastructure is illustrated in Figure 2.1. This diagram demonstrates how *Company A* can use a model that belongs to *Company B* using the proposed standardized platform. To do that, *Company A* upload their data that they want to run through the model. The platform converts this data to match what the model requires and forwards the reformatted data to *Company B*. At this point, *Company B* run this data through their model and send back the results to the standardized platform. Finally, *Company A* can retrieve the results from the platform.

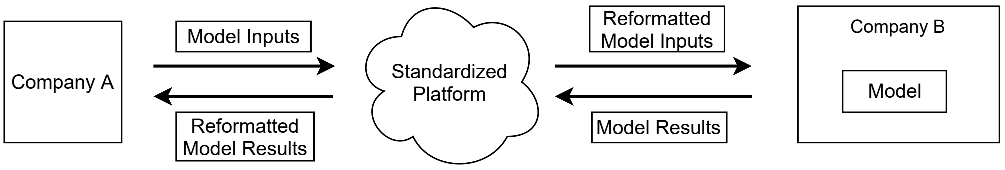


Figure 2.1: Standardized infrastructure overview for sharing model results

One of the most important aspects of the infrastructure (shown in Figure 2.1) is that *Company B* can let other companies use their model without giving up their ownership. This means *Company B* is essentially offering their model as a service (MaaS) that also has the potential to become an additional source of income. On the other hand, *Company A* can have access to one additional model without having to set up and maintain any extra infrastructure. Moreover, since the platform is acting as an intermediary, *Company A* does not have to think about formatting their data to match the model inputs.

The activities performed during this PDEng project aimed to design and develop a prototype for the platform depicted in Figure 2.1. This prototype should serve as a guideline for the future development of an enterprise level MaaS infrastructure. An example model provided by Unilever was used to avoid any legal and privacy issues. This model can calculate the sensory taste perception (saltiness, sourness, sweetness, and tomato flavor) of tomato soup based on its ingredients.

One of the primary objectives of this prototype was to demonstrate its possibilities and capabilities to the INoF partners as well as other leading food production companies to attract further collaboration. If done successfully, this can provide necessary resources and funding to develop an enterprise grade MaaS infrastructure. Moreover, technical challenges and roadblocks identified during the development of this prototype will serve as a starting point for any further extensions.

2.4 Assumptions

During the development of the MaaS prototype, several assumptions were made that were discussed and verified with the major stakeholders. These assumptions are as follows:

- In the context of this project, it was sufficient to develop a prototype of MaaS and not an enterprise grade system.
- As mentioned in Section 2.2, models are very heterogeneous in nature. However, it was sufficient for the intended prototype to only support the (example) models that were provided by Unilever.
- It was sufficient for the prototype to be deployed on a local server or general purpose cloud infrastructure to be used for demonstration purposes.
- The recipe in the food product definition contained ingredients that were chosen from a predefined list. It was sufficient to include only the ingredients provided by Unilever.
- For this prototype, it was sufficient to implements basic authentication and authorization. Moreover, administrative features (e.g., user management) were not in the scope this project.
- Detailed security measures (i.e., against hackers or malicious users) were out of the scope of this project.

2.5 Constraints

This project was executed under certain constraints. Those are:

- The duration of this project was ten months that included not only the time required for design and development of the prototype but also other PDEng related activities such as PDEng defense presentation, writing graduation report, comeback days.
- The implementation of the MaaS prototype should not use any proprietary tool, product, service, or software component. It must make use of technologies that are relatively popular, freely available, and preferably open source.
- The MaaS prototype should not implement any tools or software components that are available otherwise and can be reused.
- The prototype must be in a stable state and demonstrable to the INoF partners before the INoF team meeting scheduled in June 2020.

3 Stakeholder Analysis

The purpose of stakeholder analysis is to identify involved people, organizations, and their roles as well as concerns. This process also groups all related stakeholders according to their participation, influence, and interest in the project. This grouping is important for determining how to involve and communicate with each of these stakeholder groups throughout the project.

As explained in Section 1.2, this PDEng graduation project was part of Work Package 1 of the larger Internet of Food (INoF) consortium. Therefore, all the partner organizations of the INoF were direct or indirect stakeholders of this graduation project. This chapter focuses on the organizations and individuals that were direct stakeholders of this project.

3.1 Stakeholders by Organization

Although there were several organizations involved in the INoF, not all of them were directly connected to this project. Four organizations were the primary stakeholders of this project. Those were Eindhoven University of Technology (TU/e), Unilever, Nizo, and Wageningen University and Research (WUR).

The following sections discuss more about these organizations and the nature of their involvements. These sections also talk about the individuals from these organizations and their roles throughout this PDEng graduation project. Figure 3.1 shows an overview of these individuals and the organizations they belong to.

3.1.1 Eindhoven University of Technology (TU/e)

TU/e played two different roles in this project. The first one was connected to the academic and educational aspects. Before this project started, TU/e approved that it was complex enough to be a doctoral level project and provided supervisor to offer necessary expert knowledge as well as to make sure that the quality of work done was high enough.

The second role TU/e played was as a technical partner of the INoF project. As mentioned in Section 1.1.2, INoF aimed to make sustainable improvements to food production by increasing inter-organization collaboration using technology. TU/e aimed to promote agri-food tech, a transitional science, by linking data and minds. In this context, TU/e provided expert technical and engineering support focused on developing the underlying FAIR-based digital architectures to manage, connect, and analyze heterogeneous data. TU/e also led the Work Package 1 of the INoF.

Additionally, the Jheronimus Academy of Data Science (JADS), which is a collaboration between TU/e and the Tilburg University, provided supervision throughout the project and significantly con-

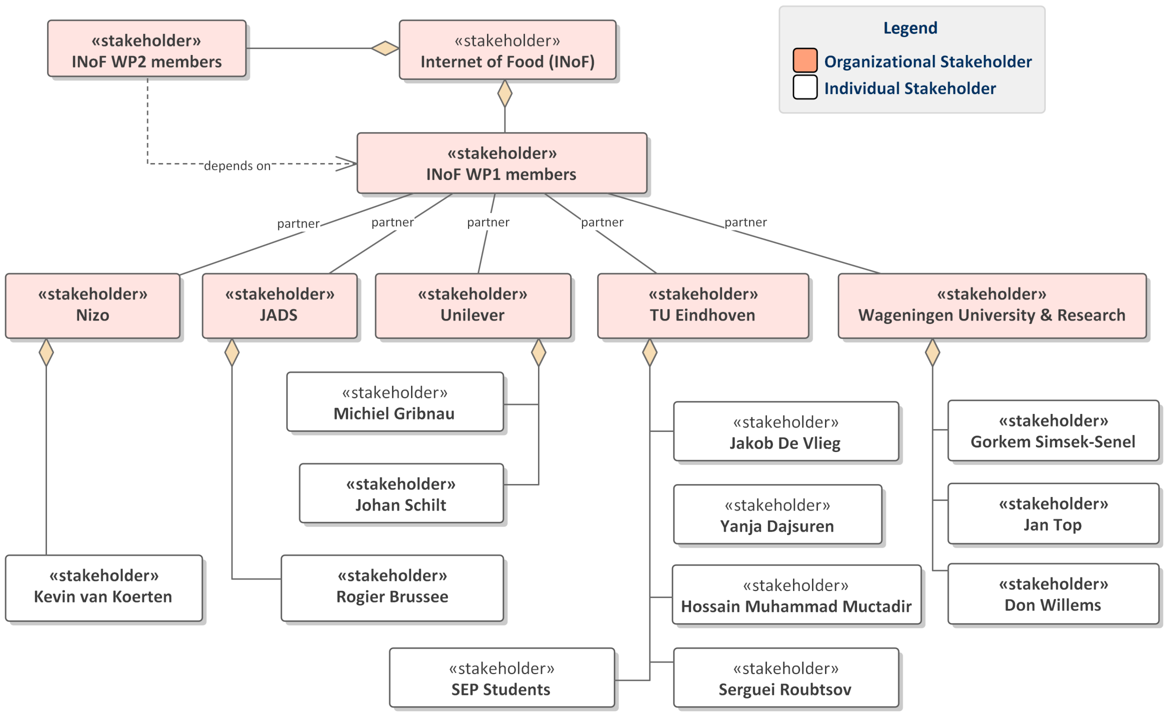

«stakeholder» INoF WP2 members
«stakeholder» Internet of Food (INoF)
Legend
Organizational Stakeholder
Individual Stakeholder
depends on
«stakeholder» INoF WP1 members
partner
partner
partner
partner
partner
«stakeholder» Nizo
«stakeholder» JADS
«stakeholder» Unilever
«stakeholder» TU Eindhoven
«stakeholder» Wageningen University & Research
«stakeholder» Michiel Gribnau
«stakeholder» Johan Schilt
«stakeholder» Jakob De Vlieg
«stakeholder» Yanja Dajsuren
«stakeholder» Hossain Muhammad Muctadir
«stakeholder» Serguei Roubtsov
«stakeholder» SEP Students
«stakeholder» Gorkem Simsek-Senel
«stakeholder» Jan Top
«stakeholder» Don Willems
«stakeholder» Kevin van Koerten
«stakeholder» Rogier Brussee

Figure 3.1: Stakeholders and related organizations

tributed with feedbacks at various stages. The Table 3.1 lists the individual stakeholders from TU/e and JADS.

Table 3.1: List of TU/e Stakeholders

Yanja Dajsuren	
Project Role	Academic Supervisor and Software Technology PDEng Program Director
Interest	• Ensure the quality of the work is high enough and is aligned with PDEng program • Provide feedback and guidelines on technical (e.g., design and architecture) as well as non-technical (e.g., communication) aspects of the project • Ensure high quality of the graduation report
Jakob de Vlieg	
Project Role	Company Supervisor and Chair of Applied Data Science Group

Interest	• Create a prototype of a data fusion infrastructure that can combine data from various sources • Ensure that the data fusion can be demonstrated using the implemented system • Demonstrate this project to attract more industry-academia collaboration • Provide necessary feedback about the design, implementation, and graduation report • Ensure that the project activities are aligned with the INoF agenda
Serguei Roubtsov	
Project Role	Academic Supervisor
Interest	• Review the graduation report and ensure that it maintains a high enough quality • Replace Yanja Dajsuren when she is not available as an academic supervisor
Rogier Brussee	
Project Role	Additional Supervisor
Interest	• Provide feedback throughout the graduation project to make sure that the INoF goals are fulfilled • Maintain high quality of the project work by contributing ideas and challenging decisions • Review and ensure high quality of the graduation report
Software Engineering Project (SEP) Students (details in Section 8.4)	
Project Role	Developer
Interest	• Obtain good grades from the SEP project by developing a good product • Understand the SEP assignment, identify requirements and their priorities • Keep the PDEng trainee (SEP customer) satisfied to ensure good feedback at the end of the project
Hossain Muhammad Muctadir	
Project Role	PDEng Software Technology Trainee
Interest	• Successful and timely completion of the PDEng graduation project • Design and develop a system that fulfills customer requirement • Gather experience in software design, architecture, communication, and project management

3.1.2 Unilever Nederland Holdings B.V

As a global leader in the food production domain, Unilever is one of the major partners of the INoF. In the context of this PDEng graduation project, Unilever is the major external stakeholder. It actively participated in all stages of this project and played the role of a client. The initial set of requirements were elicited based on the feedback of the Unilever representatives. During both the implementation and validation phase, they provided regular feedback that helped steer the project into the right direction.

Moreover, during the execution of this project, Unilever contributed the majority of the necessary domain expertise by organizing regular meetings with related personnel. It also provided the example models (explained in Section 1.2), which was essential for the project to move forward. Table 3.2 lists the individual stakeholders from Unilever.

Table 3.2: List of Unilever Stakeholders

Michiel Gribnau	
Project Role	Company Supervisor
Interest	• Ensure that the elicited requirements are aligned with expectation • Ensure that the implementation is aligned with the expectation/requirements • Develop an infrastructure that can facilitate data driven decision-making • Demonstrate the potential and usefulness of the model sharing platform to the higher management • Attract more organizations to collaborate in the INoF consortium
Johan Schilt	
Project Role	Observer
Interest	• Stay up-to-date and informed about the INoF activities • Occasionally provide expert advice

3.1.3 Wageningen University and Research (WUR)

The Wageningen University and Research (WUR) is one of the leading institutions carrying out education and research in the fields related to food production, environment, lifestyle, as well as health. The Wageningen Food and Biobased Research (WFBR) is one of the research institutes of WUR and a partner of the INoF. In the context of this PDEng project, WFBR developed the model ontology (discussed in Section 6.2.2) that was essential for the implementation of the model sharing. They also provided domain knowledge in the field of food informatics. Table 3.3 lists the stakeholders from WUR.

Table 3.3: List of WUR Stakeholders

Gorkem Simsek-Senel	
Project Role	Ontology Expert and Developer
Interest	• Provide ontology related domain knowledge (i.e., usage of ontology) • Develop model ontology and make it useful in the context of the PDEng graduation project
Don Willems	
Project Role	Ontology and IT Expert
Interest	• Provide expert advice regarding ontology based software development
Jan Top	
Project Role	Ontology Expert
Interest	• Represent WUR in the INoF consortium and provide feedbacks related to ontology

3.1.4 Other Organizations

There were several INoF partners who were directly connected to this PDEng project but did not actively participate. They mostly acted as observers and occasionally provided feedback. One of the organizations in this category is NIZO Food Research B.V. Table 3.4 lists the stakeholder from NIZO.

Table 3.4: Nizo Stakeholder

Kevin van Koerten	
Project Role	Domain Expert
Interest	• Introduce and explain gPROMS that is a simulation software widely used in food production domain • Stay up-to-date and informed about the INoF activities • Provide expert advice

4 Requirement Analysis

This chapter describes the identified use cases and the results of the requirement analysis. These results include the descriptions of functional as well as non-functional requirements.

4.1 Introduction

In [11], a requirement is defined as: "A statement that identifies a product or process operational, functional, or design characteristic or constraint, which is unambiguous, testable or measurable, and necessary for product or process acceptability (by consumers or internal quality assurance guidelines)." In software engineering projects, understanding customer requirements and their priorities at an early stage is one of the biggest challenges. An unambiguous list of requirements are essential to steer a project into the right direction. However, as the customer needs are subject to change, new requirements can emerge and the old ones may need updating during the project lifetime. In this project, a set of initial requirements were elicited using the usecase analysis and interview techniques. These requirements were revisited and updated throughout the whole duration of the project.

During the early stages of this project, understanding the context and identifying the key deliverable in relation to the Work Package 1 (WP1) of the Internet of Food (INoF) consortium were emphasized. Several meetings were arranged with the major stakeholders (Chapter 3) to discuss the short-term and long-term goals of the project as well as the corresponding deliverables. Based on these discussions, a storyboard was created that, to some extent, mimicked the major features of the proposed system in the form of a slideshow. This storyboard was used to identify potential usecases that were validated by the stakeholders. Further analysis of these usecases in the context of WP1 resulted in the set of initial requirements that were updated during several iterations based on stakeholder feedback as well as newly discovered concerns. To ensure the soundness of the requirements, each iteration went through stakeholder validation.

The following sections of this chapter explain the identified usecases, corresponding functional and non-functional requirements, as well as the relationships among them.

4.2 Usecases

An usecase is a description of potential interactions between a system and its users. In the context of this project, a representative of the food production company is a typical user of the model-sharing system. Figure 4.1 shows the potential actions that can be performed by a user.

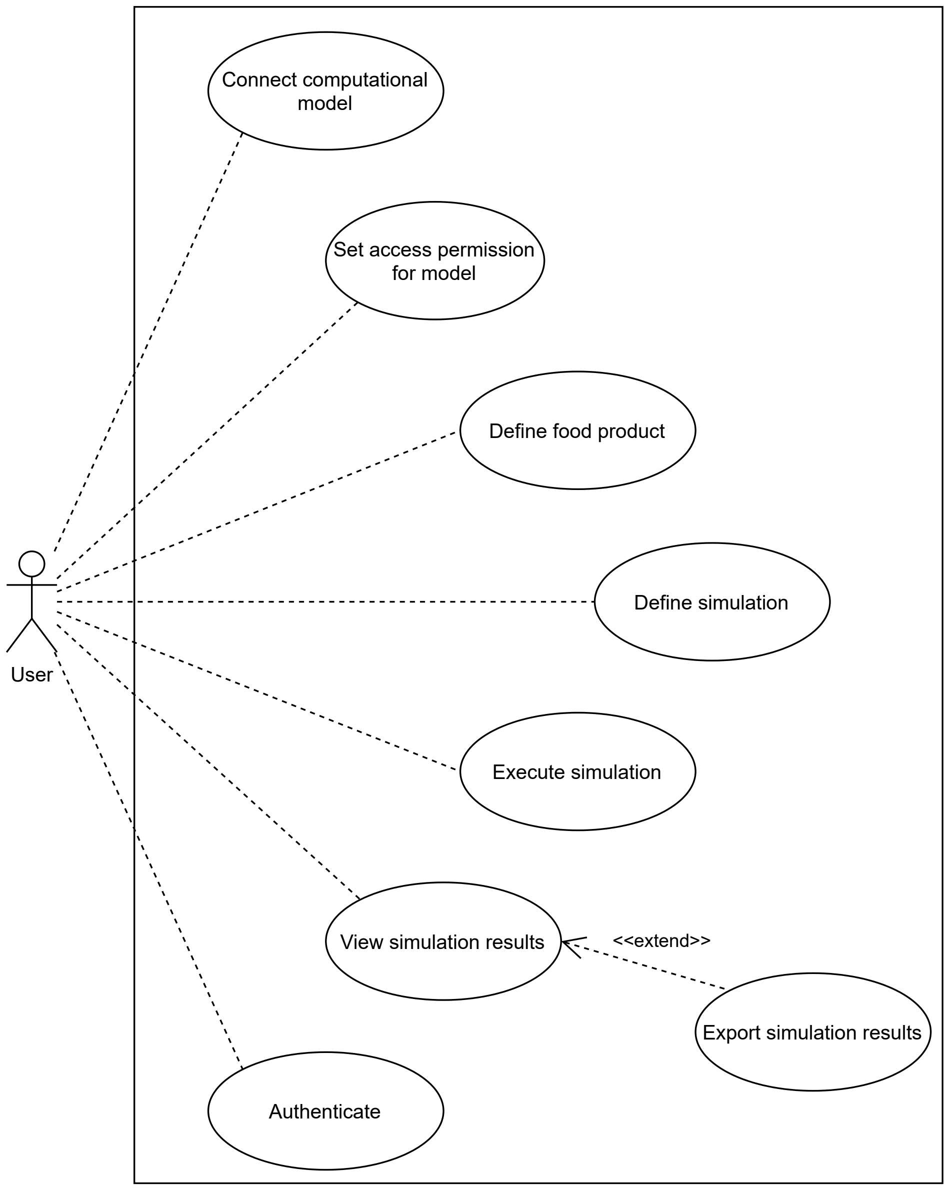


Figure 4.1: Use Cases

4.3 Requirements

In this project, the initial requirements were elicited based on the identified usecases. These requirements were analyzed further and presented to the major stakeholders for validation. The requirements established through this process were the initial set of business or functional requirements (FR) and established a baseline for the system. The non-functional requirements (NFR) were connected to the quality characteristics and constraints of the system. The FRs are explained in Section 4.3.1 and the NFRs in Section 4.3.2. Figure 4.2 depicts the connection among identified FRs and NFRs. Each requirement was given a unique identification (ID). However, these IDs do not imply any priority or order.

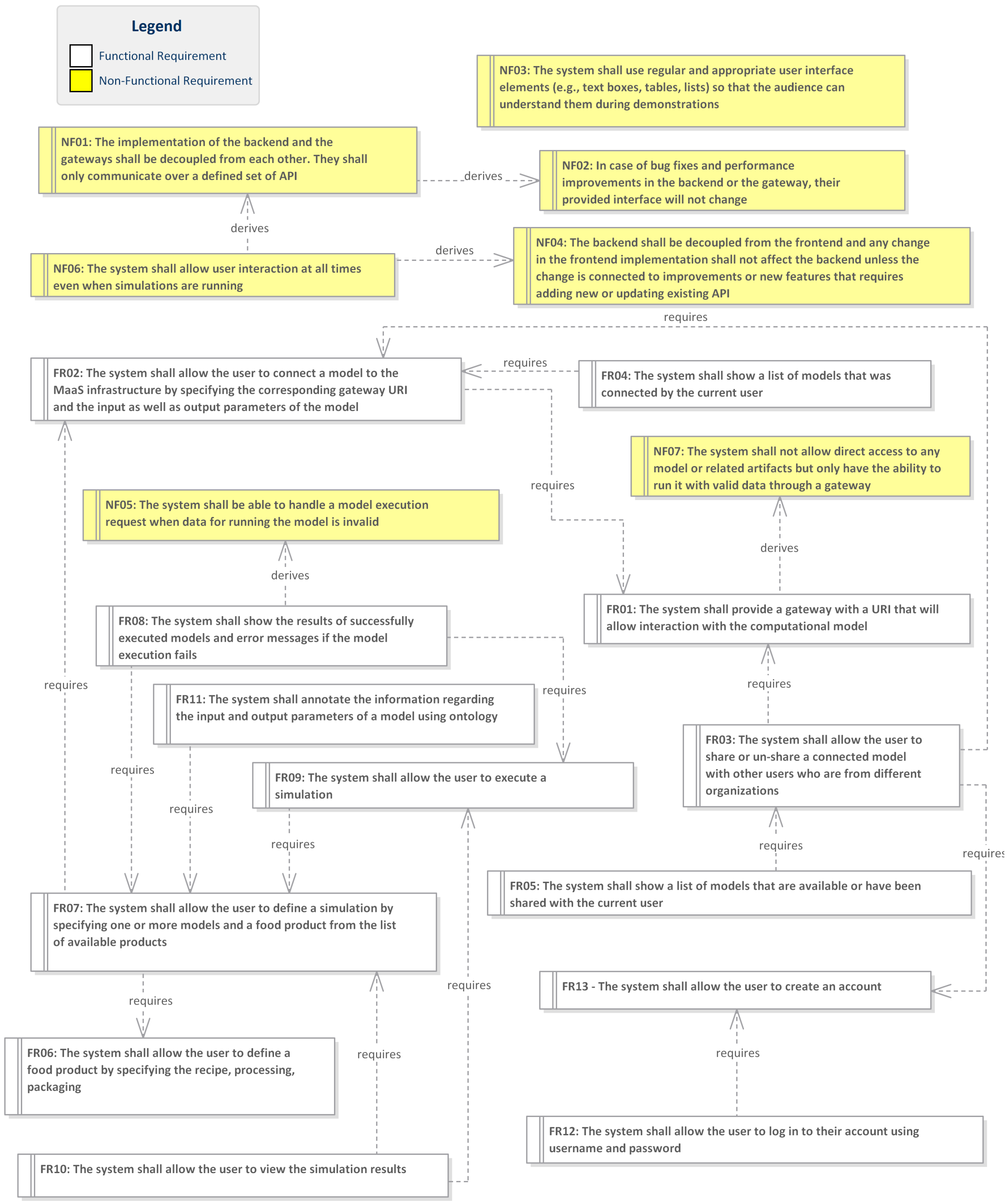

Legend
Functional Requirement
Non-Functional Requirement
NF03: The system shall use regular and appropriate user interface elements (e.g., text boxes, tables, lists) so that the audience can understand them during demonstrations
NF01: The implementation of the backend and the gateways shall be decoupled from each other. They shall only communicate over a defined set of API
NF02: In case of bug fixes and performance improvements in the backend or the gateway, their provided interface will not change
NF06: The system shall allow user interaction at all times even when simulations are running
NF04: The backend shall be decoupled from the frontend and any change in the frontend implementation shall not affect the backend unless the change is connected to improvements or new features that requires adding new or updating existing API
FR02: The system shall allow the user to connect a model to the MaaS infrastructure by specifying the corresponding gateway URI and the input as well as output parameters of the model
FR04: The system shall show a list of models that was connected by the current user
NF07: The system shall not allow direct access to any model or related artifacts but only have the ability to run it with valid data through a gateway
NF05: The system shall be able to handle a model execution request when data for running the model is invalid
FR08: The system shall show the results of successfully executed models and error messages if the model execution fails
FR01: The system shall provide a gateway with a URI that will allow interaction with the computational model
FR11: The system shall annotate the information regarding the input and output parameters of a model using ontology
FR09: The system shall allow the user to execute a simulation
FR03: The system shall allow the user to share or un-share a connected model with other users who are from different organizations
FR05: The system shall show a list of models that are available or have been shared with the current user
FR07: The system shall allow the user to define a simulation by specifying one or more models and a food product from the list of available products
FR13 - The system shall allow the user to create an account
FR06: The system shall allow the user to define a food product by specifying the recipe, processing, packaging
FR12: The system shall allow the user to log in to their account using username and password
FR10: The system shall allow the user to view the simulation results
derives
requires

Figure 4.2: Relation among the Functional and Non-Functional Requirements

4.3.1 Functional Requirements

This section lists the set of functional requirements. To make them more specific, each of the requirements follows a Requirement Breakdown Structure (RBS) [12], which is highly influenced by the Work Breakdown Structure (WBS) [13]. With RBS, each product is broken down to several requirements that are further decomposed to functions. To satisfy a requirement, the system should be able to perform all the corresponding functions. The functions can be further described with sub-functions. Figure 4.3 shows an example tree structure of the RBS. In the context of this project, it was sufficient to breakdown the requirements to functions.

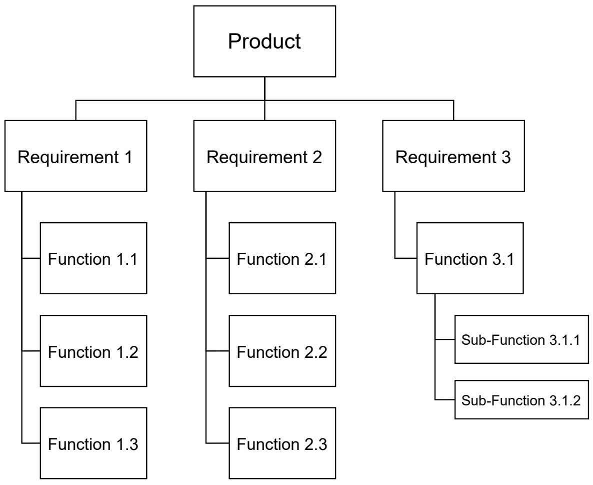


Figure 4.3: Requirement breakdown structure

The MoSCoW method [14] was used to prioritize the elicited set of requirements. The word MoSCoW is an abbreviation of four words each of which defines different priority levels. They are:

- **Must** have: Critical and non-negotiable product features that are essential for the current delivery timeline.
- **Should** have: These features are important and provide significant value.
- **Could** have: Desirable but not necessary needs that can improve the usability.
- **Won't** have: These are initiatives that are least critical and lowest priority.

Following are the list of functional requirements and their corresponding functions:

- **FR01: The system shall provide a gateway with a URI that will allow interaction with the computational model.**
  Priority: Must
  Functions:
  - Each gateway can be uniquely identifiable and accessible from the model sharing back-end.
  - Each gateway will accept data that is necessary to run the corresponding model.
  - Each gateway will run the underlying model with the provided data and reply with generated results.

- **FR02: The system shall allow the user to connect a model to the MaaS infrastructure by specifying the corresponding gateway URI and the input as well as output parameters of the model.**
  Priority: Must
  Depends on: FR01
  Functions:

  - The user can add and update the model gateway URI.
  - The user can define the input and output parameters of the model.
  - The user can define the cost in euros for running a model.
  - The user can define additional attributes of the model (e.g., description).

- **FR03: The system shall allow the user to share or un-share a connected model with other users who are from different organizations.**
  Priority: Must
  Depends on: FR01, FR02, FR13
  Functions:
  - When a model is shared with an organization, personnel belonging to that organization shall be able to view and/or execute the corresponding model.
  - When a model is un-shared, it should only be usable to the owner organization.

- **FR04: The system shall show a list of models that was connected by the current user.**
  Priority: Must
  Depends on: FR03
  Functions:
  - The system shall show which models are shared with which organization.
  - The system shall show the number of times the model was run.

- **FR05: The system shall show a list of models that are available or have been shared with the current user.**
  Priority: Must
  Depends on: FR03
  Functions:
  - Models can be shared with one or more organizations.
  - A model shared with an organization is visible to all representatives of that organization.
  - The list of models shows the cost for running the models and their input-output parameter information.

- **FR06: The system shall allow the user to define a food product by specifying the recipe, processing, packaging.**
  Priority: Must
  Functions:
  - When creating a food product, the system shall allow the user to define processing steps necessary for producing the corresponding product.
  - When creating a food product, the system shall allow the user to define ingredients and their quantities necessary for producing the corresponding food product.
  - When creating a food product, the system shall allow the user to define the expected chemical properties of the product.
  - When creating a food product, the system shall allow the user to define how the product is packaged.

- **FR07: The system shall allow the user to define a simulation by specifying one or more models and a food product from the list of available products.**
  Priority: Must

Depends on: FR02, FR06
Functions:

- When creating a simulation, the system shall allow the user to choose only models that are available to the current user.
- When creating a simulation, the system shall allow the user to choose one or more food products to run the corresponding simulation on.

• **FR08: The system shall show the results of successfully executed models and error messages if the model execution fails.**
Priority: Must
Depends on: FR07, FR09
Functions:
  - In case of successful execution of a simulation, the system must return the results of the simulation.
  - In case of failed execution of a simulation, the system must return the appropriate error messages.

• **FR09: The system shall allow the user to execute a simulation.**
Priority: Must
Depends on: FR07
Functions:
  - The system shall allow the user to execute a simulation that has previously been defined by the current user.
  - The simulation can only execute if the product definition is a valid input to run the models in the corresponding simulation.

• **FR10: The system shall allow the user to view the simulation results.**
Priority: Must
Depends on: FR07, FR09
Functions:
  - The system shall provide a URI through which the simulation results can be accessed.
  - When the simulation is not successful, the system shall show related error messages. In this case, the simulation result might be incomplete.

• **FR11: The system shall annotate the information regarding the input and output parameters of a model using ontology.**
Priority: Must
Depends on: FR09
Functions:
  - The input and output parameters stored using ontology must specify parameter name, physical units including the conversion coefficient to/from other similar commonly used units.

• **FR12: The system shall allow the user to log in to their account using username and password.**
Priority: Must

Depends on: FR13
Functions:

  - When login is not successful, the system will show an error message with the reason.
  - The system shall allow the user to logout from their account.

- **FR13: The system shall allow the user to create an account.**
  Priority: Must
  Functions:
  - When a user registers, the system shall verify the user's identity by sending a verification email.
  - The system shall allow the user to change his/her account information (name, address, password, company.)

4.3.2 Non-Functional Requirements

- NF01: The implementation of the backend and the gateways shall be decoupled from each other. They shall only communicate over a defined set of API.
- NF02: In case of bug fixes and performance improvements in the backend or the gateway, their provided interface will not change.
- NF03: The system shall use regular and appropriate user interface elements (e.g., text boxes, tables, lists) so that the audience can understand them during demonstrations.
- NF04: The backend shall be decoupled from the frontend and any change in the frontend implementation shall not affect the backend unless the change is connected to improvements or new features that requires adding new or updating existing API.
- NF05: The system shall be able to handle a model execution request when data for running the model is invalid.
- NF06: The system shall allow user interaction at all times even when simulations are running.
- NF07: The system shall not allow direct access to any model or related artifacts but only have the ability to run it with valid data through a gateway.

5 System Architecture and Design

To be able to implement a software system that satisfies the customer requirements, proper planning is absolutely essential. An architecture is one of the most important parts of that plan. As defined in [15], "Architecture is the set of structures needed to reason about the system, which consists of software elements, relations among them, and properties of both." In other words, an architecture of a system is a detailed definition of the individual components that communicate and work together to make up the corresponding system. Architecture provides the possibility to look at the intended system from different viewpoints and discover potential problems even before the implementation. An architecture can also be used to show and explain the system design to the related stakeholders from different levels of abstractions.

This chapter describes the architecture that guided the development of the Model as a Service (MaaS) infrastructure in the context of the Internet of Food (INoF) consortium. At first, activities that the MaaS system is expected to perform are discussed in Section 5.1. This is followed by a discussion on the key design challenges and decisions made to solve them (Section 5.2). In the next sections, the software system overview and the descriptions of the components that make up the MaaS are explained (Section 5.4).

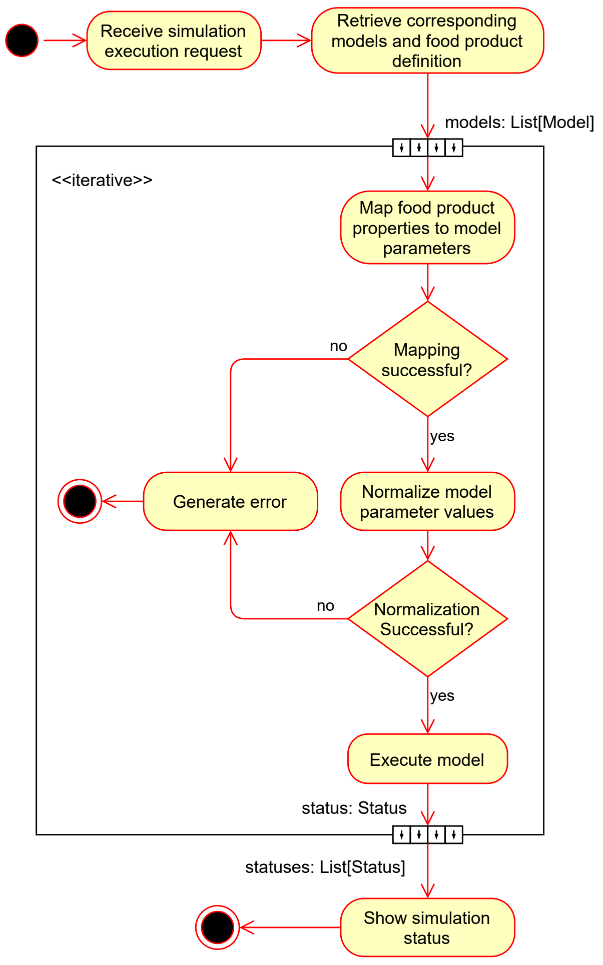
5.1 Expected Activities of MaaS

The requirement analysis in Chapter 4 explains the usecases and the requirements that the MaaS system is expected to fulfill. The current section discusses how each of those requirements are connected to each other. For explaining the execution of simulation and view corresponding results, activity diagrams [16] are used, which are graphical representations of workflows of step-wise activities and actions.

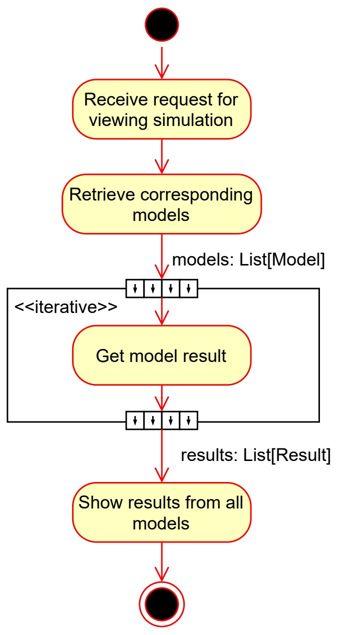
As discussed in Section 2.3, the primary focus of the MaaS infrastructure is to be able to execute models belonging to other organizations to increase productivity and collaboration. Models are run as part of a simulation. As introduced in Chapter 2, a simulation is a pairing of one food product definition and one or more models, each of which can be owned by different organizations.

To be able to run a model, the first step is to connect it. This means specifying where and how the MaaS system can locate a model. This step is very important because each model typically runs in its organization's own environment and MaaS infrastructure won't be able to communicate with it without appropriate connection information. The next step is to define a food product that includes required ingredients, processing steps, and packaging methods (details in 2.1.1). These properties are later used as input parameters for models.

After connecting models and defining food product, it is now possible to define a simulation. After


Receive simulation execution request
Retrieve corresponding models and food product definition
models: List[Model]
<<iterative>>
Map food product properties to model parameters
Mapping successful?
no
yes
Generate error
Normalize model parameter values
Normalization Successful?
no
yes
Execute model
status: Status
statuses: List[Status]
Show simulation status

(a) Activities for running all models in a simulation


Receive request for viewing simulation
Retrieve corresponding models
models: List[Model]
<<iterative>>
Get model result
results: List[Result]
Show results from all models

(b) Activities for showing results of all models in a simulation

Figure 5.1: Activity diagrams for running a simulation and viewing results

this step, the user can make a request to run the simulation. Running a simulation means, run all the models specified in the simulation and use the properties of the corresponding food product definition as input for each of those models. Following this step, the user can check whether the models in the simulation have finished running. As soon as the model runs are completed, the user can see the results from each of the models.

Figure 5.1a shows the activities involved in running a simulation. Since several models can be part of a simulation, the expansion region (black box annotated with *iterative*) in this figure shows the set activities that are executed separately for each model. Similar strategy is used to collect and visualize the results from each model execution (Figure 5.1b) as soon as the executions are complete.

5.2 Design Challenges

The following key design challenges were derived from the functional and non-functional requirements described in Chapter 4:

5.2.1 Heterogeneous Data Formats and Interoperability

From the problem analysis in Chapter 2, it is clear that different organizations follow different standards to structure and format their data and models. Often times, the input and output parameters of the models developed by different organizations differ in several ways. These variations can include the way the data is stored or transferred (e.g., binary file stream, XML, JSON), naming of different parameters or units of measurements. One of the major design challenges is to be able to achieve interoperability amongst these differences.

5.2.2 Localization and Accessibility

Typically, organizations develop their models for in-house use. This means, the models are developed, deployed, and used in the corresponding organization's own environment. Therefore, models from different organizations are physically deployed in different locations and are often available only from the corresponding company environment. As a result, being able to access and communicate with these models from an external environment is one of the major design challenges.

5.2.3 Privacy and Ownership

Developing models is a resource intensive task. Once developed, these models are owned by the corresponding organizations and are often regarded as trade secrets. Therefore, it is very unlikely that the corresponding organization would allow any kind of external physical access to the source code, executable, or any artifacts related to their models. On the other hand, the goal of the MaaS infrastructure is to facilitate collaboration among different organizations so that they can use each other's models. The key challenge here is to allow sharing the models without giving away their ownership.

5.2.4 Model Discovery

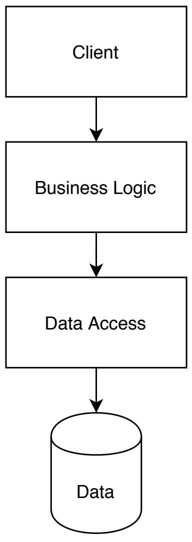
In the context of the INoF, the purpose of MaaS infrastructure is to allow organizations to use models from other organizations. Before a model can be used, it is essential to find the model and understand what kind of computation it performs on which data. Therefore, the MaaS architecture should not only allow sharing of the models but also provide the possibility to search them.

5.3 Design Decisions

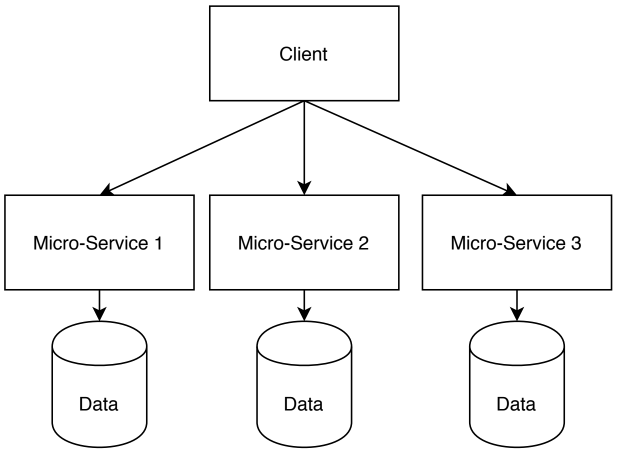
Considering the design challenges mentioned in Section 5.2, several alternative design choices were evaluated. The major design decisions are explained below.

5.3.1 Microservice Architecture

As defined by Martin Fowler in [17], "The microservice architectural style is an approach for developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deploy-able by fully automated deployment machinery." This is different from the traditional way of building applications where the whole system is built as a single unit.



(a) Monolithic Architecture



(b) Microservice Architecture

Figure 5.2: Differences between Monolithic and Microservice Architecture

Business applications typically have three basic building blocks: a client that is responsible for exchanging information with the user, a backend that implements business logic and a data storage. The backend typically provides all its services through an HTTP interface. As shown in Figure 5.2a, the backend receives requests from the client, executes necessary business logic, performs necessary data updates, and replies to the client with appropriate response. This kind of backend is called a monolith and this architecture is known as monolithic architecture. Here all business logic are built together in a single executable and runs as a single process.

From a development point of view, monolithic applications are difficult to maintain as small changes in the system require the whole application to be rebuilt. As discussed in [18], this is especially problematic when multiple developers are working on the same code base. They need to make sure that the whole system keeps working when changes are made to the code. These complexities slow down the development process. Scaling is also an issue with monolithic architecture as critical parts of the application cannot be scaled separately as the entire application is closely tied together.

Microservice architecture solves the majority of the problems in the monolithic architecture by dividing the whole application into suites of services. These services define a strict boundary and can be developed, built, as well as deployed independently. This architecture also allows independent scaling of the different services.

In the context of the MaaS infrastructure, localization and accessibility are major design challenges as explained in Section 5.2.2. Certain aspects of the microservice architecture can help to overcome this challenge.

Organizations have their models running in their own environment. To be able to interact with these models from outside their environment, gateways were set up for each of the models (Figure 5.3). To the outside world, models are represented by their corresponding gateways that are microservices and expose the functionalities provided by the respective models as an HTTP API.

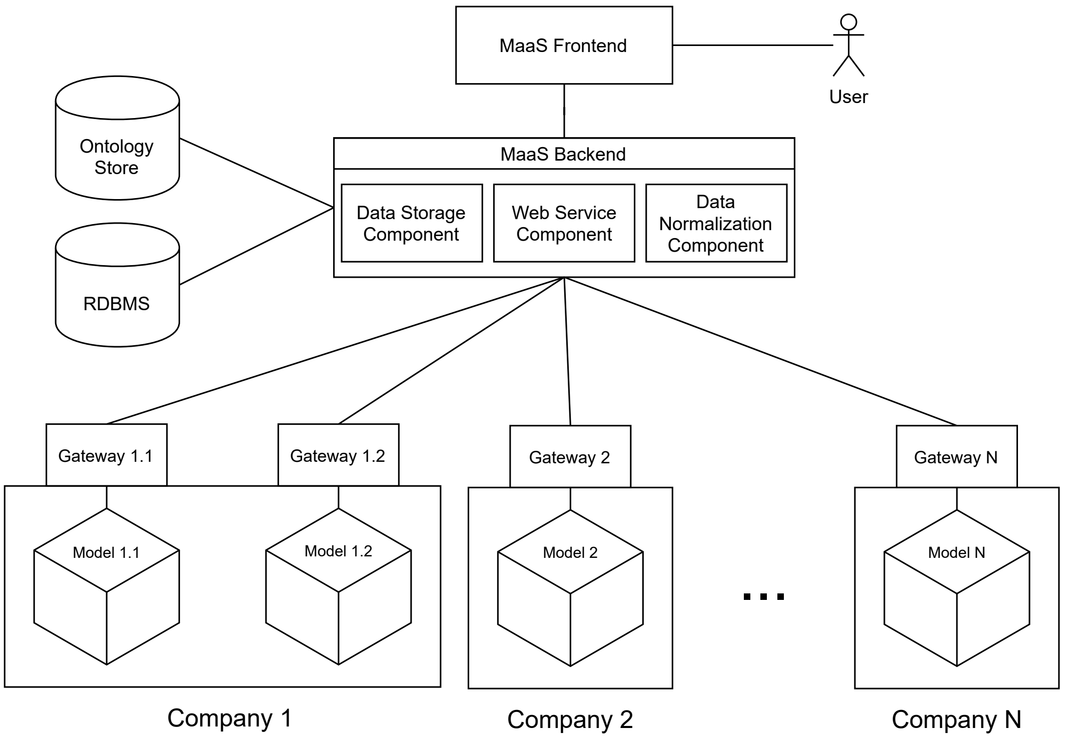


Figure 5.3: MaaS microservice architecture

In this microservice-based architecture, the gateways are independent of each other. Therefore, they can have their own data storage or specialized implementation for running a model. However, the gateways must implement a set of standardized HTTP API so that they can be interacted with.

In addition to the gateways, there is a backend that acts as a broker for the client requests and forwards them to the appropriate gateway. Additionally, the backend has the responsibility to store and manage information about the models, perform authentication, as well as data normalization if necessary.

5.3.2 Ontology Based Data Normalization

As explained in Section 5.2.1, heterogeneous data is one of key challenges when it comes to interoperability. Achieving a complete data interoperability is a very broad research topic and, therefore, is not in the scope of this project. However, the MaaS infrastructure uses ontology (explained in Section 6.2.2) to overcome part of that challenge with the Data Normalization Component shown in Figure 5.3. The responsibility of this component is to map the user provided data to the input of the models. While doing so, the normalization component also converts the data to an appropriate unit of measurement if necessary.

5.3.3 Heterogeneous Database System

Shared models need to be discoverable before they can be used by different organizations. Therefore, model discovery is one of the key challenges for the MaaS infrastructure as discussed in Section 5.2.4. Another challenge discussed in Section 5.2.1 is the heterogeneous nature of the model input/output parameters. These challenges imply the necessity of some sort of storage mechanism. However, keeping the nature of the data in mind, the traditional structured data storage is not sufficient.

To solve this problem, a heterogeneous database system has been introduced, which is shown in Figure 5.3. Depending on the data to be stored, this system uses different database technologies for data storage. One of the databases is the traditional relational database that is used for storing structured information such as user credentials, model access permissions, and food product details. To store the model input/output data that do not follow any specific structure or schema, a graph-based data storage was used, which is labeled as the Ontology Store in Figure 5.3. It stores data as triples and is a perfect match for storing the model input/output information that is represented in terms of ontology, as mentioned in Section 5.3.2.

5.4 Software Component Decomposition

Based on the requirements described in Chapter 4, design challenges mentioned in Section 5.2 and the system overview depicted in Figure 5.3, the whole MaaS infrastructure has been decomposed into several components. This section describes what these components are and how they are connected. The components are also shown in Figure 5.4.

5.4.1 Gateway and Computational Model

The gateway is introduced in Section 5.3.1 where the microservice architecture is discussed. The same section explains that the computation models, which typically run in an internal company environment, are represented by the gateway to the external users. The communication between the computational models and the respective gateway takes place over the *IExecute* interface. This interface has no defined structure and depends entirely upon the implementation of the corresponding model. This lack of structure does not affect the rest of the system since it stays hidden behind the gateway.

According to the microservice architecture explained in Section 5.3.1, the gateways can have independent implementation as long as they have a set of APIs defined as *ISimulate* that are accessible over HTTP. The definition of this interface can be found in the implementation chapter under Section 6.2.1.

5.4.2 MaaS Infrastructure Backend

As explained in Section 5.3.1, the backend component has the role of broker in the MaaS infrastructure. It facilitates seamless communication between the frontend and computational models via the gateway. This communication is facilitated by the *WebService* sub-component via the *IWebService* interface that provides a set of HTTP APIs. Section 6.2.1 explains the implementation of this interface.

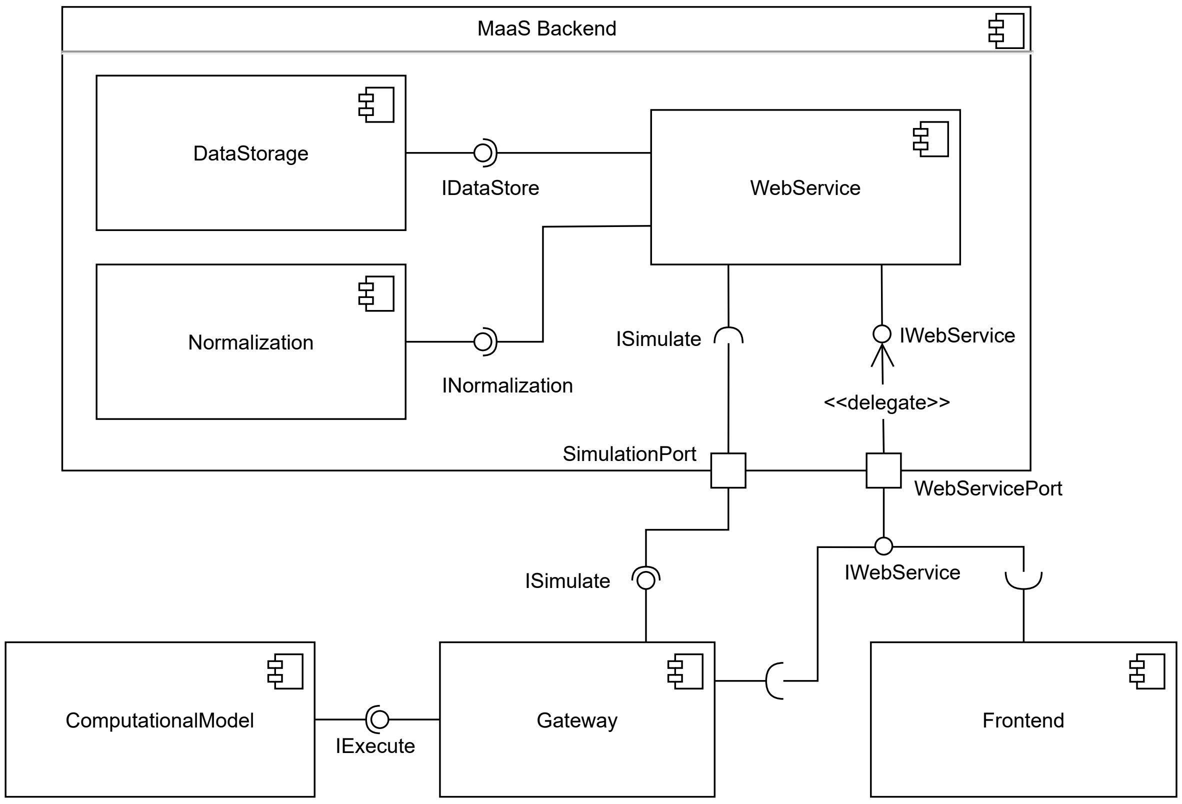


Figure 5.4: Software system component diagram of the MaaS infrastructure

One of the major roles of the backend is to make sure that the models are executed with appropriate input parameters. This means, the backend maps food product properties to appropriate input parameters of the corresponding model before sending the request to the gateway. During this mapping, backend also converts the mapped food product properties to appropriate units of measurements if they are different from the inputs of the corresponding model. This whole process has been named Normalize and the sub-component responsible for this task is called Normalization. According to American Heritage Dictionary [19], the word "normalize", is defined as "to make regular and consistent," which is exactly what the Normalization sub-component is responsible for.

The backend stores all related data including the food product properties, models, and simulations using the *DataStorage* sub-component. It is responsible for providing a persistence service by communicating with the heterogeneous database system, which is explained in Section 5.3.3.

5.4.3 Frontend

This component is the web-based user interface for the MaaS system that can be viewed by any modern browser (e.g., Firefox, Google Chrome). The purpose of this component is to provide the user with an easy-to-understand and intuitive interface to consume the services provided by the MaaS backend. The frontend is also responsible for client-side data validation.

6 Implementation

This chapter describes the implementation of the prototype of the Model as a Service (MaaS) infrastructure that is introduced in Section 2.3. The Chapter 5 discusses the major design challenges for this prototype and defines an architecture to overcome them. The prototype was implemented following this architecture.

The discussion in this chapter begins with the choice of technologies for implementing the components introduced in Section 5.4. The later sections explain how the major design decisions listed in Section 5.3 are implemented.

6.1 Technology Choice

The component decomposition for the MaaS infrastructure is depicted in Figure 5.4. The technologies necessary to implement these components were chosen based on the following criteria:

- Open source: The chosen technologies should be freely available and preferably open source. It was also required by the stakeholders to avoid proprietary technologies as much as possible.
- License: The chosen technologies should not imply any restrictions for the usage and distribution of the developed prototype.
- Fast development: Since the goal of this project is to develop a prototype for the MaaS infrastructure, the chosen technology should allow high throughput.
- Strong ecosystem and support: For maintainability purposes, it was important to choose technologies that had a strong ecosystem of reusable libraries and a good community support.
- Platform independent: Although Linux was the primary platform for development, the developed prototype should run on most of the popular platforms that include operating systems (e.g. Windows, macOS) and containers (e.g. Docker).
- Lightweight: As defined in Section 5.3.1, the prototype should follow the microservice architecture. Therefore, it was important to choose technologies that are lightweight and requires limited resources.
- Familiarity: As the duration of this project and all related activities is limited to ten months, the chosen technology should be familiar to the trainee to minimize the required learning time.
- Learning goal: Although familiarity is an important criterion, the chosen technologies should also fulfill the learning goals of the trainee.

Besides these general criteria, each technology may have specific additional reasons for getting accepted or rejected.

6.1.1 Web Service Framework

According to the architecture defined in Section 5.3.1, both the backend and the gateways involved web services. Nowadays, a variety of web service frameworks are available in almost all popular programming languages. Based on previous experience of the PDEng trainee, Python and C# was considered for the language of choice. Several web service frameworks of both languages were compared and the results are shown in Table 6.1.

After considering several aspects, Python was chosen for the programming language as it is lightweight, has a strong community and is very well known in the food production domain. For the web service framework, Flask was chosen for its versatility, simplicity, strong community support, and lightweightness. This combination also provided a good balance between familiarity and learning goals of the trainee.

Table 6.1: Criteria matrix for choosing web service framework

	Python		C#	
	Flask	Django	ASP.NET Core	ASP.NET 4.x
Open Source	Yes	Yes	Yes	No
Re-distribution restriction	None	None	None	None
Fast development	Yes	No	Yes	No
Ecosystem and support	Yes	Yes	Relatively new but growing	Yes
Platform support	All major	All major	All major	Windows
Lightweight	Very	No	Yes	No
Familiarity	No	No	Yes	Yes
Learning	Yes	Yes	Yes	No

6.1.2 Data Storage and Related Libraries

The heterogeneous database system is introduced in Section 5.3.3. This includes traditional relational databases for storing structured data as well as graph based databases for storing ontology and semi-structured data. The usage of ontology is explained in Section 6.2.2.

Table 6.2 shows the criteria matrix for choosing the database technologies. Although, MySQL and PostgreSQL are similar in many ways, the latter was chosen because it is open source and has better community support. Moreover, GraphDB[1] was selected as the graph database of choice because of its direct support for Resource Description Framework (RDF) files.

Additional drivers or libraries are needed for reading and writing data to the chosen databases. Object Relational Mapping (ORM) libraries are very popular for communicating with relational databases like PostgreSQL. They provide an object-oriented interface for reading, writing and updating data without having to write raw SQL queries. This project uses SQLAlchemy[2] because it is a well known

[1]http://graphdb.ontotext.com/

[2]https://www.sqlalchemy.org/

Table 6.2: Criteria matrix for choosing databases

	Relational Databases		Graph Databases	
	PostgreSQL	MySQL	Neo4j	**GraphDB**
Open source	Yes	Community edition available	Yes	Community edition available
Community support	Yes	Yes	Limited	Limited
RDF support	No	No	Not explicitly	Yes
Platform support	All major	All major	All major	All major
Docker supported	Yes	Yes	Yes	Yes but requires additional steps
Familiarity	Limited	Yes	No	No
Learning	Yes	Yes	Yes	Yes

and one of the most feature rich open source Python ORM.

On the other hand, GraphDB supports SPARQL, which is an RDF query language. To execute these queries, GraphDB provides a set of REST like APIs that accepts SPARQL query as string and returns the results in JSON format. SPARQLWrapper[3] was used for preparing and executing queries on GraphDB server. It is an open source Python library that provides a wrapper around SPARQL service for remotely executing queries.

6.2 Implementation of the Major Design Decisions

The architecture chapter of this report (Chapter 5) analyzes and lists the major design challenges (Section 5.2). To solve them, several design decisions were made (Section 5.3) after considering different alternatives. This chapter revisits these design decisions and explains how they were implemented.

6.2.1 Microservice Architecture

The microservice architecture is introduced in Section 5.3.1. This architectural style is the blueprint for building all other components or services. It also defines which components are communicating with each other and how. As depicted in Figure 5.3, the MaaS infrastructure has four major types of services. Each of them are explained in the following sections.

MaaS Infrastructure Backend

As defined in Section 5.4, the backend facilitates the communication between the frontend and the gateways through a set of web service endpoints. The services provided by these endpoints makes use of a complex set of data processors, routing modules and configurations. The Flask framework was chosen (Section 6.1.1) to implement the web service.

[3]https://github.com/RDFLib/sparqlwrapper

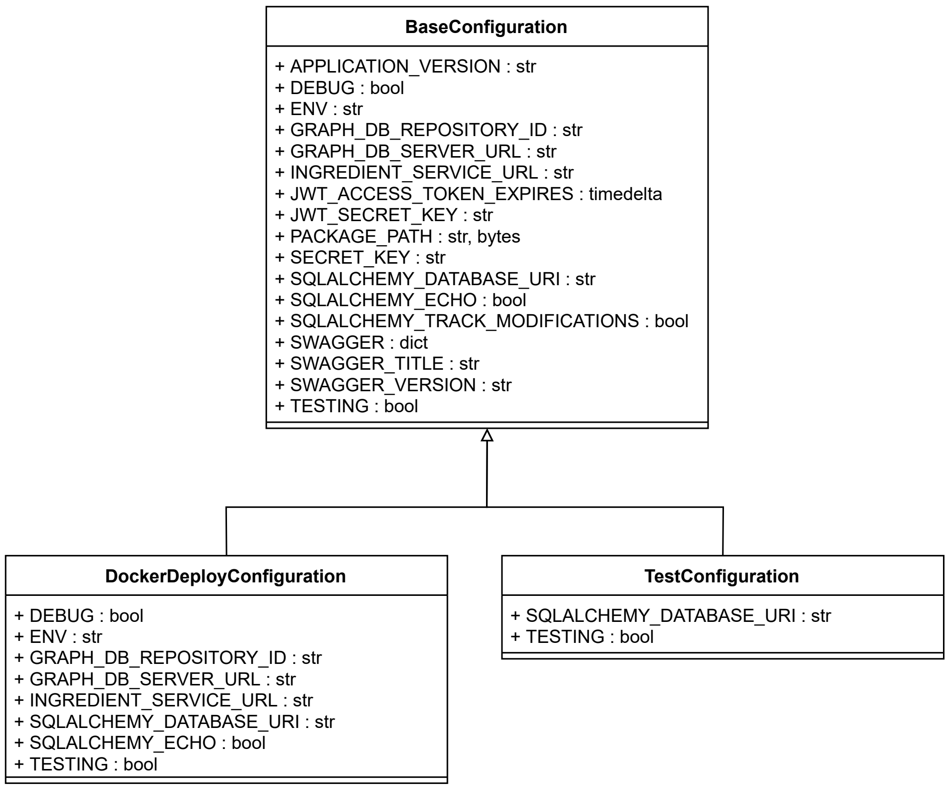

BaseConfiguration
+ APPLICATION_VERSION : str
+ DEBUG : bool
+ ENV : str
+ GRAPH_DB_REPOSITORY_ID : str
+ GRAPH_DB_SERVER_URL : str
+ INGREDIENT_SERVICE_URL : str
+ JWT_ACCESS_TOKEN_EXPIRES : timedelta
+ JWT_SECRET_KEY : str
+ PACKAGE_PATH : str, bytes
+ SECRET_KEY : str
+ SQLALCHEMY_DATABASE_URI : str
+ SQLALCHEMY_ECHO : bool
+ SQLALCHEMY_TRACK_MODIFICATIONS : bool
+ SWAGGER : dict
+ SWAGGER_TITLE : str
+ SWAGGER_VERSION : str
+ TESTING : bool
DockerDeployConfiguration
+ DEBUG : bool
+ ENV : str
+ GRAPH_DB_REPOSITORY_ID : str
+ GRAPH_DB_SERVER_URL : str
+ INGREDIENT_SERVICE_URL : str
+ SQLALCHEMY_DATABASE_URI : str
+ SQLALCHEMY_ECHO : bool
+ TESTING : bool
TestConfiguration
+ SQLALCHEMY_DATABASE_URI : str
+ TESTING : bool

Figure 6.1: Configurations for backend web service

Backend Configuration: As Flask is a lightweight web service framework, it does not provide any additional functionality or smart configuration out of the box. Therefore, the first task was to configure the web service to be able to host necessary endpoints. A class structure with three classes was created that held the configuration values. As depicted in Figure 6.1, the *BaseConfiguration* class holds all the configuration properties and provides default values for them. These default values are ideal for running the web service on local machine (e.g. localhost). There are two additional configuration classes that extends from *BaseConfiguration*. The *DockerDeployConfiguration* class overrides several properties from its parent that are necessary to deploy the web service on a Docker container. Moreover, the *TestConfiguration* is used for unit testing the endpoints. The configuration classes hold properties for not only the web service but also application specific configuration values such as, database connection strings, URL for other microservices, secret keys used in user authentication. Storing these properties as Flask configuration makes it possible to access them from different parts of the backend code.

Endpoint Routing: In the context of web services, routing is the act of mapping URLs or endpoints to actions. As shown in the component diagram (Figure 5.4), the backend provides an *IWebService* interface that is consumed by the frontend and gateway for exchanging data.

The endpoints provided by the *IWebService* interface is shown in Figure 6.2. Each of the endpoints defines two essential parts: HTTP verb [20] (e.g. GET, POST) and an URL (e.g. `/api/auth_token`). Endpoints can optionally have a parameter and/or response data that are always formatted as JSON. However, regardless of the presence of a response data, all endpoints returns a response

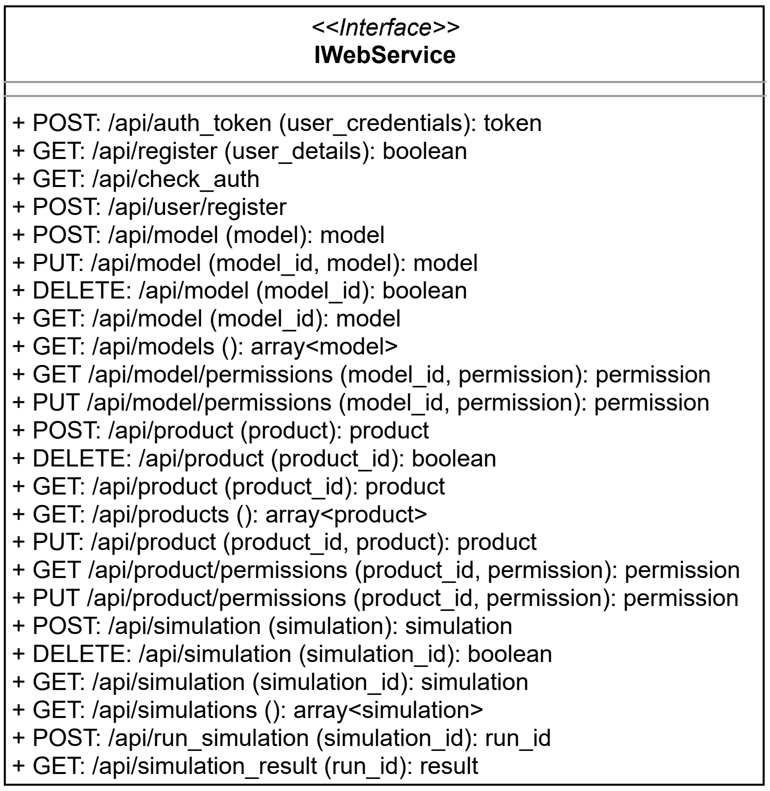

<<*Interface*>>
IWebService
+ POST: /api/auth_token (user_credentials): token
+ GET: /api/register (user_details): boolean
+ GET: /api/check_auth
+ POST: /api/user/register
+ POST: /api/model (model): model
+ PUT: /api/model (model_id, model): model
+ DELETE: /api/model (model_id): boolean
+ GET: /api/model (model_id): model
+ GET: /api/models (): array<model>
+ GET /api/model/permissions (model_id, permission): permission
+ PUT /api/model/permissions (model_id, permission): permission
+ POST: /api/product (product): product
+ DELETE: /api/product (product_id): boolean
+ GET: /api/product (product_id): product
+ GET: /api/products (): array<product>
+ PUT: /api/product (product_id, product): product
+ GET /api/product/permissions (product_id, permission): permission
+ PUT /api/product/permissions (product_id, permission): permission
+ POST: /api/simulation (simulation): simulation
+ DELETE: /api/simulation (simulation_id): boolean
+ GET: /api/simulation (simulation_id): simulation
+ GET: /api/simulations (): array<simulation>
+ POST: /api/run_simulation (simulation_id): run_id
+ GET: /api/simulation_result (run_id): result

Figure 6.2: *IWebService* Interface

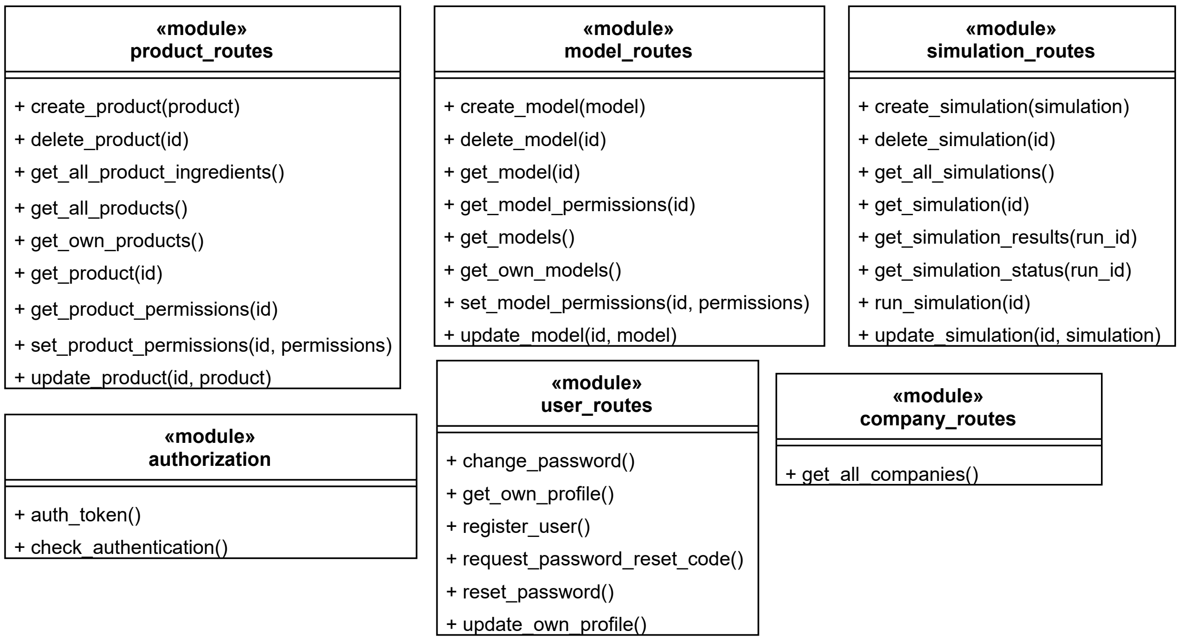


Figure 6.3: Routing modules of the web service component

status code (e.g. 200 for success) that are aligned with the HTTP standard [20]. These endpoints are explained further in Appendix B.

The endpoints shown in Figure 6.2 are mapped to python functions using the Flask library. Based on functionality, these functions are grouped into Python modules, which are shown in Figure 6.3. The endpoints and the map functions has an one-to-one mapping. This means, each endpoint is mapped to exactly one route function and vice versa. For example, the $POST : /api/auth\_token$ route is mapped to *authorization.auth_token()* function. The main business logic is implemented in the routing functions.

Data Structures and Heterogeneous Database System: As mentioned in Section 6.1.2, this project makes use of a heterogeneous database system: PostgreSQL and GraphDB. The SQLAlchemy ORM library is used to read, write and update data into the PostgresSQL database. This ORM library provides *orm.Query* class that contains methods for executing various queries in the database. For convenience, the *orm.Query* class has been extended in several stages that is depicted in Figure 6.4. The most important extension is the *PermissionFilteredQuery* class containing methods for querying entities that is filtered by accessibility of the logged-in user.

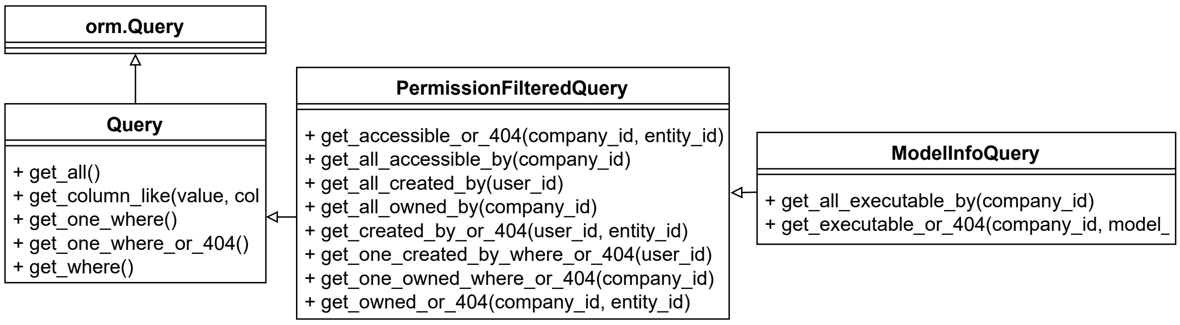

orm.Query
Query
+ get_all()
+ get_column_like(value, col
+ get_one_where()
+ get_one_where_or_404()
+ get_where()
PermissionFilteredQuery
+ get_accessible_or_404(company_id, entity_id)
+ get_all_accessible_by(company_id)
+ get_all_created_by(user_id)
+ get_all_owned_by(company_id)
+ get_created_by_or_404(user_id, entity_id)
+ get_one_created_by_where_or_404(user_id)
+ get_one_owned_where_or_404(company_id)
+ get_owned_or_404(company_id, entity_id)
ModelInfoQuery
+ get_all_executable_by(company_id)
+ get_executable_or_404(company_id, model_

Figure 6.4: Class diagram of the query classes showing their inheritance relation

In SQLAlchemy, each table in the database is represented as a model class in the code. Each of these classes must extend from a base model class known as declarative base class. The Figures 6.5 and 6.6 show the most important model classes of this project. As seen in the figures, all model classes extent from the abstract class *BaseModel*. This class extends from the SQLAlchemy declarative base and provides some standard functionalities that can be reused by its children classes.

Figure 6.5 shows the model classes related to the food product definition, which is explained in Section 2.1.1. As usual, all classes shown here extends from the *BaseModel*. The *FoodProduct* class is composed of other model classes and contains all necessary properties for a food product definition.

Information about the computational models, which is explained in Section 2.1.2, are stored both in PostgreSQL and GraphDB. Simple data such as the model's name, description, URL of the corresponding gateway are stored in the PostgreSQL. The model input and output information is stored in the GraphDB along with their corresponding units of measurements. The ontology used for the model input and output is explained in Section 6.2.2.

The *ModelInfo* class shown in Figure 6.6, is the part of the computation model information that is stored in PostgreSQL. The same figure also shows *ModelInfo* and *FoodProduct* is connected with the *Simulation* class. Simulation is explained in Section 2.1.3 as a paring between a food product and

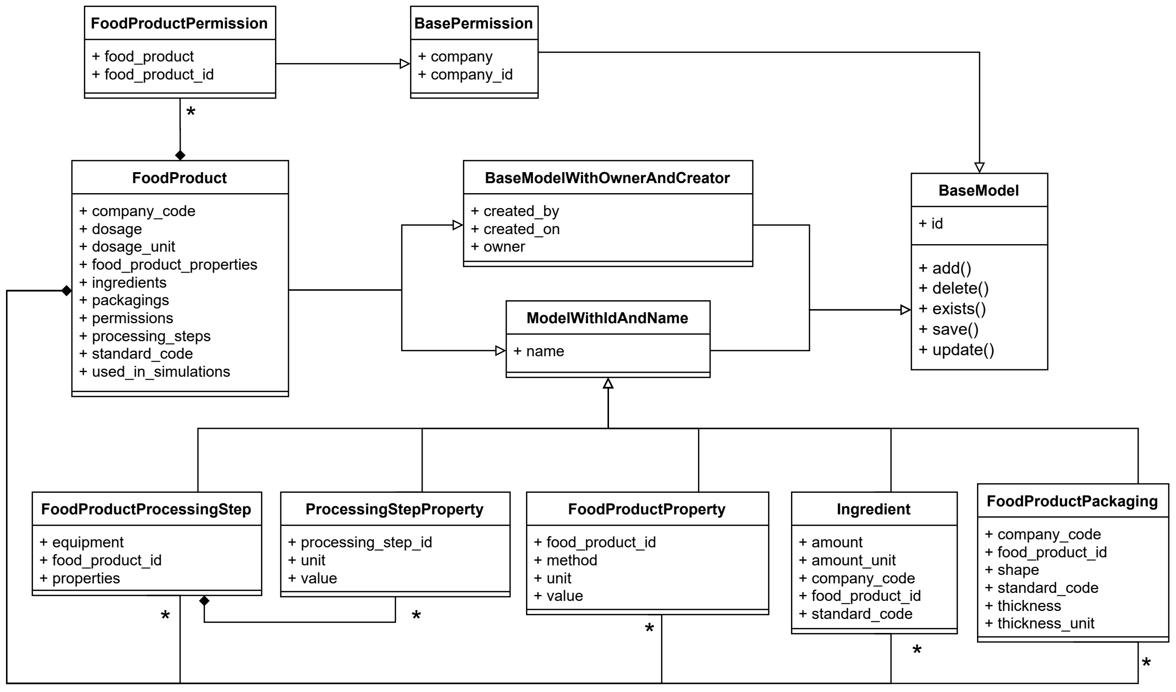


Figure 6.5: Class diagram showing the food product definition related classes

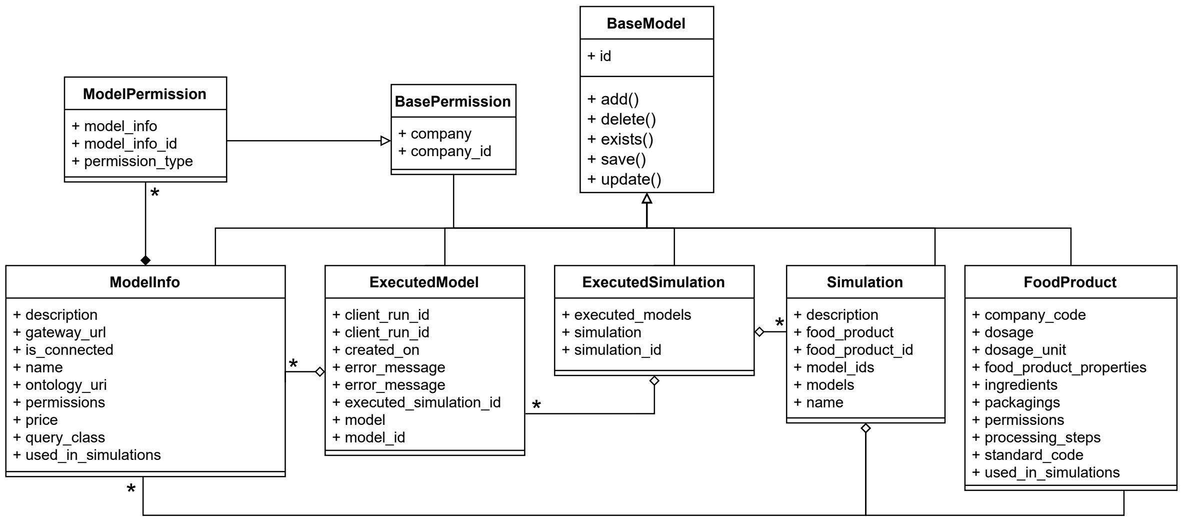


Figure 6.6: Class diagram showing the relation among model information, food product definition and simulation related classes

one or more computational model. Similar relation is also shown in the diagram. It also depicts the *ExecutedSimulation* and *ExecutedModel* classes that store the execution information for each run of the corresponding simulation. Here a point to note is the *ExecutedModel* stores error messages in case of a failed model run but it does not store the results from the execution. They are stored by the gateway that provides endpoints for retrieving model run results. This is a design choice to keep the backend relatively simple without loosing any functionality and avoid large amount of data on the backend side.

A close inspection of the *ModelInfo* class reveals that it does not contain any information about the input and output parameters of a computational model. This is because they are annotated using different ontology and stored separately in GraphDB. The *ontology_uri* of the *ModelInfo* class holds the reference to the corresponding model ontology. While retrieving models, queries are run on both PostgreSQL and GraphDB. The results of those queries are combined before they are populated. The model input-output parameters are further discussed in Section 6.2.2.

Permissions: One of the key aspects of the MaaS infrastructure is to allow organizations to use models from other partners and permission plays a large role in this use case. Figure 6.7 shows the class diagram for the implementation of permissions in this project. As seen in the figure, permissions are granted to companies. There are two permission implementation classes: *ModelPermission* and *FoodProductPermission*. Both of them are associated with a company and store permissions for *ModelInfo* (Figure 6.6) as well as *FoodProduct* (Figure 6.5) respectively. The *ModelPermission* class has the *permission_type* property that maintains two level of permissions: view and execute. The view permission allows the corresponding company view only access. Companies with execute permission can view and run a model.

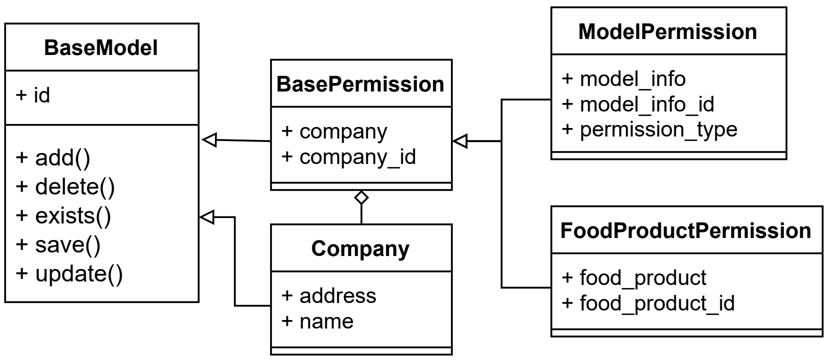


Figure 6.7: Class diagram showing hierarchy of the permission classes

User Authentication and Authorization: This project implements a token based authentication. The backend generates a unique token for a correct pair of username and password that can be provided from the frontend. This token is used in subsequent requests to verify whether the corresponding user is authenticated or not.

Along with other information, the username and password are stored with the *User* class as shown in Figure 6.8. Each user is associated with one *Company*. Each user is granted the same permissions as the company he/she belongs to.

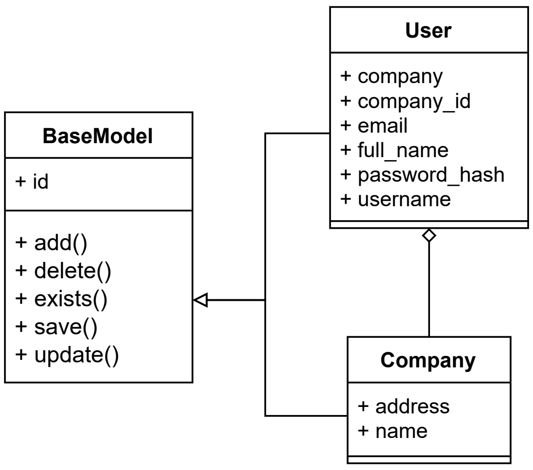


Figure 6.8: Class diagram for user related classes

Gateway and Computational Models

The heterogeneity of models is identified as one of the key challenges in Section 5.2.1. Models can be very different in the way they are developed, deployed and run. The purpose of the gateway is to hide this heterogeneity and provide a standard interface. According to the architecture defined in Section 5.3.1, a gateway exposes the services of a computational model to the backend through a standardized web interface called *ISimulate*. The implementation of this interface differs based on the model it represents. As shown in Figure 6.9, this interface is relatively simple and provides three endpoints:

- POST $/api/run\_model$: accepts food product properties and initiates a run of the corresponding model. It returns an unique id for each run.
- GET $/api/get\_result$: returns the results based on the provided id of the run if the corresponding model has finished executing.
- GET $/api/get\_model\_run\_state$: returns the status of the model run. There are four possible statuses: submitted, running, completed, failed.

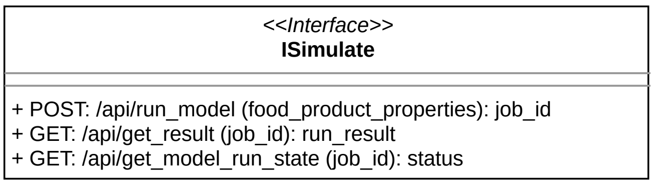

<<Interface>>
ISimulate
+ POST: /api/run_model (food_product_properties): job_id
+ GET: /api/get_result (job_id): run_result
+ GET: /api/get_model_run_state (job_id): status

Figure 6.9: *ISimulate* Interface

In this project, the example model provided by Unilever, which is explained in Section 2.3, was used to implement the gateway and its corresponding computational model. It calculates the taste perception (i.e. sweetness, sourness, saltiness, tomato flavor) of tomato soup based on the ingredients.

Since same libraries are used, the web service implementation of the gateway is very similar to the backend, which is described earlier in this section. The gateways use the Flask library to implement the endpoints specified in the interface *ISimulate*. The configurations used in the web services of the gateway are shown in Figure 6.10.

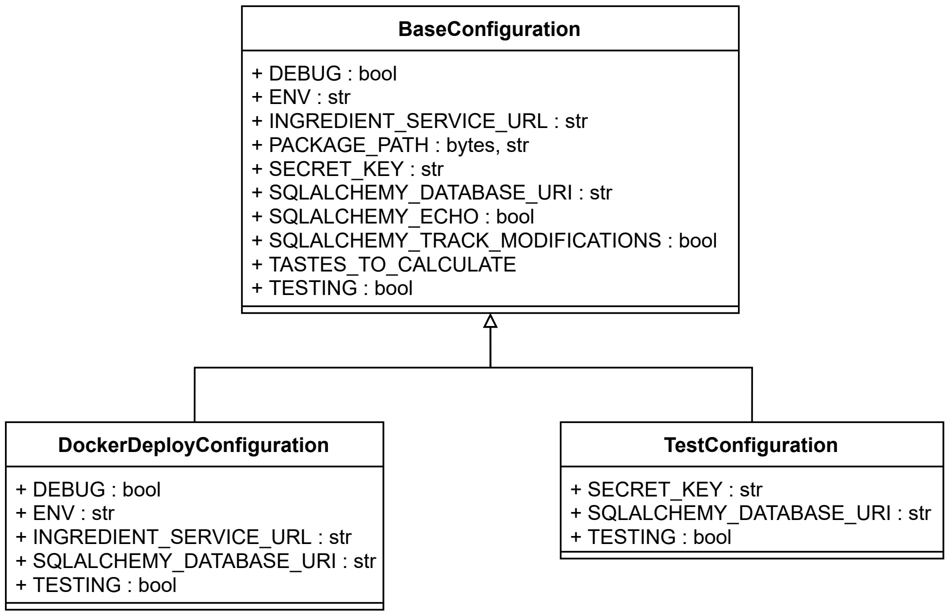

BaseConfiguration
+ DEBUG : bool
+ ENV : str
+ INGREDIENT_SERVICE_URL : str
+ PACKAGE_PATH : bytes, str
+ SECRET_KEY : str
+ SQLALCHEMY_DATABASE_URI : str
+ SQLALCHEMY_ECHO : bool
+ SQLALCHEMY_TRACK_MODIFICATIONS : bool
+ TASTES_TO_CALCULATE
+ TESTING : bool
DockerDeployConfiguration
+ DEBUG : bool
+ ENV : str
+ INGREDIENT_SERVICE_URL : str
+ SQLALCHEMY_DATABASE_URI : str
+ TESTING : bool
TestConfiguration
+ SECRET_KEY : str
+ SQLALCHEMY_DATABASE_URI : str
+ TESTING : bool

Figure 6.10: Web service configurations for the gateway

As mentioned earlier in this section, the model run results are stored in the gateway. Similar to the backend, for storage the PostgreSQL relational database is used in association with SQLAlchemy ORM library. For this purpose, the *SimulationRun* ORM class is used that is shown in Figure 6.11. Upon receiving a model run request, the gateway generates a unique identifier that is stored in the database along with received parameters, date and sender of the request. At this stage, the status of the run request is submitted. As soon as the model execution starts, the status is changed to running. Finally, the model run can finish with either success or failed status. The transitions among the execution states are shown in Figure 6.12.

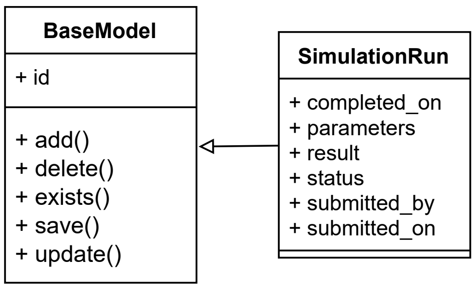


Figure 6.11: Data classes for storing model run results

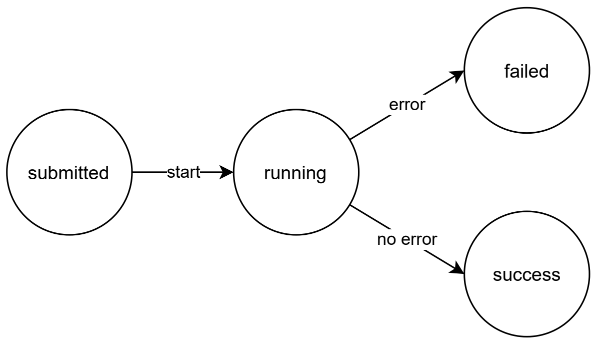


Figure 6.12: Transition among run statuses of a model

MaaS Frontend

The frontend of the MaaS infrastructure is a web application that consumes the services provided by the endpoints of the MaaS Backend. The web application is implemented using Angular[4], which is a Typescript based open source frontend framework. The UI design of the frontend follows the material design guidelines [21].

The frontend consists of several pages each of which are linked to one or more other pages. The graph in Figure 6.13 shows all the major pages and how they are connected to each other. Appendix D presents screenshots of the major pages of the frontend.

The frontend was implemented by a group of bachelor students under the supervision of the author as part of the Software/Web Engineering Project (SEP), which is introduced in Section 8.4. Further implementation details about the unit translation functionality is available in [1].

6.2.2 Ontology in MaaS Infrastructure

In the context of this project, interoperability has been identified as one of the key challenges, which is discussed in Section 5.2.1. The Section 5.3.2 explains the decision to use ontology to be able address the challenge. Therefore, an ontology for representing the input and output parameters of a computational model has been developed. The development activities was performed by representatives from WUR (see Section 3.1.3) in collaboration with the author of this report.

Figure 6.14 shows the structure of essential parts of the developed computational model ontology. Each has several inputs and outputs that are annotated with the *Quantity* class. This class can have several types such as *Density*, *Volume*, *MassFraction*. The *Quantity* is also associated with a *Unit*. The Ontology of Units of Measure (OM2.0) [2] has been used to annotate the *Unit* and *Quantity* related classes. As shown in Figure 6.15, the OM2.0 ontology provides classes, instances, and properties representing different concepts used for defining and using units. It includes, for instance, common units such as the SI units meter (*om:metre*) and kilogram (*om:kilogram*), but also units from other systems of units such as the mile (*om:mile*) or nautical mile (*om:nauticalMile-International*) [2].

As mentioned in Section 2.3, this project uses an example computational model provided by Unilever that can calculate the sensory taste perception of tomato soup based on the amount of its ingredients. Figure 6.16, shows the ontology of the tomato soup model that follows the structure shown in

[4]https://angular.io/

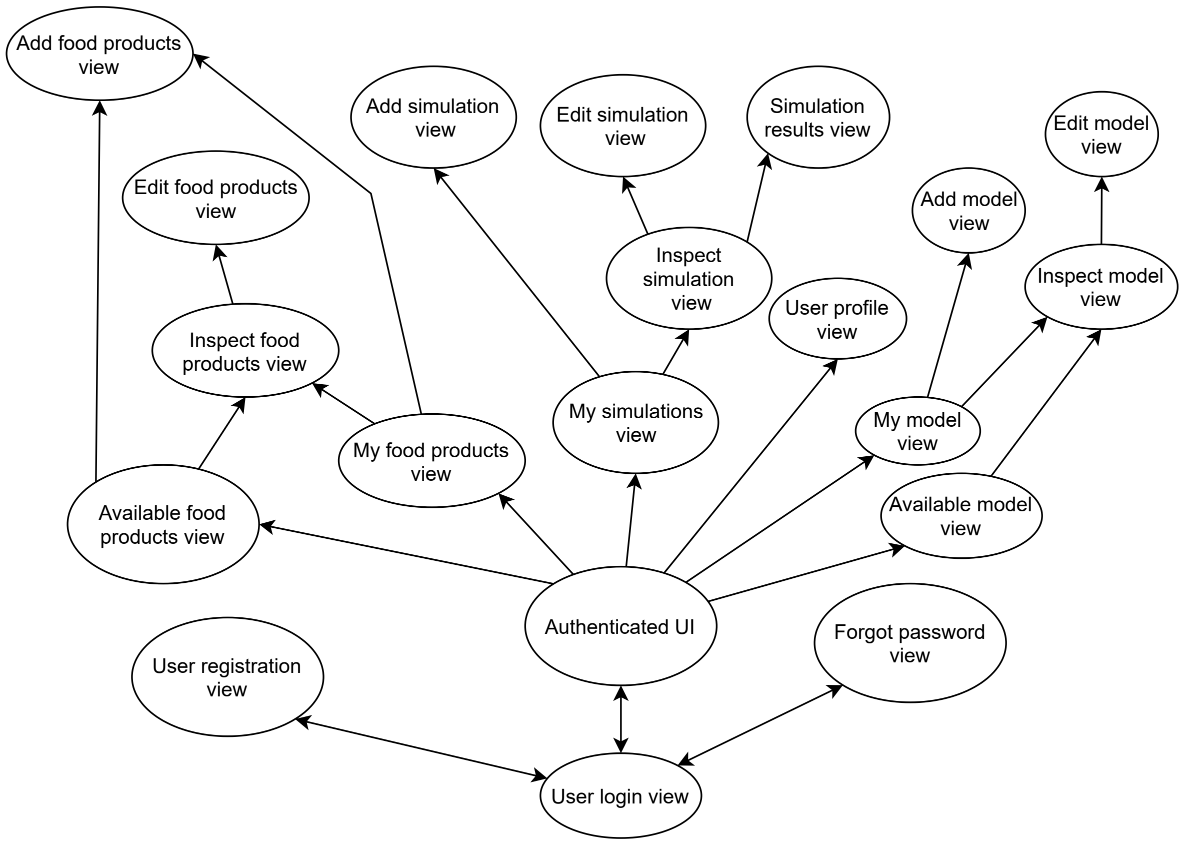


Figure 6.13: Pages in the frontend and how they are connected (adapted from [1])

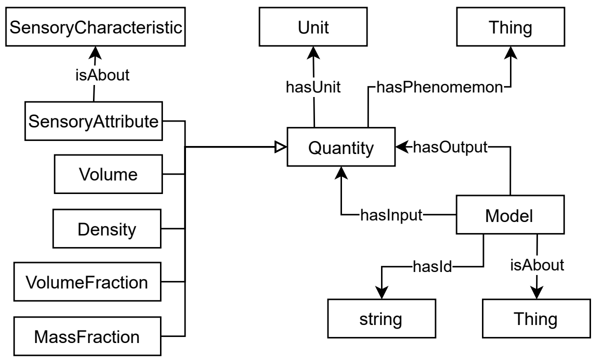


Figure 6.14: Computational model ontology structure

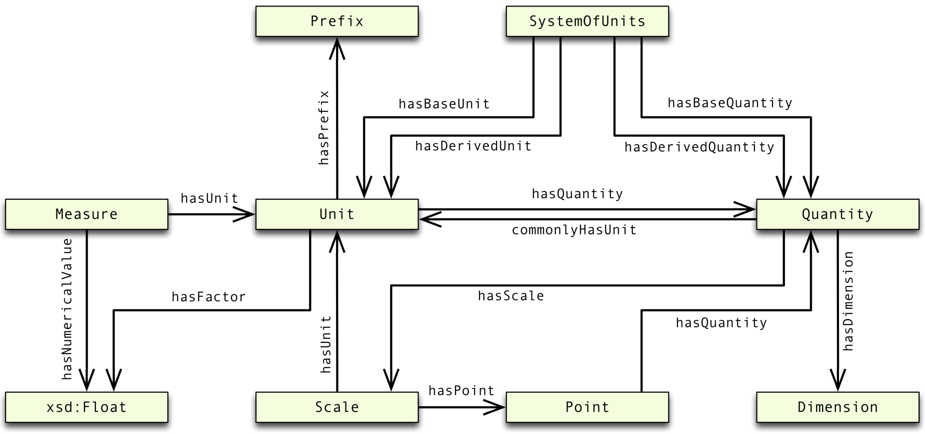


Figure 6.15: The class structure of the OM2.0 ontology [2]

Figure 6.14. The right-hand side of the figure shows the input parameters and their corresponding properties including type of the quantities and their units. The left-hand side depicts the outputs of the computational model that are of type *sensoryAttribute*.

Ontology and the Normalization Component

The component diagram of the MaaS infrastructure shown in Figure 5.4 introduces the *Normalization* sub-component as part of the *MaaS Backend*. The primary goals of the *Normalization* sub-component are as follows:

- Map properties defined in food product definition to model input parameters.
- Convert between comparable units of measurements (e.g. gram to kilogram).

Because of this sub-component a user can define the food product with the unit of measurements that are suitable for him/her and use it in a simulation without worrying about manually mapping the food product properties or converting to the units expected by the corresponding models.

The mapping between the food product properties and the model input parameters are done based on their name. An example mapping is shown in the Figure 6.17. The tomato soup model ontology

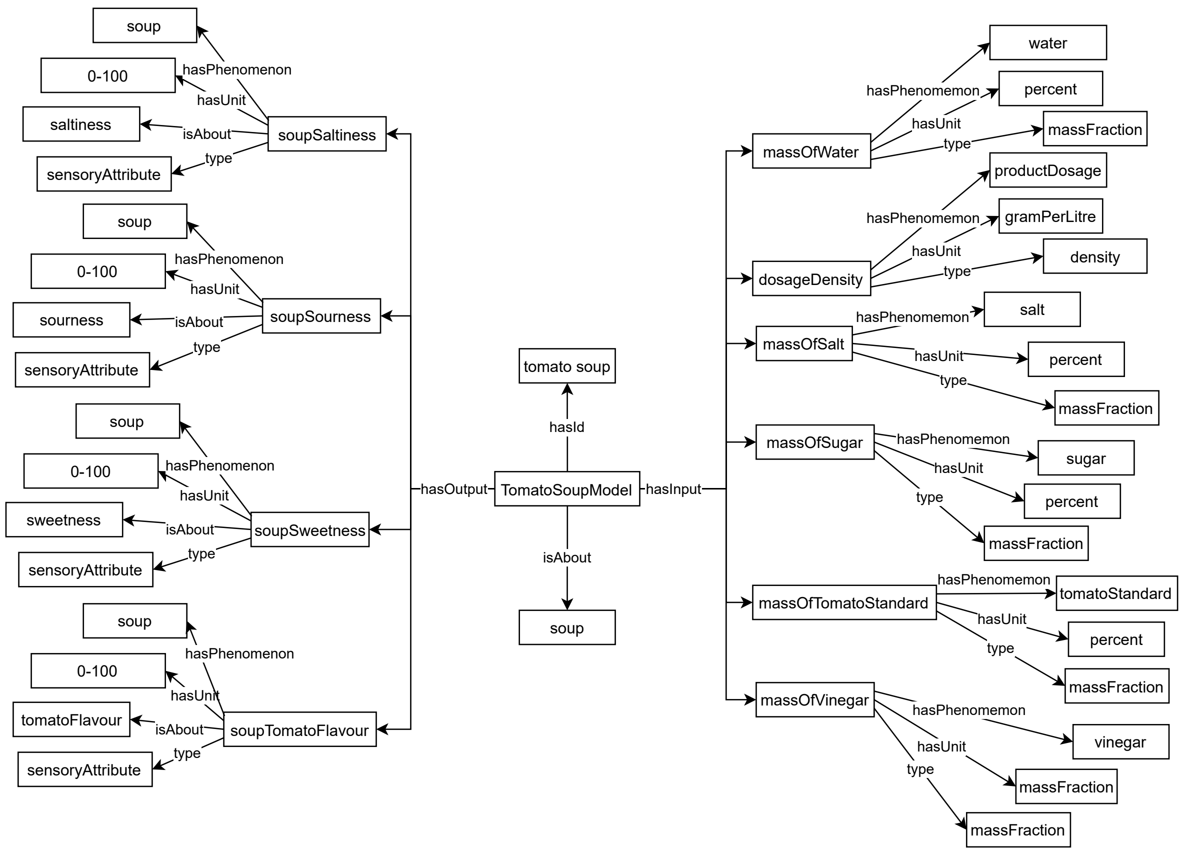


Figure 6.16: Ontology of tomato soup taste model input and output parameters

defines the name of the parameters in several languages (i.e. English, Dutch) that is considered during the mapping process. As a result, the *suiker* ingredient of the food product can be mapped with the *sugar* input parameter of the computational model.

Figure 6.18 shows how the *Normalization* component is used by the *MaaS Backend* for converting a value from a certain unit to another. The *Normalization* component uses the OM2.0 for implementing the unit conversions. Almost all the units defined in OM2.0 are connected to a factor value with a *hasFactor* edge (Figure 6.15). This factor can be used to convert a unit to its corresponding SI (International Unit System [22]) equivalent. For example, foot (*om:foot-International*) has a factor of 0.3048 that means multiplying a foot value with this factor will convert it to meter which is the SI unit for length. During a conversion, the *Normalization* component converts the source value to its SI equivalent and then to the target unit using the factors defined in the OM2.0 ontology.

The unit conversion functionality of the *Normalization* component was developed by a group of bachelor students under the supervision of the author as part of the Software/Web Engineering Project (SEP), which is introduced in Section 8.4. Further implementation details about the unit translation functionality is available in [1].

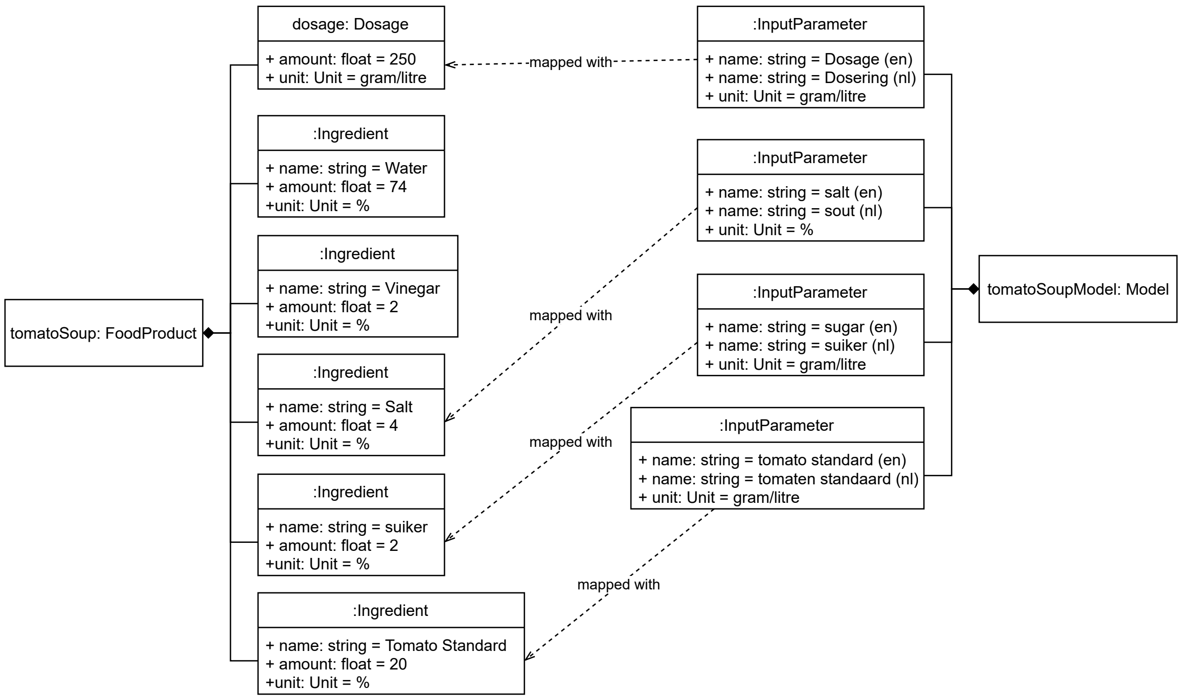


Figure 6.17: Example of mapping properties of food product definition to the model input parameters

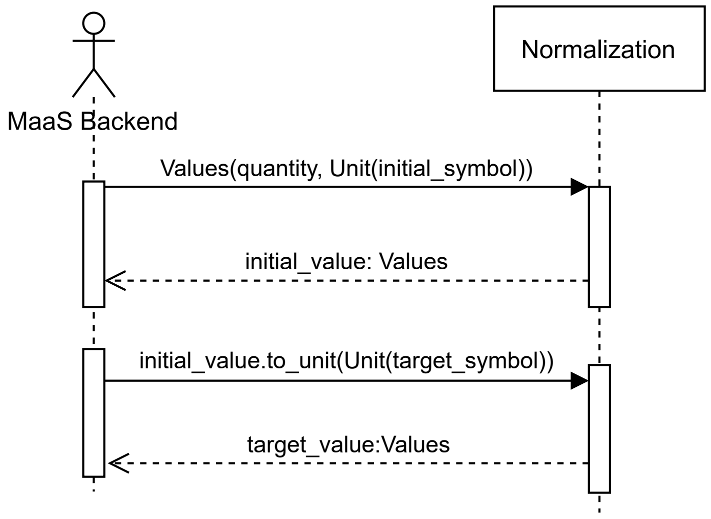

MaaS Backend
Normalization
Values(quantity, Unit(initial_symbol))
initial_value: Values
initial_value.to_unit(Unit(target_symbol))
target_value:Values

Figure 6.18: Sequence diagram showing the usage of normalization component (adapted from [1])

Querying Ontology and Related Data Structures

The ontology depicted in Figure 6.16 is stored in the GraphDB that uses *SPARQL* for querying data. The GraphDb provides the following REST endpoints to read and write data:

- GET $/repositories/\{repositoryID\}/\{query\}$: Execute specified *query* on the repository specified with *repositoryID*. This endpoint can execute a read-only query and return the results as JSON.

- POST $/repositories/\{repositoryID\}/statements/\{update\}$: Performs operations specified with *update* parameter on the data in the repository specified with *repositoryID*. This endpoint is used for updating the ontology stored in GraphDB. In this case, update means modifying the existing triples as well as adding new ones.

The *SparQlRunner* class shown in Figure 6.19 is a wrapper around the GraphDB endpoints. It provides *read* and *write* methods that are responsible for querying the ontology and modifying it. The *read* method returns a list of *QueryResult* that is an object representation of the JSON returned by the GraphDB query endpoint. Figure 6.20 shows the structure of the *QueryResult* class.

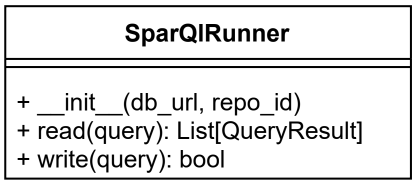


Figure 6.19: *SparQlRunner* class that communicates with GraphDB

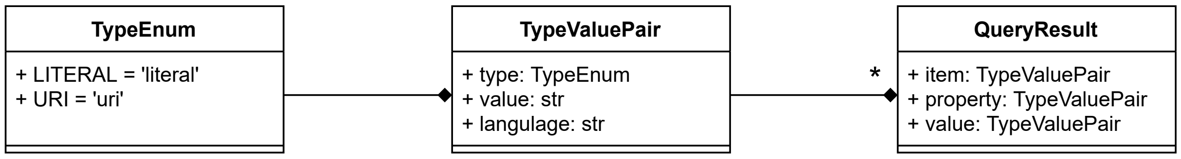


Figure 6.20: Query result classes returned by *SparQlRunner*

The *QueryResult* is a set of *TypeValuePair*. The type in the *TypeValuePair* can be a *literal* or an *uri*. A *literal* type means the corresponding *value* contains actual data. On the other hand, an *uri* typed value holds a reference to another graph node.

The *QueryResult* return by *SparQlRunner* is processed further and converted to *GraphDbModel*, which is shown in Figure 6.21. The *GraphDbModel* is a class representation of the computational model ontology shown in Figure 6.14. Each *GraphDbModel* has an unique *uri* and a set of input as well as output parameters. Each *GraphDbModelParameter* can have several *Labels* and a *Unit*. The *Unit* class holds references to OM2.0 ontology that is essential for unit conversion and maintaining interoperability.

6.3 Deployment and Running the Project

Docker[5] has been intensively used to deploy the MaaS infrastructure. Docker is a set of tools that use OS level virtualization to package software into containers. Each container is isolated from one another and contains all necessary files, libraries, as well as configurations to run the corresponding software.

As depicted in the deployment diagram in Figure 6.22, each of the microservices and database instances are deployed in separate Docker containers. The directed associations show the dependencies

[5]https://www.docker.com/

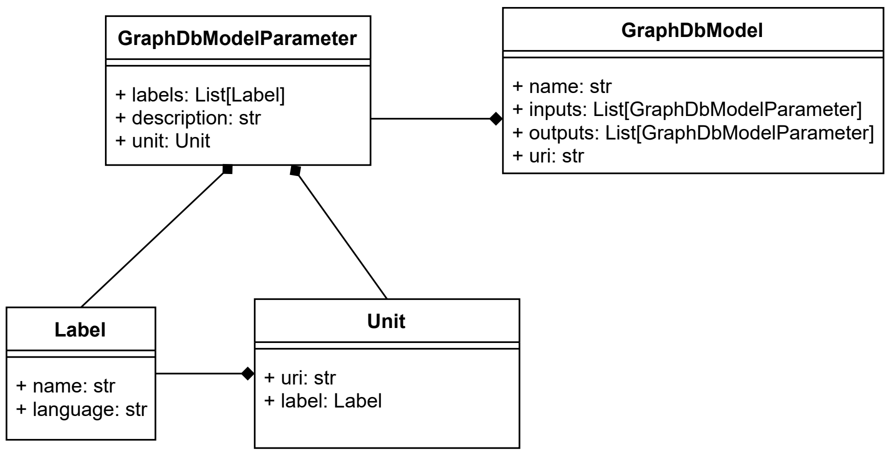


Figure 6.21: Class representation of a computational model ontology

among them. The containers communicate with each other using a private virtual network. As the diagram shows, all containers, including the gateway, are deployed on one docker host machine. According to the architecture explained in Section 5.3.1, gateways represent computational model from another entity (e.g. company) and should ideally run on different infrastructure. However, since the example tomato soup taste model (see Section 2.3) does not require any special environment (software/hardware), the docker container of the corresponding gateway is sufficient to deploy the model.

To define and run all the Docker containers shown in Figure 6.22, this project uses Docker Compose[6]. It is a multi-container management tool that provides one point of interaction to create, modify and remove all containers. Appendix A provides further details about deploying the containers and running the project for debugging.

[6]https://docs.docker.com/compose/

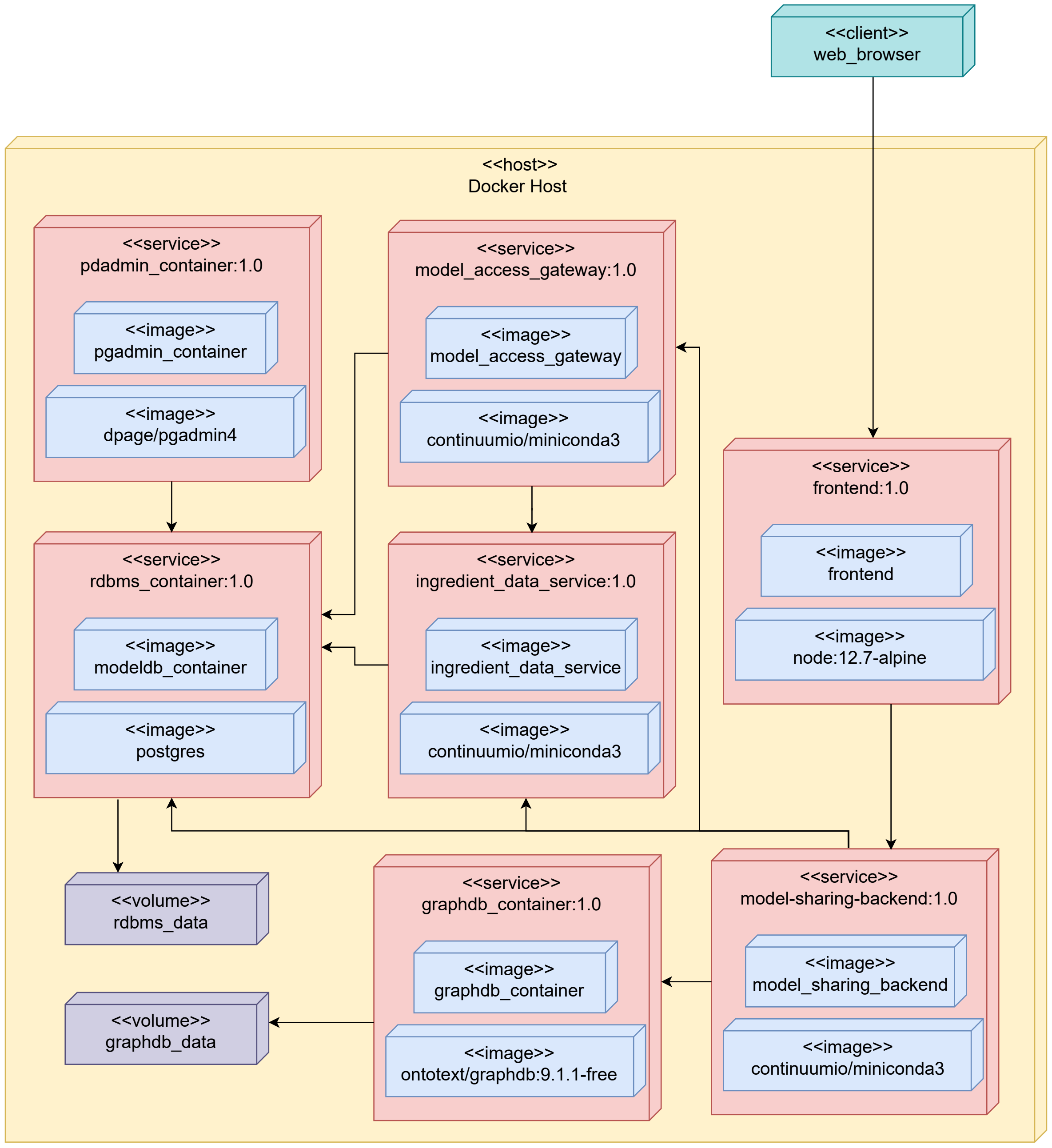


Figure 6.22: Docker deployment diagram of MaaS infrastructure

7 Verification and Validation

Verification and validation refers to separate processes that checks whether a software product fulfills its requirements and specification. The later sections of this chapter describe the verification and validation processes that were used during the development of the MaaS infrastructure.

7.1 Verification

The IEEE-STD-610 standard [23] defines verification as, "The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase." In simpler terms, verification is a continuous process for checking the application that is being build to ensure it adheres to certain specifications (e.g., well-designed, error free).

During the development, the MaaS infrastructure was verified in three ways: unit testing, integration testing, and code consistency checking. Each of these processes are elaborated in the following sections.

7.1.1 Unit Testing

Unit test is a software testing method that checks if the individual units of the corresponding software have expected behavior. These tests are typically performed by the developer by writing additional code that automatically tests the software. In the context of this project, unit tests were used not only to test the newly implemented features, but also to ensure that the existing functionalities were not broken.

For unit testing, units are defined differently for each of the components of the MaaS infrastructure. In case of the MaaS backend, each of the endpoints, which are explained in Section 6.2.1, is defined as a unit. For the frontend, the Angular[1] components and services are considered as units.

For each of the endpoints in the backend, there is at least one unit test that checks the inputs, outputs, and response codes of the corresponding endpoint. These unit tests are listed in Table 7.1. The testing API provided by *Flask* (Python web service framework introduced in Section 6.1.1) and *pytest*[2] was used to write the unit tests. *Pytest* is one of the most popular and feature rich unit test framework for testing Python code. To make sure that the endpoints were reliable and stable, they were tested with invalid/dummy and correct data. The dummy data was generated using a fake data generation tool

[1]https://angular.io/

[2]https://docs.pytest.org/en/stable/

called *faker*[3].

Table 7.1: Unit tests for testing MaaS backend endpoints

Unit Test Name	Endpoint Type	Status
test_user_registration_and_login	Authentication	Passed
test_check_authorization	Authentication	Passed
test_login_wrong_credential	Authentication	Passed
test_update_permissions	Food Product Definition	Passed
test_set_permission_to_food_product	Food Product Definition	Passed
test_create_and_get_accessible_product	Food Product Definition	Passed
test_create_and_get_owned_product	Food Product Definition	Passed
test_create_and_get_one_food_product	Food Product Definition	Passed
test_create_and_update_one_food_product	Food Product Definition	Passed
test_create_and_get_all_food_products	Food Product Definition	Passed
test_create_and_delete_food_products	Food Product Definition	Passed
test_delete_food_product_used_in_simulation	Food Product Definition	Passed
test_get_all_ingredients	Food Product Definition	Passed
test_model_not_accessible_by_other_company	Model	Passed
test_model_create_and_delete	Model	Passed
test_model_not_deletable_by_other_people	Model	Passed
test_model_create_and_update	Model	Passed
test_model_not_update_by_other_user	Model	Passed
test_model_access_to_other_company	Model	Passed
test_update_permissions	Model	Passed
test_delete_model_used_in_simulation	Model	Passed
test_create_and_get_simulations	Simulation	Passed
test_get_accessible_simulations_test	Simulation	Passed
test_simulation_not_accessible_from_other_organization	Simulation	Passed
test_update_simulation	Simulation	Passed
test_delete_simulation	Simulation	Passed
test_no_delete_by_different_user	Simulation	Passed
test_use_unauthorized_model_or_product	Simulation	Passed
test_use_model_with_execute_permission	Simulation	Passed
test_use_model_with_view_permission_throw_error	Simulation	Passed
test_use_not_connected_model_throw_error	Simulation	Passed
test_get_model_by_uri	GraphDB Qeury	Passed
test_get_all_models_from_graph_db	GraphDB Qeury	Passed

The frontend was part of the SEP assignment and was developed by a group of bachelors' students, which is explained in Section 8.4. They developed a unit test plan [24] that describes the tested items, test procedure, and results of the testing.

[3]https://faker.readthedocs.io/en/master/

7.1.2 Integration Testing

Integration tests are performed to test if the individual units are working as expected after integrating them together. This is the next testing step after the unit test. The main goal here is to test the interfaces between the components.

In the context of this project, the frontend application developed by the SEP students acted as the base of the integration test. During the development of the frontend, the students received the specification of the available backend endpoints in the form of a Swagger[4] API specification that included a short description of each endpoint, their input and output parameters, as well as HTTP return codes. The students developed the frontend and integrated it with the backend based on this specification without knowing any details of the backend implementation. The automated unit tests that were implemented in the frontend indirectly calls the backend endpoints. These tests indicated whether the integration between the frontend and the backend was working properly.

Additionally, end-to-end tests were also implemented in the backend. These tests automatically defined a food product, connected a computational model, created a simulation with previously created items, and finally, ran the simulation. Table 7.2 shows the implemented end-to-end tests. The integration between the backend and the database servers as well as the gateways were tested using these tests.

Table 7.2: End-to-end tests for testing minimum functionalities needed for running a simulation

Test Name	Description	Status
test_end_to_end_with_invalid_model_gateway_url	Trying to run a model with unreachable gateway should return error	Passed
test_end_to_end_with_invalid_model_and_food_product	Trying to run a model with invalid data (i.e., food product properties that can not be mapped) should return error	Passed
test_end_to_end_with_valid_tomato_soup	Running a model with proper data should be successful and return results from model	Passed

7.1.3 Code Consistency Checking

As mentioned in Section 6.1.1, the backend was developed with Python. To ensure that the coding style was consistent throughout the whole code repository, the PEP8 style guide [25] for Python was used. This style guide was developed by the creators of the Python language and has been widely accepted. For automatically checking the coding style, PEP8 compliance options were enabled in the PyCharm[5], which was the integrated development environment (IDE) used in this project. This

[4]https://swagger.io/

[5]https://www.jetbrains.com/pycharm/

provided basic guidelines (i.e., tabs vs. spaces for indenting, indent width, line length) for the code and made it easier to read.

7.2 Validation

The IEEE-STD-610 standard [23] defines validation as, "The process of evaluating a system or component at the end of the development process to determine whether it satisfies specified requirements." The purpose of this process is to ensure that the right product has been developed and it meets the expectations of the stakeholder. In the context of this project, the system under development was validated by the key stakeholders (stakeholder analysis in Sec. 3.1) at various stages, which are discussed in details in the following sections.

7.2.1 Regular Stakeholder Feedback

During the execution of this project, weekly meetings were scheduled with the key stakeholders. The purpose of these meetings were to keep the key stakeholders involved in the development process, perform short-term validations and identify inconsistent requirements as early as possible. During these meetings, several architectural diagrams were used to explain to the stakeholders how the MaaS infrastructure would be implemented. Based on these discussions, the stakeholders could identify whether the development activities were progressing towards the right direction or not. The mentioned diagrams are presented and explained in Chapter 5.

7.2.2 Stakeholder Demonstration

According to the project timeline explained in Section 8.2.1, the design and implementation phase of this project contained two milestones. Each of the milestone resulted into a prototype. The final prototype, which was the result for the second milestone, was demonstrated to the broader Internet of Food (INoF) partners. The purpose was to demonstrate the newly developed MaaS infrastructure and showcase its possibilities as well as capabilities, which is one of the key goals of this project as explained in Section 2.3.

The final prototype of the MaaS infrastructure implemented all the requirements described in Chapter 4. An overview of these satisfied requirements is shown in Table 7.3. The prototype was also demonstrated to the INoF partners during the final prototype demonstration. The partners including the key stakeholders (explained in Section 3.1) accepted that the prototype met their expectation and satisfied all the elicited requirements. Moreover, several feedbacks were received during the demonstration that indicated the direction in which the INoF partners would like to further improve the MaaS infrastructure. These feedbacks are listed in Section 9.1.

7.2.3 Project Goal Evaluation

Section 2.2 defines the problem in the form of three questions that this project aimed to solve. A hypothetical standardized infrastructure is introduced in Section 2.3 to answer these questions. This section evaluates if the developed MaaS infrastructure can answer the questions from Section 2.2 and has similar characteristics compared to the hypothetical infrastructure introduced in Section 2.3.

Table 7.3: Statuses of functional requirements after implementation

Req. No.	Priority	Requirement Description	Status
FR01	Must	The system shall provide a gateway with a URI that will allow interaction with the computational model	Satisfied
FR02	Must	The system shall allow the user to connect a model to the MaaS infrastructure by specifying the corresponding gateway URI, input and output parameters of the model	Satisfied
FR03	Must	The system shall allow the user to share or unshare a connected model with other users who are from different organizations	Satisfied
FR04	Must	The system shall show a list of models that was connected by current user	Satisfied
FR05	Must	The system shall show a list of models that are available or has been shared with the current user. This list also shows the cost of running the models and their input-output format information	Satisfied
FR06	Must	The system shall allow the user to define a food product by specifying the recipe, processing, packaging	Satisfied
FR07	Must	The system shall allow the user to define a simulation by specifying one or more models and a food product from the list of available products	Satisfied
FR08	Must	The system shall show the results of successfully executed models and error messages if the model execution fails	Satisfied
FR09	Must	The system shall allow the user to execute a simulation	Satisfied
FR10	Must	The system shall allow the user to view the simulation results	Satisfied
FR11	Must	The system shall annotate the information regarding the input and output parameters of a model using ontology	Satisfied
FR12	Should	The System shall allow the user to login to their account using username and password	Satisfied
FR13	Should	The system shall allow the user to create an account	Satisfied

1. **How can models and/or results from them be shared with other organizations without replicating the original execution environment and sharing related artifacts?** One of the key microservices of the MaaS infrastructure are the gateways. As explained in Section 5.3.1, each gateway represents a computational model and exposes the services of the corresponding model to external parties (the backend in this case) through a set of standardized endpoints. These can be used to upload data to run a model on and get corresponding results. The URL of the gateway is used to connect the computational models to the backend. Finally, these connected models can be shared with other organizations. With this architecture, models can keep running in their original execution environment and it is not required to replicate the environment or share related artifacts (e.g., executables, source codes) to be able to share the models.

2. **How can results from several computational models belonging to different organizations be made available under a unified infrastructure?** According to the microservice architecture explained in Section 5.3.1, gateways connect the computational models to the MaaS infrastructure backend. The backend implements a permission system (explained in Section 6.2.1) that can be used to share a connected model with external parties. Moreover, the MaaS

infrastructure provides the functionality to define simulations that are pairing of a food product definition and one or more computational models (more in Section 2.1.3). Therefore, with the MaaS infrastructure, it is possible to define, run, and get results from a simulation that contains models from different sources.

3. **How can one company use a model from other companies without reformatting data to match the model inputs (e.g., different units of measurement)?** This question refers to the interoperability that was identified as one of the major challenges of this project (more in Section 5.2.1). As explained in Section 5.3.2, achieving complete interoperability is a broad topic and is out of the scope of this project. However, this project aims to prove that it is possible to achieve interoperability to some extent by using appropriate technology. To do that, MaaS used two different ontologies: Ontology of Units of Measure (OM2.0) [2] and model ontology.

   The Ontology of Units of Measure (OM 2.0) was used to convert between units of measurements while executing models (details in Sec. 6.2.2). The model ontology was developed in collaboration with Wageningen University and Research (WUR) based on the tomato soup model, which was provided by Unilever. Using this ontology, the input and output parameters of a model was annotated that was essential for mapping properties of food product definition to the model input parameters. The model ontology was highly influenced by the tomato soup model as it was the only model available during the development. Therefore, the model ontology can only annotate models similar to the tomato soup model. To be specific, the model must follow the structure depicted in Figure 6.14 and explained in Section 6.2.2. Despite this limitation (also explained in Sec. 9.1.1), the MaaS prototype was able to prove that contextual interoperability is indeed achievable.

8 Project Management

The project management activities started from the very beginning of this PDEng project. This chapter describes how it was organized and managed.

8.1 Way of Working

As described in Chapter 1, the Internet of Food (INoF) project was one of several ongoing projects of the Sustainable Food Initiative (SFI). Being part of the INoF, MaaS project had stakeholders from different organizations and with diverse backgrounds. This contributed greatly to the complexity of this project. Therefore, a proper project management became one of the key activities to steer this project in the right direction.

The time duration for the project was ten months. This constraint limited the number of features or user stories that could be implemented. The requirement analysis process explained in Chapter 4, made sure that these features or user stories were elicited as early as possible. However, in case of real life multi-disciplinary projects, like this one, requirements can be rather dynamic in nature and the used project management process should be able to deal with these uncertainties. To be able to do that, a hybrid approach was used. This approach bought some ideas from the Waterfall [26] methodology to be able to deal with the time constraint and, to some extent, a fixed set of deliverable. To deal with the dynamic nature of the project, several concepts from the Agile [26] methodology were also used.

8.2 Waterfall Software Development Model

The fundamental idea of the waterfall software development model was introduced by Dr. Winston W. Royce [27] in the 70's. [3] defines the waterfall model as, "a sequential design process, often used in software development processes, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of Conception, Initiation, Analysis, Design, Construction, Testing, Production/Implementation, and Maintenance."

Figure 8.1 shows how each step in the waterfall model depends on the previous step and the next step is not started until the current step is complete. This kind of workflow creates a timeline where the progress of each step is linearly comparable to the total time spent. A similar timeline was created in the context of the MaaS project, which is explained in Section 8.2.1.

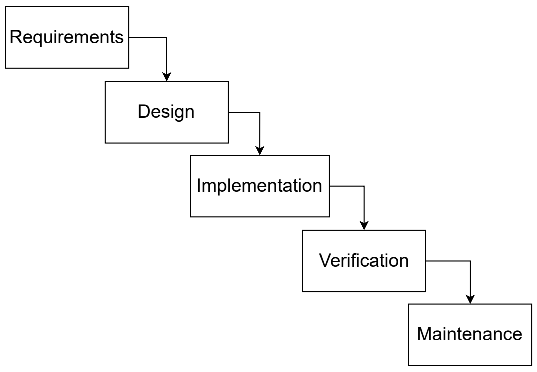
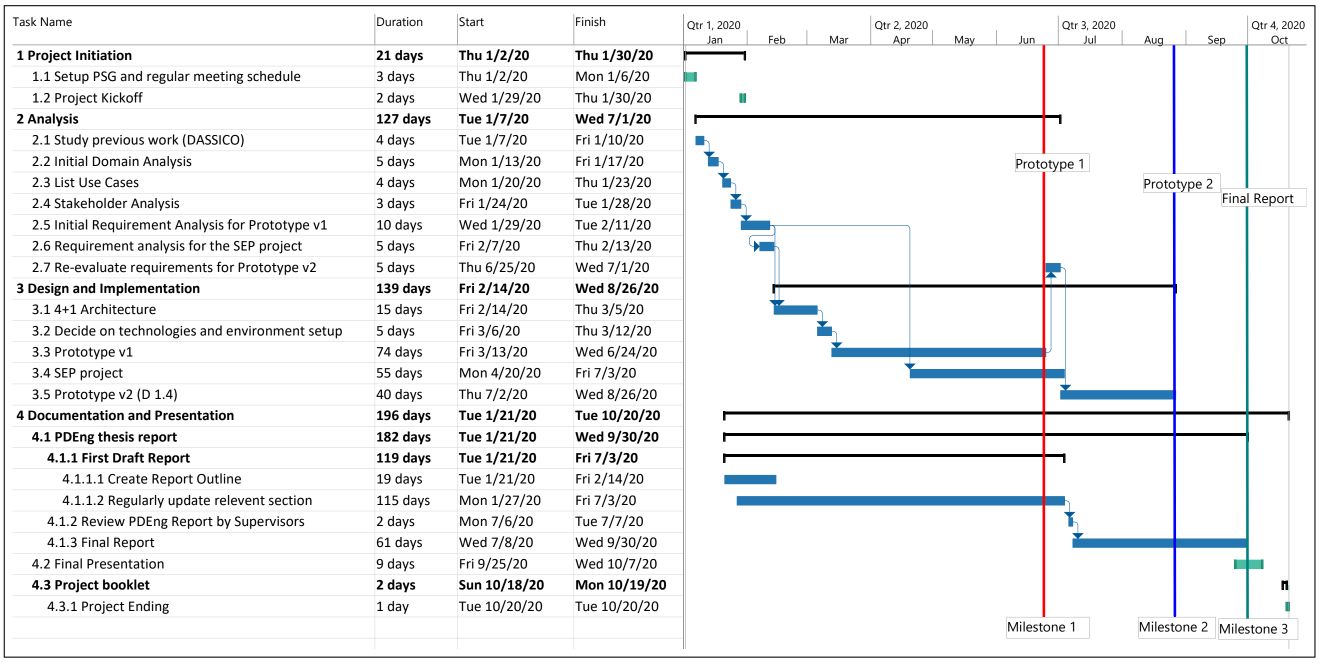


Figure 8.1: Waterfall progress flows from the top to the bottom, like a cascading waterfall [3]

8.2.1 Project Timeline

To keep a high level overview during this project, a project timeline was set up in the beginning and was updated several times to make sure that the activities were in sync with it. This timeline helped not only to evaluate project progress, but also to visualize the impact of one activity on others.

As mentioned earlier, the MaaS project belonged to the food production domain and had multiple stakeholders. Therefore, understanding the context and the domain was absolutely essential to be able to produce an effective product. The project timeline made sure that the domain analysis and requirement elicitation phases were scheduled before any implementation related activity.



Task Name	Duration	Start	Finish
1 Project Initiation	**21 days**	**Thu 1/2/20**	**Thu 1/30/20**
1.1 Setup PSG and regular meeting schedule	3 days	Thu 1/2/20	Mon 1/6/20
1.2 Project Kickoff	2 days	Wed 1/29/20	Thu 1/30/20
2 Analysis	**127 days**	**Tue 1/7/20**	**Wed 7/1/20**
2.1 Study previous work (DASSICO)	4 days	Tue 1/7/20	Fri 1/10/20
2.2 Initial Domain Analysis	5 days	Mon 1/13/20	Fri 1/17/20
2.3 List Use Cases	4 days	Mon 1/20/20	Thu 1/23/20
2.4 Stakeholder Analysis	3 days	Fri 1/24/20	Tue 1/28/20
2.5 Initial Requirement Analysis for Prototype v1	10 days	Wed 1/29/20	Tue 2/11/20
2.6 Requirement analysis for the SEP project	5 days	Fri 2/7/20	Thu 2/13/20
2.7 Re-evaluate requirements for Prototype v2	5 days	Thu 6/25/20	Wed 7/1/20
3 Design and Implementation	**139 days**	**Fri 2/14/20**	**Wed 8/26/20**
3.1 4+1 Architecture	15 days	Fri 2/14/20	Thu 3/5/20
3.2 Decide on technologies and environment setup	5 days	Fri 3/6/20	Thu 3/12/20
3.3 Prototype v1	74 days	Fri 3/13/20	Wed 6/24/20
3.4 SEP project	55 days	Mon 4/20/20	Fri 7/3/20
3.5 Prototype v2 (D 1.4)	40 days	Thu 7/2/20	Wed 8/26/20
4 Documentation and Presentation	**196 days**	**Tue 1/21/20**	**Tue 10/20/20**
4.1 PDEng thesis report	**182 days**	**Tue 1/21/20**	**Wed 9/30/20**
4.1.1 First Draft Report	**119 days**	**Tue 1/21/20**	**Fri 7/3/20**
4.1.1.1 Create Report Outline	19 days	Tue 1/21/20	Fri 2/14/20
4.1.1.2 Regularly update relevent section	115 days	Mon 1/27/20	Fri 7/3/20
4.1.2 Review PDEng Report by Supervisors	2 days	Mon 7/6/20	Tue 7/7/20
4.1.3 Final Report	61 days	Wed 7/8/20	Wed 9/30/20
4.2 Final Presentation	9 days	Fri 9/25/20	Wed 10/7/20
4.3 Project booklet	**2 days**	**Sun 10/18/20**	**Mon 10/19/20**
4.3.1 Project Ending	1 day	Tue 10/20/20	Tue 10/20/20

Figure 8.2: MaaS project timeline

The final timeline is shown in the Figure 8.2. It is divided into four primary activities that are Project Initiation, Analysis, Design and Implementation, Documentation and Presentation. The purpose of the project initiation phase was to define how the project would be organized in the upcoming ten months. This included setting up an initial project timeline, scheduling essential meetings with the supervisors and the stakeholders, as well as defining a communication protocol.

The analysis phase was used to understand the domain and to define the problem that the MaaS project was trying to solve. The domain and problem analysis led to the set of initial use cases. Further analysis of these use cases helped in defining the initial set of requirements.

The design and implementation started after the analysis phase was finished and enough knowledge was gathered about the project as well as the domain. The implementation part was divided into two sub-phases that are labeled Milestone 1 and Milestone 2 in Figure 8.2. Each of these sub-phases resulted into a prototype and the second prototype was the final product of this project. The SEP (Software End Project) was part of this phase where a team of Computer Science Bachelor students from the TU/e participated in developing certain components of the first prototype (details in Sec. 8.4).

The final phase was the documentation and the presentation phase. Unlike the previously mentioned phases, this one ran in parallel with the other phases. The purpose of this phase was to periodically document all the findings and prepare the graduation report as well as the PDEng defense presentation.

8.3 Agile Methodology

To deal with the dynamic nature of the project, the MaaS development process was highly influenced by the agile manifesto [28]. It is a brief document published in 2001 and is built on four values as well as 12 principles for agile software development. The values form the base of the agile mentality and differentiates itself from other methods. These values are:

1. Individuals and interactions over processes and tools.
2. Working software over comprehensive documentation.
3. Customer collaboration over contract negotiation.
4. Responding to change over following a plan.

One of the principles in the agile manifesto [28] states, "Welcome changing requirements, even late in development." This implies that in reality, requirements have the possibility to change and the development methodology must be able to take those changes into account. The agile manifesto also mentions, "Deliver working software frequently, from a couple of weeks to a couple of months, with a preference for the shorter timescale." This principle refers to the iterative development practice where the full development time duration is divided into smaller chunks and a small subset of the requirements are addressed during each of those chunks. This is a contrast to the traditional bulk delivery method and provides an early opportunity for feedback.

8.3.1 Scrum

Scrum is a lightweight process framework that is a subset of the agile methodology. Scrum defines a set of roles, meetings, artifacts, and fixed length iterations known as sprints. Each sprint can have a duration of one to four weeks. Figure 8.3 depicts the key activities and roles of a Scrum process.

The Scrum process starts with a product backlog. It is a set of desired features that are sorted by priority. A product owner is responsible for maintaining the product backlog and explaining it to the team. He is also responsible for keeping the team motivated with a goal and vision.

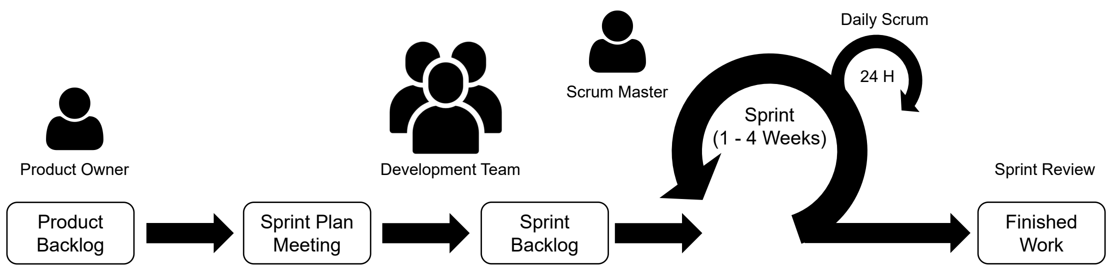


Figure 8.3: The Scrum Process

At the start of each sprint, the product owner explains the top priority backlog items to the team in a sprint planning meeting. The team chooses the tasks that they can complete during the sprint and moves them from the product backlog to the sprint backlog (which is a list of tasks to complete in the sprint).

During the sprint, there is a daily Scrum meeting. It is a 10 to 15 minutes stand-up meeting where each team member talks about their work progress, short term goals, and any issues that have come up. The daily stand-up meetings takes place every day in the morning and helps the team keep an eye on the overall progress as well as maintain transparency.

At the end of each sprint, the team presents their work to the stakeholders in terms of a live demonstration. This meeting is known as the sprint review. Moreover, after each sprint, a retrospective meeting occurs where the team reflects on how well Scrum is working for them and talks about any changes that need to be made in the next sprint.

8.3.2 Agile/Scrum in MaaS

In the context of the MaaS project, the PDEng trainee worked alone. There were, of course, other stakeholders involved. However, they were not directly involved in the core development activities. Although this setup is less than ideal to have a full Scrum, several ideas were adopted from Scrum into the process management of this project.

During the project initiation, a project backlog was set up to be able to record and keep track of all expected features. During the project, this backlog was updated very frequently as new and/or more specific information became available. The backlog included not only the features to be implemented but also the PDEng related activities (e.g., come back day presentation, TSP submission) that had to be completed before specific deadlines.

During this project, the activities were split into one-week sprints. Each sprint started with a review of the backlog. Based on priority and/or urgency, some backlog items were moved to the sprint backlog. The size of the sprint backlog depended on the available time during the sprint.

During the sprint, at least two meetings were scheduled to demonstrate the project progress to the company and academic supervisors. Section 8.6 explains more about these meetings and the communication plans in general.

At the end of each sprint, the sprint backlog was reviewed one more time to find out which items remained incomplete. These items were moved to the next sprint. This provided a nice opportunity to reflect on the previous sprint to find out the things that went well and things that didn't. Finally, if needed, decisions were made to make necessary adjustments to the next sprints to have better results.

8.4 Software/Web Engineering Project (SEP)

At TU/e, bachelor students from the software science major are required to complete a large and complex software development project as part of their degree requirement. These projects are called the Software/Web Engineering Project (SEP). These projects are conducted as a group. The goal is to demonstrate the student's ability to develop large non-trivial software in a group context. A project group consists of 9-11 students. They work almost full-time on the project during a full quarter (including exam weeks) [29].

In the context of the MaaS project, certain parts of the intended system were outsourced as an assignment to a SEP group who called themselves Foo Development. A detailed technical discussion about this assignment can be found in Section 6.2.1. This project was conducted in the fourth quarter of the academic year 2019-2020.

The SEP assignment started on 20 April 2020 with a project kickoff and ended on 3 July 2020 through a demonstration of the results. During this period, the PDEng trainee acted as a client for the Foo Development team. He was responsible for explaining the assignment to the students and making sure that they were producing expected results. To do that, the trainee accepted or rejected requirements that were elicited by the students and provided regular feedback. This was done during the weekly planning meetings where the students demonstrated their progress and the PDEng trainee could assign priorities to the upcoming tasks.

8.5 Risk Management

The risk management process started by identifying an initial set of risks at the beginning of the MaaS project. Each identified risk was assigned a severity and probability score. During the execution of the project, the initial set of risks were updated by adding newly identified risks and adjusting previously identified ones when they became obsolete or mitigated or less relevant.

RISK 01 — Computational models (or example) are not made available in time.
Mitigation strategy:

- Remind related personnel about the models and explain the risks if they are not available in time.
- Prepare and ask questions to understand what the model means in the context of the food product domain.

RISK 02 — The MaaS infrastructure prototype works with the example models but needs significant change to work with real models.
Mitigation strategy:

- Find out how a real model is different from the example model.
- Carefully study the provided model and design the architecture as modular as possible.
- Explicitly mention the risk to the major stakeholders and how this can affect the project given the limited time duration.

RISK 03 — SEP assignment not completed before the first prototype demonstration.

Mitigation strategy:

- Plan a Minimum Viable Product (MVP) that has most of the required functionality.
- During the weekly SEP planning, set task priorities keeping the MVP in mind.
- In case the project lags behind, communicate it to the SEP academic supervisor.
- For the worst case scenario, prepare some buffer time that can be used to take over the SEP project and finish the MVP.

RISK 04 — Due to COVID-19 [30] outbreak, communication with the SEP students was not effective leading to misunderstandings.
Mitigation strategy:

- Set up an effective communication channel where the students can communicate directly with the trainee.
- Get readonly access to the SEP project repository.
- Occasionally, run the project and try to find issues.
- Closely monitor the progress of the SEP project.
- In case the project lags behind, communicate it to the SEP academic supervisor.

RISK 05 — Change in priorities or deliverable mid-project.
Mitigation strategy:

- Schedule regular meetings to demonstrate the project progress to the major stakeholders.
- Encourage feedback during the demonstrations.
- Discuss with the major stakeholders if any contradicting priorities or deliverable emerge.
- Make a list of assumptions and verify them with the major stakeholders.
- Make a list of elicited requirements and verify them with the major stakeholders.

RISK 06 — A major stakeholder not available for long time due to reasons such as resignation, medical issues, change in position.
Mitigation strategy:

- Make the change explicit to other major stakeholders.
- Find out who is assuming the responsibilities in the absence of the stakeholder.
- Schedule a meeting with the new person as soon as possible and explain the project, way of working, progress, and any important points that may have appeared by the time.

RISK 07 — Due to the COVID-19 [30] outbreak, all supervisor meetings are online and these meetings may not be as effective as in person meetings. As a result, issues might pass on undetected.
Mitigation strategy:

- Organize a meeting (online) at least once a week with the supervisors to discuss and demonstrate project progress.
- Use online drawing tools as a replacement of the physical whiteboard to make rough sketches during the online meetings.
- Take extra care about verifying assumptions and communicate immediately in case of confusions.

8.6 Communication Plan

As mentioned in Chapter 3, the MaaS is a multidisciplinary project where the stakeholders have very diverse backgrounds. Therefore, it was important to set up a clear and regular communication channel to avoid any misunderstanding and encourage early feedback. Most of the meetings that took place during the execution of this project can be divided into three different categories: weekly update meetings, monthly update meetings, and other meetings that were organized on demand.

8.6.1 Weekly Update Meetings

These meetings were scheduled at the very beginning of the MaaS project to occur every week on Thursdays until the end of the project. The usual participants were the company supervisors, the TU/e supervisor, and the PDEng trainee. The purpose of these meetings was to demonstrate incremental and small updates to the major stakeholders. These updates included the tasks that had been completed in the previous week, any blocking issues, and plan for the upcoming week. The weekly meetings made sure that the project was running in the right direction and provided the option to spot any misunderstanding as early as possible. These meetings played the role of the sprint review, which is explained in Section 8.3.1.

8.6.2 Project Steering Group Meetings

The Project Steering Group (PSG) is introduced in Chapter 3. This group consisted of major stakeholders of the MaaS project including the PDEng trainee. During the project, the PSG had monthly meetings typically on the last Thursdays of the respective month. The main purpose of these meetings was to encourage feedback from the stakeholders.

Typically, these meetings started with a demonstration where the trainee explained the major features that were implemented since the previous PSG meeting. This was followed by a high level discussion about the design and the architecture of the system, requirements as well as risks identified by the PDEng trainee. A high level plan for the coming month was also discussed towards the end of these meetings.

8.6.3 On-Demand Meetings

Besides the regular weekly and monthly meetings, several other meetings took place during the execution of the MaaS project. Although these meetings were not regular and mostly scheduled on demand, they played key role in communicating with the key stakeholders and understanding the project context.

8.6.4 Communication Medium

During the project kickoff, an agreement was made among the trainee and the major stakeholders regarding the way of communication. According to the agreement, a face-to-face in person communication was set as the preferred way for discussions. However, as the trainee and the stakeholders were not located in the same location, an in person meeting was not always possible. In such cases, Microsoft Teams was chosen as the remote video conferencing tool via which anyone could participate in meetings remotely.

Email was set as the preferred way for exchanging digital media (e.g., documents, images) and sending meeting invitations. Moreover, direct phone calls as well as various social media (e.g., Skype, WhatsApp) remained as backup tools for any urgent communications.

Due to the COVID-19 pandemic [30] that broke out three months after the MaaS project kickoff, offices were shutdown, everyone was advised to work from home if possible, and all academic activities were moved online. As a result, the initial communication agreement was revisited and a new agreement was made to hold all meetings as well as discussions remotely using Microsoft Teams.

9 Research Possibilities and Future Work

One of the goal of the development of the MaaS prototype was to set up a baseline for further development. This chapter discusses the identified future works and research possibilities.

9.1 Limitations and Future Work

During the evaluation of the MaaS prototype, several limitations were identified. Some of them (listed in Section 2.4) were known and accepted by the stakeholders considering the prototype nature of the project. Others were identified by the PDEng trainee throughout the project timeline. These limitations were explicitly discussed with the stakeholders and decisions were made to give them a lower priority or put them out of the scope of this project due to various restrictions.

In addition to identifying the limitations, they were analyzed to spot possibilities for future enhancements. The identified limitations and improvement possibilities regarding the MaaS prototype are discussed in the following sections.

9.1.1 Diverse Models

During the development of the MaaS prototype, Unilever provided an example model that outputs four sensory taste perceptions based on the ingredients of a tomato soup recipe. This was the only model available during the development. Therefore, the prototype was designed, developed, and tested based on this available model. As a result, the prototype was tested to work with models that are similar to the Unilever provided tomato soup taste model. To be specific, the model must follow the structure depicted in Figure 6.14 and explained in Section 6.2.2. The lack of diverse models limited the versatility of the MaaS infrastructure and was identified as a risk (see Section 8.5). The key stakeholders were made aware of this risk and the corresponding effects. However, based on the discussions with the key stakeholders and considering the prototype nature of this project, it was decided to give a lower priority to this risk.

Models are very heterogeneous in nature as they are typically developed in-house by different organizations. Although the microservice based gateways (more in Sec. 5.3.1) and ontology based normalization processes (details in Sec. 6.2.2) implemented in the MaaS infrastructure provided a level of abstraction as well as achieved interoperability in the context of this project, they need further development and testing to accommodate diverse models. To do that, it is important to get access to more models and set up an active feedback loop with the corresponding domain experts to understand as well as support more models in the MaaS infrastructure.

9.1.2 Unit Conversion

The MaaS prototype implements a normalization component (see Section 5.3.2) that includes a unit conversion sub-component. The development of this sub-component was part of the SEP assignment (see Section 8.4) for the bachelors' students. They used the Ontology of Units of Measure (OM2.0) [2] to implement the unit conversions. During the development they found a few issues with some units of measurements from OM2. Their findings regarding the OM2.0 are given in Appendix C.

To summarize, the convertibility between two units of measurements are checked based on their dimensions (more in Section C.1) through the *hasDimension* property of a unit. For example, a conversion between *inch* and *meter* is possible because they are both units with the dimension of *length*. However, due to the absence or ambiguity regarding the dimension information, certain units of measurements are not supported (see Appendix C).

Although the OM2.0 played a key role in the implementation of the unit translation component and the supported units were sufficient for this project, further collaboration with the developers of the OM2.0 is necessary to better understand the ontology and be able to use it more effectively. The findings listed in Appendix C can be used as a good starting point for such a collaboration.

9.1.3 User Authentication

The MaaS infrastructure implements a JSON Web Token (JWT) based authentication that provides basic security for the endpoints available in MaaS backend. Although this authentication validates whether a user is registered or not, it does not validate the identity of the user (i.e., no security protocol to check whether a user actually belongs to a certain organization). Considering the prototype nature of the MaaS infrastructure, this implementation of authentication was a design choice that was discussed with the key stakeholders.

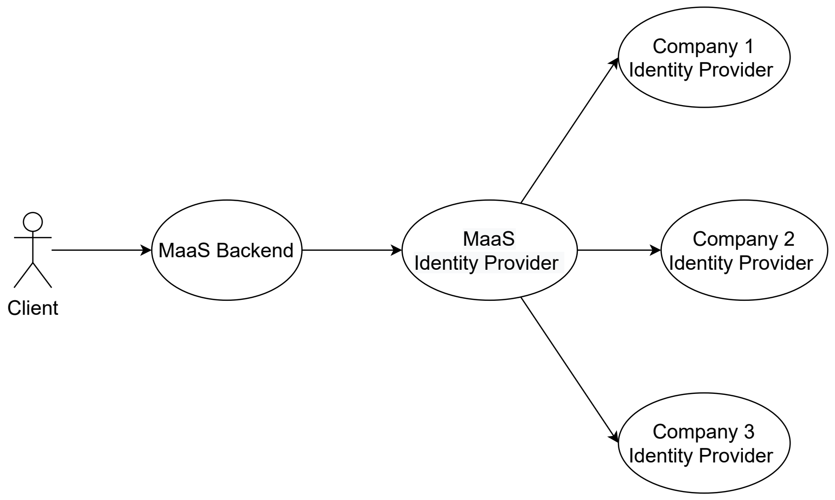

Client
MaaS Backend
MaaS Identity Provider
Company 1 Identity Provider
Company 2 Identity Provider
Company 3 Identity Provider

Figure 9.1: Proposed overview for MaaS identity provider

A distributed identity provider can be used for a more robust authentication system. An example architecture of such a system is shown in Figure 9.1. The idea here is to set up a microservice based micro-infrastructure where each partner organization can validate the identity of a user based on provided credentials.

9.1.4 Usage of Ontology

In addition to the OM2, a separate ontology was developed in collaboration with the Wageningen University and Research (WUR) to annotate the tomato soup taste model provided by Unilever. As mentioned earlier in this chapter, the tomato soup model was the only one that was available during the development of this project. Therefore, the ontology was developed only based on the domain knowledge that could be extracted from this available model. As a result, this ontology is highly coupled with the tomato soup model and can only be used to annotate similar models. To develop a more effective ontology that can annotate larger variety of models, it is essential to have access to more models and corresponding domain experts.

In the implementation of the MaaS infrastructure, the model ontology has been used not only for annotating models but also to store information about input and output parameters of models. Although an ontology is suitable for representing knowledge, it was found to be suboptimal for data storage that requires frequent read, write, and update operations. Following are the identified issues with ontology based storage:

- In the MaaS infrastructure, GraphDB was used to store ontologies. For querying data from this database, SPARQL is used that is not optimized for performance compared to more matured data storage solutions (i.e., relational databases).
- While updating or deleting an entry, not all related triples/relations can be updated or deleted as they may be connected to other concepts. Checking for these additional connections are very costly and greatly affects the quality of service. On the other hand, leaving these unused entities in the storage needs regular maintenance to avoid data piling that might require additional costs.
- In a nutshell, ontology is a set of triples for representing information and connections among them. Therefore, it is extremely flexible and almost no restriction or constraint can be imposed on the database level. As a result, any minor bug or malicious user has the potential to easily corrupt the stored data.

Due to the issues mentioned above, it is recommended to use ontology only for knowledge representation and not for data storage. In [31], several definitions of ontology is given and almost all of them agrees with the idea of knowledge store and/or description of domain concepts. The Ontology of Units of Measure (OM2), which was used in MaaS for implementing unit conversion, is a perfect example of such usage. OM2.0 contains the knowledge of units and relations among them (i.e., conversion factors, dimensions) that are globally accepted. This knowledge base can be used in different ways (in this case is unit conversion) without making frequent changes to the ontology itself.

9.1.5 Cost and Payment

One of the key motivation for an organization to share the services of their models is to create additional source of revenue. In the MaaS infrastructure, it was possible to specify a cost for running a model. However, the payment was not enforced when executing a model and there was no connection with any payment gateway.

Moreover, during the demonstration of the MaaS infrastructure, one of the INoF partners suggested that the cost of running a model should not be a fixed amount. Rather, the user should be able to define costs separately per organization or department within that organization, as it is possible for a company to have different contracts with other entities.

9.1.6 Processing Standard

The definition of food product and related concepts are explained in Section 2.1.1. One of these concepts is processing that is defined as sets of mechanical and/or chemical operations performed to the ingredients of a certain food product during production. In the same section, it is also explained that ISA-88 [10] is a global standard for defining processing steps and MaaS implements a subset of it that was discussed and agreed with the key stakeholders. However, to develop an enterprise grade MaaS infrastructure it is recommended to thoroughly implement the ISA-88 standard.

9.1.7 Performance

The MaaS prototype is not optimized for performance. This is specially noticeable when querying ontologies from GraphDB. As explained in Section 2.3, the primary goal of this prototype was to demonstrate its possibilities to the Internet of Food (INoF) stakeholders. Therefore, after discussing with the client, it was decided to spend limited resources on performance optimizations.

9.2 Research Possibilities

During the execution of this project several research possibilities were identified. Some of these possibilities are more related to enhancement and further development of MaaS that are listed as future work in Section 9.1. The rest are discussed in the following sections.

9.2.1 Model Parameter Mapping

Before running a model, a mapping step is performed between the model input parameters and the properties of food product definition. This step is explained in details in Section 6.2.2. Currently, the mapping is done based on the names, which can be in different languages (i.e., English, Dutch), of the model parameters and food product properties.

This name based mapping puts lots of responsibility to the users of the MaaS as they have to take extra care to conform to the standard set my the model owner (i.e., spelling of the names of ingredients). Therefore, the current mapping technique can be greatly enhanced using context aware artificial intelligence algorithms to compensate for human errors.

Derived mapping is another possible research topic, which was suggested by one of the stakeholders. In this context, derived mapping refers to calculating the value of a model parameter based on the available data in the food product definition. For example, given a food product definition that contains volume and mass information, it is possible to calculate the density using the equation $density = \frac{maas}{volume}$ and use the result as an input to a model if it required so. Implementing it can greatly increase the interoperability of the MaaS infrastructure. However, this requires more research and possibly, development of related ontology.

9.2.2 Guidelines for Model Development

The heterogeneity of models and related challenges or problems are mentioned several times in this report. The models are heterogeneous in nature mainly because they are developed by different companies in isolation and without following any guideline or commonly accepted standard. As a result, they are not ideal for sharing as a service and additional steps are necessary to do so, which might be very complicated considering the legacy models that are still in use. In this context, there are quite some possibilities of research for establishing a set of model development guidelines that are easy to follow, work with most of the popular model development platforms, and take the sharing aspects into account. The outcomes of such research requires a large-scale industry-academia collaboration and can have global impact.

9.2.3 Common Ontology for Model

In Section 9.1.4, the lack of a common model ontology is discussed. Developing such an ontology requires extensive research, access to variety of models, and expertise from domain experts.

9.2.4 Standardization of Ingredient Description

As discussed in Section 2.1.1, food product definition includes a list of ingredients that are necessary for producing the corresponding product. Typically, different companies use different suppliers to acquire these ingredients. As a result, similar ingredients obtained from separate sources can have distinct or divergent characteristics. For example, tomatoes obtained from Supplier A can be sweeter than tomatoes from Supplier B. These kinds of dissimilarities can affect the consistency of the food product. To solve this problem, companies measure the relevant properties of each ingredient and assign an identification to it, which is used later to adjust the recipe. Each company has its own conventions for defining these properties and generating corresponding identifications. As a result, interoperability can not be guaranteed for models that depend upon ingredient properties. This provides possibilities for further research to find a standard ingredient vocabulary that can be used to translate between different ingredient definition systems.

10 Conclusion

This chapter concludes this report by summarizing the project results and describing the artifacts that were delivered to the key stakeholders. Moreover, this chapter also contains a personal note from the author of this report that includes self-reflection and major learning points with regard to this PDEng graduation project.

10.1 Results

During this project, a prototype infrastructure was developed where companies could share the services of their computational models and use the ones shared by other companies. The infrastructure was named MaaS, which is an abbreviation of Model as a Service. The goal was to demonstrate the benefits and possibilities of the prototype to the Internet of Food (INoF) consortium partners to attract further collaboration.

MaaS provides a web-based infrastructure for executing models from various sources and viewing their results under a unified infrastructure. Heterogeneous models, which are typically developed by companies for in-house usage, are connected to the unified MaaS infrastructure using gateways. These are microservices that abstracted computational models and exposed their functionalities as a service via a set of well-defined endpoints. Each gateway is connected to exactly one computational model and takes care of all the complexities of running it. The MaaS infrastructure implements a permission system that can be used with connected models allowing companies other than the owner of the model to execute them.

The MaaS infrastructure also offers the functionality to define a food product that includes corresponding ingredients, processing steps, and packaging information. A food product definition and one or more models from various sources are coupled together to define a simulation, which can be executed. Executing a simulation runs the food product definition through all the corresponding models. Once a simulation is complete, the results from all the models are available through MaaS. This capability of viewing model results that are acquired from diverse sources is an important initial step towards data fusion, which is one of the key goals of the INoF consortium.

To deal with the heterogeneity of the computational models, a data normalization component was implemented into the MaaS infrastructure. This component is responsible for mapping properties of the food product definition to the input parameters of the computational model. The conversion of model parameters from the given to the expected units of measurements is also the responsibility of the normalization component. These normalization steps use ontologies that were developed in collaboration with Wageningen University and Research (WUR).

During this project, a web-based frontend was developed that provided a user-friendly UI for defining a food product, connecting models, specifying permissions for connected models, and executing as well as viewing simulation results. The frontend was developed in collaboration with a group of bachelors' students from Eindhoven University of Technology (TU/e).

MaaS was implemented using a microservice based architecture where each of the services (i.e., gateways, frontend, backend, data storage) are separate microservices that communicate with each other using a REST like API. Each of these microservices runs on separate Docker container that makes the MaaS infrastructure highly modular and deployable on various cloud as well as in-house servers.

The major goal of this project was to develop a standardized platform that would allow inter-organization sharing of model results and demonstrate its capabilities as well as possibilities to the INoF partners in order to attract further collaboration. With its microservice based architecture and several normalization processes, the MaaS prototype has achieved that goal. The prototype was also demonstrated to the INoF partners and earned quite a few compliments.

10.2 Delivered Artifacts

After successfully completing this ten month long PDEng graduation project, the following artifacts were delivered to the stakeholders. The artifacts were delivered to the clients digitally via email as a single zip file.

- **MaaS infrastructure source code:** This refers to all the source code and ontologies that was produced by the PDEng trainee, the bachelors' students from TU/e and the representatives from WUR.
- **Cloud deployment:** During this project, TU/e provided temporary access to a cloud infrastructure that was accessible under the domain name `http://internet-of-food.win.tue.nl`. The MaaS prototype was deployed under this URL.
- **Deployment instructions:** Different components of the MaaS infrastructure could be deployed on their own Docker container. The step-by-step instructions for the deployment were included in the source code repository. The same instructions are also available in Appendix A.
- **Video demonstration and presentation slides:** For demonstration purposes, a video was produced and delivered to the customer. This video is a screen recording that shows how a typical user would use the MaaS infrastructure to define food products, connect models, and define, execute as well as view results of a simulation. Throughout this project, several PowerPoint presentations were produced that included animations and illustrations explaining the MaaS prototype. These slides were also included in the deliverable.
- **Project report:** It refers to this PDEng graduation report. This report not only explains the activities and designs involved in the development of the MaaS infrastructure, but also recommendations for improving and extending it.

10.3 Author's Note

The following text is a retrospective note from the author of this document:

This graduation project assignment was very challenging and fulfilling experience for me. During the past ten months, I have gathered various technical and non-technical skills as well as improved the ones that I already had. This project also provided me the perfect opportunity to practice the skills that I gained through various workshops, courses, and training projects during the first year of my PDEng.

Working in the food production domain was a completely new experience for me. Therefore, understanding the domain was the first major challenge I faced. Tips and feedbacks from my academic as well as company supervisors were crucial for me to be able to navigate through this unfamiliar terrain. Besides my supervisors, I had regular meetings with other stakeholders who contributed greatly to my understanding of the domain concepts. To identify misconceptions and knowledge gaps, I used various diagrams and interview techniques to communicate my ideas. This way I was able to gather enough information to move forward with the technical design and implementation.

During this project, I was working with a multinational and multidisciplinary group of people who are leaders of their own domains and working at world-class institutions. Being part of such team was an honor and challenging at the same time. During various team meetings and presentations, I had to find a common way of communicating that should work for everyone. I believe, this experience will help me become a better engineer and team player.

Finally, I would like to say that this project and the PDEng ST program as a whole helped me become a better professional. I believe, the experiences I gathered and skills I learned will greatly help my future career.

Bibliography

[1] A. M. Altawekji, C. I. Bahrin, T. C. Chirvasuta, P. S. Friptu , A. R. Garban , S. S. van Grieken , D. Hegger, J. Martens, P. Petrov, P. Remkes, and A. E. Voicu. Software/Web Engineering Project, Software Design Document, Internet of Food, July 2020.

[2] Hajo Rijgersberg, MLI Wigham, Dieudonné Johan Martin Willems, and Jan Lubertus Top. *OM 2.0*. Number 1596. Wageningen UR-Food & Biobased Research, 2015.

[3] Waterfall Model. Waterfall model. *Luettavissa: http://www. waterfall-model. com/. Luettu*, 3, 2015.

[4] Bernard Gilland. World population and food supply: can food production keep pace with population growth in the next half-century? *Food policy*, 27(1):47–63, "2002".

[5] J Ranganathan, R Waite, T Searchinger, and C Hanson. How to sustainably feed 10 billion people by 2050, in 21 charts. *World Research Institute, Washington DC, USA. Retrieved on August*, 8:2019, 2018.

[6] Nikos Alexandratos and Jelle Bruinsma. World agriculture towards 2030/2050: the 2012 revision. 2012.

[7] Tim Searchinger, Craig Hanson, Janet Ranganathan, Brian Lipinski, Richard Waite, Robert Winterbottom, Ayesha Dinshaw, Ralph Heimlich, Maryline Boval, Philippe Chemineau, et al. Creating a sustainable food future. a menu of solutions to sustainably feed more than 9 billion people by 2050. world resources report 2013-14: interim findings. 2014.

[8] SFI Representative. Home - sustainable food initiative. `https://www.sfifood.nl/`, 2020. [Online; accessed 01-July-2020].

[9] Rubén Prieto-Díaz. Domain analysis: An introduction. *ACM SIGSOFT Software Engineering Notes*, 15(2):47–54, 1990.

[10] American National Standards Institute. *ANSI-ISA-88.00. 01-2010: Batch Control Part 1: Models and Terminology*. ISA, 2010.

[11] Jeremy Dick, Elizabeth Hull, and Ken Jackson. *Requirements engineering*. Springer, 2017.

[12] Lean stage planning in the face of an incomplete solution: Part 2 - the requirements breakdown structure. https://www.projecttimes.com/articles/lean-stage-planning-in-the-face-of-an-incomplete-solution-part-2-the-requirements-breakdown-structure.html, 2020. [Online; accessed 01-August-2020].

[13] Robert C Tausworthe. The work breakdown structure in software project management. *Journal of Systems and Software*, 1:181–186, 1979.

[14] Dai Clegg and Richard Barker. *Case method fast-track: a RAD approach*. Addison-Wesley Longman Publishing Co., Inc., 1994.

[15] Paul Clements, David Garlan, Reed Little, Robert Nord, and Judith Stafford. Documenting software architectures: views and beyond. In *25th International Conference on Software Engineering, 2003. Proceedings.*, pages 740–741. IEEE, 2003.

[16] Marlon Dumas and Arthur HM Ter Hofstede. Uml activity diagrams as a workflow specification language. In *International conference on the unified modeling language*, pages 76–90. Springer, 2001.

[17] Martin Fowler and James Lewis. Microservices. `https://martinfowler.com/articles/microservices.html`, March 2014. [Online; accessed 26 April, 2020].

[18] Mario Villamizar, Oscar Garcés, Harold Castro, Mauricio Verano, Lorena Salamanca, Rubby Casallas, and Santiago Gil. Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. In *2015 10th Computing Colombian Conference (10CCC)*, pages 583–590. IEEE, 2015.

[19] normalize. `https://www.thefreedictionary.com/normalize`, 2020-04-30. [Online; accessed 30 April, 2020].

[20] Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. Hypertext transfer protocol–http/1.1, 1999.

[21] Material Design. `https://material.io/`, 2020. [Online; accessed 05-August-2020].

[22] International Bureau of Weights, Measures, Barry N Taylor, and Ambler Thompson. *The international system of units (SI)*. US Department of Commerce, Technology Administration, National Institute of . . . , 2001.

[23] Anne Geraci, Freny Katki, Louise McMonegal, Bennett Meyer, John Lane, Paul Wilson, Jane Radatz, Mary Yee, Hugh Porteous, and Fredrick Springsteel. *IEEE standard computer dictionary: Compilation of IEEE standard computer glossaries*. IEEE Press, 1991.

[24] A. M. Altawekji, C. I. Bahrin, T. C. Chirvasuta, P. S. Friptu , A. R. Garban , S. S. van Grieken , D. Hegger, J. Martens, P. Petrov, P. Remkes, and A. E. Voicu. Software/Web Engineering Project, Unit Test Plan, Internet of Food, July 2020.

[25] G van Rossum, B Warsaw, and N Coghlan. Pep8-style guide for python, 2013.

[26] S. Balaji and M. Sundararajan Murugaiyan. Wateerfallvs v-model vs agile: A comparative study on sdlc. 2012.

[27] Winston W Royce. Managing the development of large software systems: concepts and techniques. In *Proceedings of the 9th international conference on Software Engineering*, pages 328–338, 1987.

[28] Martin Fowler, Jim Highsmith, et al. The agile manifesto. *Software Development*, 9(8):28–35, 2001.

[29] Technical University Eindhoven. Final bachelor project. `https://educationguide.tue.nl/programs/bachelor-college/majors/software-science/final-bachelor-project/`, 2020. [Online; accessed 17-June-2020].

[30] Wikipedia contributors. Coronavirus disease 2019 - wikipedia. `https://en.wikipedia.org/wiki/Coronavirus_disease_2019`, 2020. [Online; accessed 16-June-2020].

[31] Reinout Van Rees. Clarity in the usage of the terms ontology, taxonomy and classification. *CIB REPORT*, 284(432):1–8, 2003.

A Running and Deploying the Project

A.1 Running on IDE

A.1.1 Prepare Development Environment

This section describes the tools necessary to prepare the development environment. This setup is tested on Linux Mint 19.3 64 bit version. However, any popular and recent Linux based OS should have the capability to install the necessary tools and run this project. Following are the list of necessary tools,

Miniconda (Python version 3.7 or above)

This project uses *conda* to create and maintain virtual python environments. *Miniconda* is the minimum installer for *conda*. Download the 64 bit version of *miniconda* from the official website[1]. Installation instructions for different platforms can be found on the same page. During the installation there will be possibility to choose the location for the installation. Make sure to install *miniconda* on *$HOME/Programs/miniconda3*. This will make the running of the project much easier.

Docker (version 19.03.8 or above)

This project uses *Docker* containers for deploying all the microservices. Although it is not necessary to run all the microservices during a debug run, PostgreSQL and GraphDB containers are necessary as they are used data storage. On most Linux distributions, *Docker* can be installed using their package manager (e.g. *apt* in Ubuntu). The installation instructions can be found at the official website[2].

Docker Compose (version 1.17.1 or above)

Docker-compose is a very convenient way to run and maintain multiple *Docker* containers at the same time. The official *Docker* website[3] has the installation instructions for *Docker-compose*.

[1]https://docs.conda.io/en/latest/miniconda.html
[2]https://docs.docker.com/engine/install/
[3]https://docs.docker.com/compose/install/

npm (version 6.14.8 or above)

npm is needed for running the frontend application. Necessary installation instructions are given in the official website[4].

Jetbrains Pycharm Community Edition (version 2020.1 or above)

Download and install it from official Jetbrains website[5]. The instructions to install are also available on the same page.

A.1.2 Dependency Management and Virtual Environment

As mentioned earlier, this project uses Conda for creating separate virtual environment for each microservice and their dependency management. Each microservice has a *requirements.yml* file that contains necessary information for creating corresponding virtual environment and installing dependencies in it that are needed to run the microservice.

A.1.3 Opening and Running the Project

- Clone the project from `git@gitlab.tue.nl:20184737/inof-implementation.git` into *inof-implementation*. Make sure to use *–recurse-submodules* option while cloning to initialize and pull all submodules. See corresponding commands in Section A.2.
- Navigate to cloned directory and run the database containers.

```
$ cd inof-implementation
$ docker load --input ./model-sharing-platform/db_files/graphdb_9
    .1.1-free.tar
$ docker-compose up --build --remove-orphans graphdb-container
    pgadmin-container db-container
```

- Create all necessary Python virtual environments and install needed dependencies by running:

```
$ cd model-sharing-platform $ ./create_conda_environments.sh
```

- Open the *inof-implementation/model-sharing-platform* directory with PyCharm.
- Execute the run configuration *Run Backend*. This will start the backend as well as other necessary microservices.
- For running the frontend, executing the following commands. Make sure the frontend is pointing to the correct backend in file $src/environments/environment.ts$.

```
$ cd inof-frontend
$ npm install
$ npm install -g @angular/cli
$ ng serve
```

[4]https://www.npmjs.com/get-npm
[5]https://www.jetbrains.com/pycharm/download/

Now, open a web browser (e.g., Firefox, Google Chrome) and navigate to `http://localhost:4200/login`.

- Optional: The unit-translation-component in the backend uses *redis*[6] for caching Ontology of Units of Measure (OM2.0) related data (details in [1]). If there is a *redis* server running on `localhost:6379`, the backend will use it.

A.2 Run Project Using *docker-compose*

- Clone the code repository from GitLab

  ```
  $ git clone --recurse-submodules <REPO_URL>
  ```

  Replace *<REPO_URL>* with `git@gitlab.tue.nl:20184737/inof-implementation.git`

- Navigate to the cloned repository

  ```
  $ cd inof-implementation
  ```

- Build and bring the containers up

  ```
  $ ./containers.sh up
  ```

  This script automates the docker deployment. Running it will load a custom image of the GraphDB and pull as well as build all other necessary images. Finally, all the built images are bought up. If the *containers.sh* file does not have execute permission, the following command will add that permission,

  ```
  $ chmod +x containers.sh
  ```

- Wait for all the containers to finish booting up. The final output should look similar to the following,

  ```
  Creating network "inof-implementation_inof_vnetwork" with driver "
     bridge"
  Creating graphdb-container ... done
  Creating db-container      ... done
  Creating sweet-sourness-model-access-gateway      ... done
  Creating ingredient-service                       ... done
  Creating tomato-saltiness-model-access-gateway    ... done
  Creating inof-implementation_pgadmin-container_1 ... done
  Creating model-sharing-backend                    ... done
  Creating inof-frontend                            ... done
  ```

- Finally, run the seed script to insert initial setup data into the databases,

  ```
  $ ./model-sharing-platform/seed.sh
  ```

  This can take around 15 minutes. To avoid long waiting time, it is also possible to comment the last line of the *seed.sh* file. This will disable redis caching of the OM2.0 related data.

[6]https://redis.io/

A.3 Running Unit Tests

The unit tests can be run using PyCharm. Follow the instructions for opening and running the project in Section A.1.3 except for the final two steps. At this point, run the gateway and ingredient service by executing run configurations *Gateway Only* and *Ingredient Service Only*. Now, execute the *Backend Test* run configuration to execute all unit tests.

B Endpoints in the Backend

B.1 Authentication API

The Authentication API is responsible for authenticating users and managing authentication related functionality, like retrieving an authentication token, checking authentication, registration

- POST /api/auth_token: upon successful authentication stores a new authentication token in the database
- GET /api/check_auth: returns whether a given token is valid
- POST /api/user/register: registers a new user in the database
- GET /api/own_profile: retrieves the information about a user's profile
- PUT /api/own_profile: edits the information about a user's profile
- POST /api/user/request_password_reset: sends a forgot-password email to the user
- POST /api/user/reset_password: used to reset a forgotten password
- POST /api/user/change_password: used to change password

B.2 Model API

The Model API is responsible for retrieving, editing and adding models to and from the database. The Models retrieved can be specific (i.e. with a certain id) or all Models.

- POST /api/model: stores a new Model in the database
- PUT /api/model/model_id: edits the information about a Model with a given model-id
- DELETE /api/model/id: removes a Model with a given id
- GET /api/model/id: retrieves a model with a given id
- GET /api/models: retrieves all Available Models
- GET /api/own_models: retrieves all Models the user's Company Owns
- GET /api/model/permissions/model_id: Get permissions of the model with the given model id
- PUT /api/model/permissions/model_id: Update permissions for model the given model id

B.3 Food Product API

The Food Product API is responsible for retrieving, editing and adding Food Products to and from the database. The Food Products retrieved can be specific (i.e. with a certain id) or all Food Product.

- POST /api/product: stores a new Food Product in the database
- PUT /api/product/id: edits the information about a Food Product with a given id
- GET /api/product/ingredients: retrieves all the ingredients
- DELETE /api/product/id: removes a Food Product with a given id
- GET /api/product/id: retrieves a Food Product with a given id
- GET /api/products: retrieves all Available Food Products
- GET /api/own_products: retrieves all Food Products the user's Company owns
- GET /api/product/permissions/product_id: Get permissions for the food product with the given product id
- PUT /api/product/permissions/product_id: Update permissions for the food product with the given product id

B.4 Simulation API

The Simulation API is responsible for retrieving, editing and adding Simulations to and from the database. It also is responsible for requesting runs for Simulation, retrieving Simulation results and Simulation status. The Simulations retrieved, deleted and edited can be specific (i.e. with a certain id) or all Simulations in the case of retrieving. Simulation run, result and status are for a Simulation with a given id.

- POST /api/simulation: stores a new Simulation in the database
- PUT /api/simulation/id: edits the information of a Simulation with a given id
- DELETE /api/simulation/id: removes a Simulation with a given id
- GET /api/simulation/id: retrieves a Simulation with a given id
- GET /api/simulations: retrieves all Simulations the user's Company owns
- POST /api/run_simulation/id: request a Simulation run for a Simulation with a given id
- GET /api/simulation_result/id: retrieves a Simulation result for a Simulation with a given id
- GET /api/simulation_status/id: retrieves a Simulation status for a Simulation with a given id

B.5 Company API

The Company API is responsible for retrieving all companies from the database.

- GET /api/companies: retrieves all companies

C Findings for Ontology of Units of Measure

This chapter describes the issues with the OM2.0 ontology [2] that was discovered by the bachelors' students during the development of the unit translation component as part of the SEP assignment (see Section 8.4).

C.1 Initial Assumptions

When the Ontology Based Conversion Library has been developed an assumption has been made upon extensively researching all units of measure. The assumption is that 2 units of measure are comparable (i.e. one can convert a quantity expressed by the initial unit of measure to a quantity expressed by the target unit of measure) if they measure the same dimension or if their base units of measure have the same sum of dimensions for each of the 7 SI dimensions. (More information about these units can be found under the ***Domain*** section of the website http://www.foodvoc.org/resource/om-2/Dimension).

C.2 Dimensions without Alternative Units

A number of dimensions can not give alternative units because they relate to only one instance of units of measure. As a result, conversion from or to these units could not be tested. Table C.1 lists these dimensions and corresponding units.

Table C.1: Dimensions without alternative units

Dimension Name	Dimension URI	Unit
Volumetric heat capacity	http://www.ontology-of-units-of-measure.org/resource/om-2/volumetricHeatCapacity-Dimension	joule per cubic metre kelvin
Thermal resistance	http://www.ontology-of-units-of-measure.org/resource/om-2/thermalResistance-Dimension	kelvin per watt.
Thermal insulance	http://www.ontology-of-units-of-measure.org/resource/om-2/thermalInsulance-Dimension	square metre kelvin per watt
Thermal conductivity	http://www.ontology-of-units-of-measure.org/resource/om-2/thermalConductivity-Dimension	watt per metre kelvin
Surface tension	http://www.ontology-of-units-of-measure.org/resource/om-2/surfaceTension-Dimension	newton per metre
Specific volume	http://www.ontology-of-units-of-measure.org/resource/om-2/specificVolume-Dimension	cubic metre per kilogram

Specific entropy or specific heat capacity	http://www.ontology-of-units-of-measure.org/resource/om-2/specificEntropyOrSpecificHeatCapacity-Dimension	joule per kelvin kilogram
Reluctance	http://www.ontology-of-units-of-measure.org/resource/om-2/reluctance-Dimension	reciprocal henry
Radiance	http://www.ontology-of-units-of-measure.org/resource/om-2/radiance-Dimension	watt per square metre steradian
Power density	http://www.ontology-of-units-of-measure.org/resource/om-2/powerDensity-Dimension	watt per square metre
Permittivity	http://www.ontology-of-units-of-measure.org/resource/om-2/permittivity-Dimension	farad per metre
Permeability of free space	http://www.ontology-of-units-of-measure.org/resource/om-2/permeabilityOfFreeSpace-Dimension	henry per metre
Molar entropy, molar heat capacity, or gas constant	http://www.ontology-of-units-of-measure.org/resource/om-2/molarEntropyOrMolarHeatCapacityOrGasConstant-Dimension	joule per kelvin mole
Molar energy	http://www.ontology-of-units-of-measure.org/resource/om-2/molarEnergy-Dimension	joule per mole
Mass flow	http://www.ontology-of-units-of-measure.org/resource/om-2/massFlow-Dimension	kilogram per second
Heat transfer coefficient	http://www.ontology-of-units-of-measure.org/resource/om-2/heatTransferCoefficient-Dimension	watt per square metre kelvin
Exposure	http://www.ontology-of-units-of-measure.org/resource/om-2/exposure-Dimension	lux second
Entropy or heat capacity	http://www.ontology-of-units-of-measure.org/resource/om-2/entropyOrHeatCapacity-Dimension	joule per kelvin
Energy density	http://www.ontology-of-units-of-measure.org/resource/om-2/energyDensity-Dimension	joule per cubic metre
Electrical resistivity	http://www.ontology-of-units-of-measure.org/resource/om-2/electricalResistivity-Dimension	ohm metre
Electrical conductivity	http://www.ontology-of-units-of-measure.org/resource/om-2/electricalConductivity-Dimension	siemens per metre
Electric flux density	http://www.ontology-of-units-of-measure.org/resource/om-2/electricFluxDensity-Dimension	coulomb per square metre
Electric charge density	http://www.ontology-of-units-of-measure.org/resource/om-2/electricChargeDensity-Dimension	coulomb per cubic metre
Current density	http://www.ontology-of-units-of-measure.org/resource/om-2/currentDensity-Dimension	ampere per square metre
Catalytic activity concentration	http://www.ontology-of-units-of-measure.org/resource/om-2/catalyticActivityConcentration-Dimension	katal per cubic metre
Angular acceleration	http://www.ontology-of-units-of-measure.org/resource/om-2/angularAcceleration-Dmension	radian per second squared
Absorbed dose rate	http://www.ontology-of-units-of-measure.org/resource/om-2/absorbedDoseRate-Dimension	gray per second

Angular speed dimension can not be fully tested as there are only 2 units of measure: radian per second and millisecond (angle) per year, while there are many other units of measure measuring this dimension that can be found upon searching on Google. Please refer to the website http://www.ontology-of-units-of-measure.org/resource/om-2/angularSpeed-Dimension for more information.

C.3 Dimension One

In OM2.0 there exists a dimension called DimensionOne (more info at http://www.ontology-of-units-of-measure.org/resource/om-2/dimensionOne). This dimension is defined in order to express units of measure that do not present a dimension factor in their description. As a result, units with this dimension are deemed convertible since the dimension information is used by the unit conversion component to determine convertibility. For example:

- square metre per square metre: this unit of measure is used to describe a 2D quantity of length fraction.
- kilogram per kilogram: this unit of measure is used to describe a 1D quantity of mass fraction.
- metre per metre: this unit of measure is used to describe a 1D quantity of length fraction.
- magnitude: this unit of measure is used to describe the brightness of stars.
- acoustic firmness: this unit of measure is used to describe an acoustic phenomenon.

C.4 Insufficient Dimension Information

The web resource presents an easy way of understanding which dimension is measured by each unit of measure, through a predicate *hasDimension*. The library exploits this structure, by defining that if two units of measure have the same dimension, then they are comparable. However, there are *2192* out of *14656* units of measure that do not present this predicate. This fact has made the conversion protocol of this library more complex, resulting in computational performance penalties. The units of measure that do not present this predicate are listed in Table C.2.

Table C.2: Units without sufficient dimension information

Unit URI
http://www.ontology-of-units-of-measure.org/resource/om-2/magnitudePerSecond-AngleSquared
http://www.ontology-of-units-of-measure.org/resource/om-2/JapaneseYen
http://www.ontology-of-units-of-measure.org/resource/om-2/molePerTerametre
http://www.ontology-of-units-of-measure.org/resource/om-2/reciprocalPartsPerMillionPerYear
http://www.ontology-of-units-of-measure.org/resource/om-2/megajoulePerSquareMetreDay
http://www.ontology-of-units-of-measure.org/resource/om-2/byte
http://www.ontology-of-units-of-measure.org/resource/om-2/SwissFranc
http://www.ontology-of-units-of-measure.org/resource/om-2/yobibit
http://www.ontology-of-units-of-measure.org/resource/om-2/zebibyte
http://www.ontology-of-units-of-measure.org/resource/om-2/millimolePerMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/zeptomolePerMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/NewZealandDollar

http://www.ontology-of-units-of-measure.org/resource/om-2/squareMetreHertz
http://www.ontology-of-units-of-measure.org/resource/om-2/colonyFormingUnit
http://www.ontology-of-units-of-measure.org/resource/om-2/MexicanPeso
http://www.ontology-of-units-of-measure.org/resource/om-2/exbibyte
http://www.ontology-of-units-of-measure.org/resource/om-2/partsPerMillionPerYear
http://www.ontology-of-units-of-measure.org/resource/om-2/CanadianDollar
http://www.ontology-of-units-of-measure.org/resource/om-2/squareMetreKelvin
http://www.ontology-of-units-of-measure.org/resource/om-2/gibibyte
http://www.ontology-of-units-of-measure.org/resource/om-2/centimetrePerDay
http://www.ontology-of-units-of-measure.org/resource/om-2/kilogramPerHectare
http://www.ontology-of-units-of-measure.org/resource/om-2/metreKilogramPerSecond-Time
http://www.ontology-of-units-of-measure.org/resource/om-2/molePerFemtometre
http://www.ontology-of-units-of-measure.org/resource/om-2/colonyFormingUnitPer25Millilitre
http://www.ontology-of-units-of-measure.org/resource/om-2/centimetre-Gram-SecondElectromagneticSystemOfUnits
http://www.ontology-of-units-of-measure.org/resource/om-2/wattPerCubicmetre
http://www.ontology-of-units-of-measure.org/resource/om-2/molePerNanometre
http://www.ontology-of-units-of-measure.org/resource/om-2/molePerGigametre
http://www.ontology-of-units-of-measure.org/resource/om-2/UnitedStatesDollar
http://www.ontology-of-units-of-measure.org/resource/om-2/moleMicrometreReciprocalSquareCentimetre
http://www.ontology-of-units-of-measure.org/resource/om-2/wattPerHertz
http://www.ontology-of-units-of-measure.org/resource/om-2/MexicanPeso
http://www.ontology-of-units-of-measure.org/resource/om-2/wattPerSecond-AngleSquared
http://www.ontology-of-units-of-measure.org/resource/om-2/molePerAttometre
http://www.ontology-of-units-of-measure.org/resource/om-2/kilojoulePerSquareMetreDay
http://www.ontology-of-units-of-measure.org/resource/om-2/moleMicrometre
http://www.ontology-of-units-of-measure.org/resource/om-2/nanosecond-TimeSquared
http://www.ontology-of-units-of-measure.org/resource/om-2/gibibit
http://www.ontology-of-units-of-measure.org/resource/om-2/petabyte
http://www.ontology-of-units-of-measure.org/resource/om-2/tebibyte
http://www.ontology-of-units-of-measure.org/resource/om-2/degreeCelsiusPerSecond-Time
http://www.ontology-of-units-of-measure.org/resource/om-2/squareMetreSteradian
http://www.ontology-of-units-of-measure.org/resource/om-2/kelvinMole
http://www.ontology-of-units-of-measure.org/resource/om-2/second-TimeToThePower-2
http://www.ontology-of-units-of-measure.org/resource/om-2/micromolePerMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/reciprocalDegreeCelsius
http://www.ontology-of-units-of-measure.org/resource/om-2/gramPerSquareMetreMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/deltaA450
http://www.ontology-of-units-of-measure.org/resource/om-2/molePerKilogram
http://www.ontology-of-units-of-measure.org/resource/om-2/degreeCelsiusDay
http://www.ontology-of-units-of-measure.org/resource/om-2/terabit
http://www.ontology-of-units-of-measure.org/resource/om-2/exbibit
http://www.ontology-of-units-of-measure.org/resource/om-2/zettamolePerMetre

http://www.ontology-of-units-of-measure.org/resource/om-2/squareMetreNanometre
http://www.ontology-of-units-of-measure.org/resource/om-2/kilogramPerGigajoule
http://www.ontology-of-units-of-measure.org/resource/om-2/metre-Kilogram-Second-AmpereSystemOfUnits
http://www.ontology-of-units-of-measure.org/resource/om-2/yobibyte
http://www.ontology-of-units-of-measure.org/resource/om-2/gigabit
http://www.ontology-of-units-of-measure.org/resource/om-2/ChineseYuan
http://www.ontology-of-units-of-measure.org/resource/om-2/exabit
http://www.ontology-of-units-of-measure.org/resource/om-2/reciprocalSquareMetreReciprocalGram
http://www.ontology-of-units-of-measure.org/resource/om-2/centimetre-Gram-SecondElectrostaticSystemOfUnits
http://www.ontology-of-units-of-measure.org/resource/om-2/centimetre-Gram-Second-FranklinSystemOfUnits
http://www.ontology-of-units-of-measure.org/resource/om-2/centimetrePerCubicCentimetre
http://www.ontology-of-units-of-measure.org/resource/om-2/second-TimePerSquareMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/hartley
http://www.ontology-of-units-of-measure.org/resource/om-2/deltaA450
http://www.ontology-of-units-of-measure.org/resource/om-2/SouthKoreanWon
http://www.ontology-of-units-of-measure.org/resource/om-2/hectareDay
http://www.ontology-of-units-of-measure.org/resource/om-2/NorwegianKrone
http://www.ontology-of-units-of-measure.org/resource/om-2/molePerCentimetre
http://www.ontology-of-units-of-measure.org/resource/om-2/micromolePerSecond-TimeGram
http://www.ontology-of-units-of-measure.org/resource/om-2/1000ColonyFormingUnit
http://www.ontology-of-units-of-measure.org/resource/om-2/molePermegametre
http://www.ontology-of-units-of-measure.org/resource/om-2/1000ColonyFormingUnitPerMillilitre
http://www.ontology-of-units-of-measure.org/resource/om-2/petabit
http://www.ontology-of-units-of-measure.org/resource/om-2/molePerPetametre
http://www.ontology-of-units-of-measure.org/resource/om-2/SwedishKrona
http://www.ontology-of-units-of-measure.org/resource/om-2/reciprocalGram
http://www.ontology-of-units-of-measure.org/resource/om-2/yottasecond-TimeSquared
http://www.ontology-of-units-of-measure.org/resource/om-2/litrePerMole
http://www.ontology-of-units-of-measure.org/resource/om-2/steradianSquareMetreHertz
http://www.ontology-of-units-of-measure.org/resource/om-2/IndianRupee
http://www.ontology-of-units-of-measure.org/resource/om-2/molePerYottametre
http://www.ontology-of-units-of-measure.org/resource/om-2/exabyte
http://www.ontology-of-units-of-measure.org/resource/om-2/molePerMillimetre
http://www.ontology-of-units-of-measure.org/resource/om-2/metrePerDay
http://www.ontology-of-units-of-measure.org/resource/om-2/reciprocalWatt
http://www.ontology-of-units-of-measure.org/resource/om-2/hartley
http://www.ontology-of-units-of-measure.org/resource/om-2/megaeuro
http://www.ontology-of-units-of-measure.org/resource/om-2/squareMetreSecond-Time
http://www.ontology-of-units-of-measure.org/resource/om-2/gramPerJoule
http://www.ontology-of-units-of-measure.org/resource/om-2/gigasecond-TimeSquared
http://www.ontology-of-units-of-measure.org/resource/om-2/baud

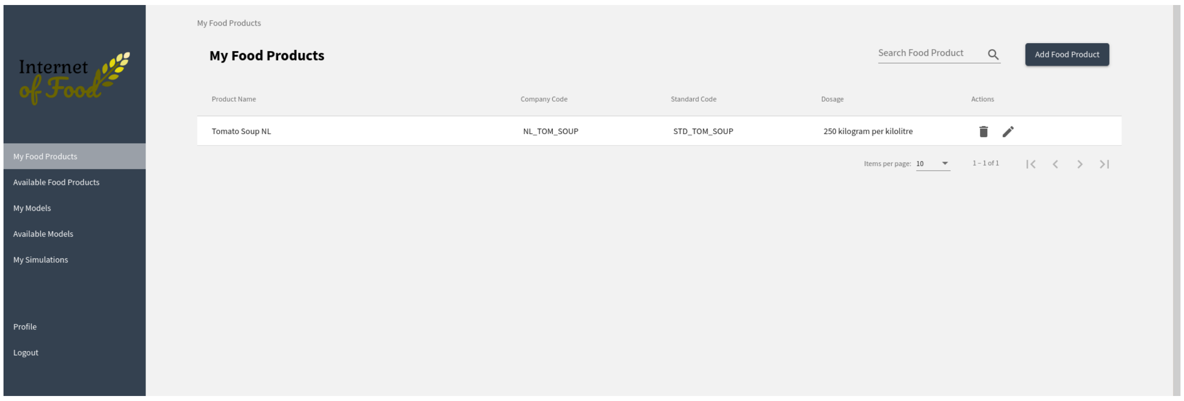
http://www.ontology-of-units-of-measure.org/resource/om-2/kilobit
http://www.ontology-of-units-of-measure.org/resource/om-2/zeptosecond-TimeSquared
http://www.ontology-of-units-of-measure.org/resource/om-2/microsecond-TimeSquared
http://www.ontology-of-units-of-measure.org/resource/om-2/molePerExametre
http://www.ontology-of-units-of-measure.org/resource/om-2/wattSquareMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/terabyte
http://www.ontology-of-units-of-measure.org/resource/om-2/TurkishLira
http://www.ontology-of-units-of-measure.org/resource/om-2/solarMassPerGigayearCubicKiloparsec
http://www.ontology-of-units-of-measure.org/resource/om-2/euro
http://www.ontology-of-units-of-measure.org/resource/om-2/bit
http://www.ontology-of-units-of-measure.org/resource/om-2/jansky
http://www.ontology-of-units-of-measure.org/resource/om-2/decimolePerMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/BrazilianReal
http://www.ontology-of-units-of-measure.org/resource/om-2/wattPerNanometre
http://www.ontology-of-units-of-measure.org/resource/om-2/poundSterling
http://www.ontology-of-units-of-measure.org/resource/om-2/metrePerCubicMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/mebibit
http://www.ontology-of-units-of-measure.org/resource/om-2/colonyFormingUnitPerMillilitre
http://www.ontology-of-units-of-measure.org/resource/om-2/JapaneseYen
http://www.ontology-of-units-of-measure.org/resource/om-2/molePerHectometre
http://www.ontology-of-units-of-measure.org/resource/om-2/reciprocalDegreeCelsiusDay
http://www.ontology-of-units-of-measure.org/resource/om-2/cubicMetrePerMole
http://www.ontology-of-units-of-measure.org/resource/om-2/molePerPicometre
http://www.ontology-of-units-of-measure.org/resource/om-2/shannon
http://www.ontology-of-units-of-measure.org/resource/om-2/molePerYoctometre
http://www.ontology-of-units-of-measure.org/resource/om-2/metreKilogram
http://www.ontology-of-units-of-measure.org/resource/om-2/wattPerSteradian
http://www.ontology-of-units-of-measure.org/resource/om-2/gramPerSquareMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/SingaporeDollar
http://www.ontology-of-units-of-measure.org/resource/om-2/decasecond-TimeSquared
http://www.ontology-of-units-of-measure.org/resource/om-2/tonnePerHectare
http://www.ontology-of-units-of-measure.org/resource/om-2/microgramPerSquareMetreSecond-Time
http://www.ontology-of-units-of-measure.org/resource/om-2/petamolePerMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/zettabit
http://www.ontology-of-units-of-measure.org/resource/om-2/centisecond-TimeSquared
http://www.ontology-of-units-of-measure.org/resource/om-2/HongKongDollar
http://www.ontology-of-units-of-measure.org/resource/om-2/femtomolePerMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/BrazilianReal
http://www.ontology-of-units-of-measure.org/resource/om-2/megaeuroPerMegawatt
http://www.ontology-of-units-of-measure.org/resource/om-2/wattPerSteradianSquareMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/voltPerWatt
http://www.ontology-of-units-of-measure.org/resource/om-2/kilogramSquareMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/kilosecond-TimeSquared

http://www.ontology-of-units-of-measure.org/resource/om-2/yottamolePerMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/UnitedStatesDollar
http://www.ontology-of-units-of-measure.org/resource/om-2/megametrePerKilojoule
http://www.ontology-of-units-of-measure.org/resource/om-2/exasecond-TimeSquared
http://www.ontology-of-units-of-measure.org/resource/om-2/squareMetrePerGram
http://www.ontology-of-units-of-measure.org/resource/om-2/GaussianSystemOfUnits
http://www.ontology-of-units-of-measure.org/resource/om-2/megajoulePerSquareMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/megaeuroPerMegatonne
http://www.ontology-of-units-of-measure.org/resource/om-2/RussianRuble
http://www.ontology-of-units-of-measure.org/resource/om-2/pebibyte
http://www.ontology-of-units-of-measure.org/resource/om-2/tebibit
http://www.ontology-of-units-of-measure.org/resource/om-2/kilogramPerMole
http://www.ontology-of-units-of-measure.org/resource/om-2/petasecond-TimeSquared
http://www.ontology-of-units-of-measure.org/resource/om-2/kibibit
http://www.ontology-of-units-of-measure.org/resource/om-2/byte
http://www.ontology-of-units-of-measure.org/resource/om-2/SwissFranc
http://www.ontology-of-units-of-measure.org/resource/om-2/gigayearCubicParsec
http://www.ontology-of-units-of-measure.org/resource/om-2/femtosecond-TimeSquared
http://www.ontology-of-units-of-measure.org/resource/om-2/joulePerSquareMetreSecond-Time
http://www.ontology-of-units-of-measure.org/resource/om-2/megaeuroPerPetajoule
http://www.ontology-of-units-of-measure.org/resource/om-2/centimetre-Gram-SecondSystemOfUnits
http://www.ontology-of-units-of-measure.org/resource/om-2/AustralianDollar
http://www.ontology-of-units-of-measure.org/resource/om-2/gigayearCubicKiloparsec
http://www.ontology-of-units-of-measure.org/resource/om-2/cubicMetreKelvin
http://www.ontology-of-units-of-measure.org/resource/om-2/centimetre-Gram-Second-BiotSystemOfUnits
http://www.ontology-of-units-of-measure.org/resource/om-2/attomolePerMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/mebibyte
http://www.ontology-of-units-of-measure.org/resource/om-2/amperePerWatt
http://www.ontology-of-units-of-measure.org/resource/om-2/molePerMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/metrePerSecond-TimePerMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/yottabit
http://www.ontology-of-units-of-measure.org/resource/om-2/decamolePerMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/gramPerSquareMetreDay
http://www.ontology-of-units-of-measure.org/resource/om-2/reciprocalAtmosphere-Standard
http://www.ontology-of-units-of-measure.org/resource/om-2/bitPerSecond-Time
http://www.ontology-of-units-of-measure.org/resource/om-2/molePerMicrometre
http://www.ontology-of-units-of-measure.org/resource/om-2/kilometrePerSecond-TimePerMegaparsec
http://www.ontology-of-units-of-measure.org/resource/om-2/yoctomolePerMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/examolePerMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/pebibit
http://www.ontology-of-units-of-measure.org/resource/om-2/gramPerSquareMetreSecond-Time

http://www.ontology-of-units-of-measure.org/resource/om-2/kilogramPerSquareMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/gigabyte
http://www.ontology-of-units-of-measure.org/resource/om-2/SouthKoreanWon
http://www.ontology-of-units-of-measure.org/resource/om-2/wattPerSteradianSquareMetreHertz
http://www.ontology-of-units-of-measure.org/resource/om-2/NorwegianKrone
http://www.ontology-of-units-of-measure.org/resource/om-2/second-TimeSquared
http://www.ontology-of-units-of-measure.org/resource/om-2/megamolePerMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/terasecond-TimeSquared
http://www.ontology-of-units-of-measure.org/resource/om-2/microgramPerJoule
http://www.ontology-of-units-of-measure.org/resource/om-2/wattPerSquareMetreNanometre
http://www.ontology-of-units-of-measure.org/resource/om-2/AustralianDollar
http://www.ontology-of-units-of-measure.org/resource/om-2/hectosecond-TimeSquared
http://www.ontology-of-units-of-measure.org/resource/om-2/zettasecond-TimeSquared
http://www.ontology-of-units-of-measure.org/resource/om-2/gramPerSquareMetreCentimetre
http://www.ontology-of-units-of-measure.org/resource/om-2/degreeCelsiusPerMinute-Time
http://www.ontology-of-units-of-measure.org/resource/om-2/joulePerSquareMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/InternationalSystemOfUnits
http://www.ontology-of-units-of-measure.org/resource/om-2/IndianRupee
http://www.ontology-of-units-of-measure.org/resource/om-2/molePerDecimetre
http://www.ontology-of-units-of-measure.org/resource/om-2/kilobyte
http://www.ontology-of-units-of-measure.org/resource/om-2/teramolePerMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/megabit
http://www.ontology-of-units-of-measure.org/resource/om-2/joulePerSquareMetreDay
http://www.ontology-of-units-of-measure.org/resource/om-2/tonnePerCubicmetre
http://www.ontology-of-units-of-measure.org/resource/om-2/SouthAfricanRand
http://www.ontology-of-units-of-measure.org/resource/om-2/litrePer100Kilometre
http://www.ontology-of-units-of-measure.org/resource/om-2/colonyFormingUnitPerGram
http://www.ontology-of-units-of-measure.org/resource/om-2/gramPerMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/baud
http://www.ontology-of-units-of-measure.org/resource/om-2/reciprocalKelvin
http://www.ontology-of-units-of-measure.org/resource/om-2/decisecond-TimeSquared
http://www.ontology-of-units-of-measure.org/resource/om-2/zettabyte
http://www.ontology-of-units-of-measure.org/resource/om-2/litrePerHour
http://www.ontology-of-units-of-measure.org/resource/om-2/ChineseYuan
http://www.ontology-of-units-of-measure.org/resource/om-2/picomolePerMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/hectomolePerMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/bit
http://www.ontology-of-units-of-measure.org/resource/om-2/zebibit
http://www.ontology-of-units-of-measure.org/resource/om-2/gigamolePerMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/poundSterling
http://www.ontology-of-units-of-measure.org/resource/om-2/yoctosecond-TimeSquared
http://www.ontology-of-units-of-measure.org/resource/om-2/SouthAfricanRand
http://www.ontology-of-units-of-measure.org/resource/om-2/kibibyte
http://www.ontology-of-units-of-measure.org/resource/om-2/metreKelvin

http://www.ontology-of-units-of-measure.org/resource/om-2/shannon
http://www.ontology-of-units-of-measure.org/resource/om-2/SingaporeDollar
http://www.ontology-of-units-of-measure.org/resource/om-2/centimolePerMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/squareMetrePerSquareMetreDay
http://www.ontology-of-units-of-measure.org/resource/om-2/kelvinKilogram
http://www.ontology-of-units-of-measure.org/resource/om-2/SwedishKrona
http://www.ontology-of-units-of-measure.org/resource/om-2/wattPerSquareMetreHertz
http://www.ontology-of-units-of-measure.org/resource/om-2/molePerDecametre
http://www.ontology-of-units-of-measure.org/resource/om-2/molePerZettametre
http://www.ontology-of-units-of-measure.org/resource/om-2/degreeCelsiusPerHour
http://www.ontology-of-units-of-measure.org/resource/om-2/steradianSquareMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/kilogramSecond-TimeToThePower-2
http://www.ontology-of-units-of-measure.org/resource/om-2/HongKongDollar
http://www.ontology-of-units-of-measure.org/resource/om-2/megabyte
http://www.ontology-of-units-of-measure.org/resource/om-2/millisecond-TimeSquared
http://www.ontology-of-units-of-measure.org/resource/om-2/molePerZeptometre
http://www.ontology-of-units-of-measure.org/resource/om-2/kilomolePerMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/second-TimePerDay
http://www.ontology-of-units-of-measure.org/resource/om-2/picosecond-TimeSquared
http://www.ontology-of-units-of-measure.org/resource/om-2/pascalSecond-TimeSquareMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/megasecond-TimeSquared
http://www.ontology-of-units-of-measure.org/resource/om-2/squareMetreDay
http://www.ontology-of-units-of-measure.org/resource/om-2/NewZealandDollar
http://www.ontology-of-units-of-measure.org/resource/om-2/colonyFormingUnit
http://www.ontology-of-units-of-measure.org/resource/om-2/kilogramPerHectareDay
http://www.ontology-of-units-of-measure.org/resource/om-2/moleMicrometreReciprocalSquareCentimetreReciprocalSecond-Time
http://www.ontology-of-units-of-measure.org/resource/om-2/gramPerMegajoule
http://www.ontology-of-units-of-measure.org/resource/om-2/attosecond-TimeSquared
http://www.ontology-of-units-of-measure.org/resource/om-2/reciprocalSquareMetreReciprocalMetre
http://www.ontology-of-units-of-measure.org/resource/om-2/milligramPerKilometre
http://www.ontology-of-units-of-measure.org/resource/om-2/molePerKilometre
http://www.ontology-of-units-of-measure.org/resource/om-2/CanadianDollar
http://www.ontology-of-units-of-measure.org/resource/om-2/TurkishLira
http://www.ontology-of-units-of-measure.org/resource/om-2/solarMassPerGigayearCubicParsec
http://www.ontology-of-units-of-measure.org/resource/om-2/RussianRuble
http://www.ontology-of-units-of-measure.org/resource/om-2/euro
http://www.ontology-of-units-of-measure.org/resource/om-2/yottabyte
http://www.ontology-of-units-of-measure.org/resource/om-2/jansky
http://www.ontology-of-units-of-measure.org/resource/om-2/nanomolePerMetr

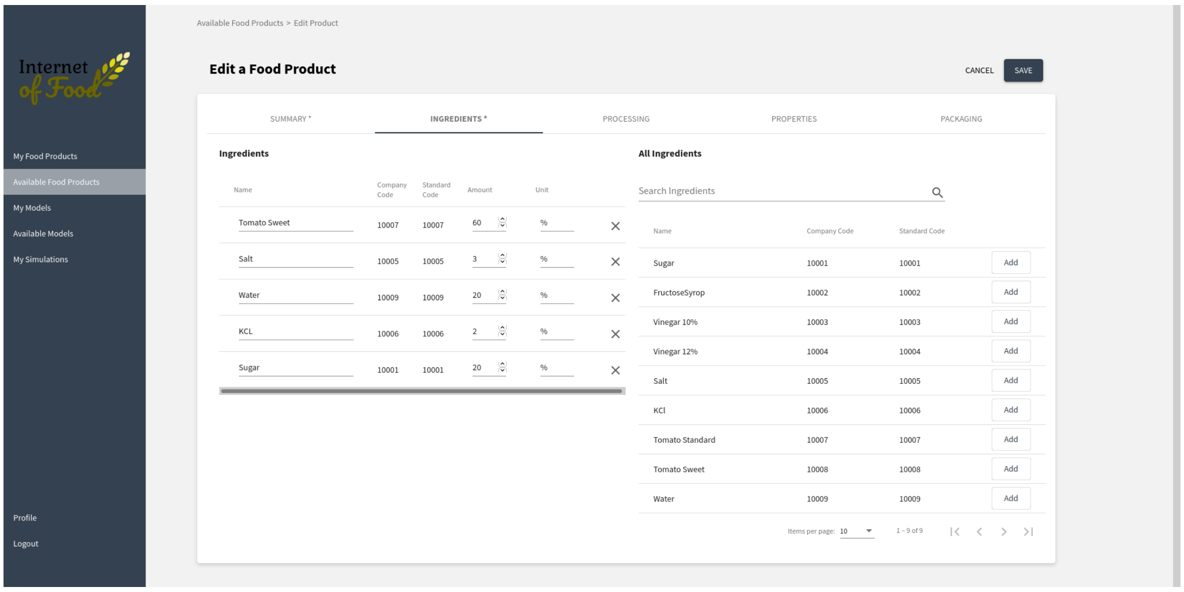
D MaaS Frontend Pages



My Food Products

Product Name	Company Code	Standard Code	Dosage	Actions
Tomato Soup NL	NL_TOM_SOUP	STD_TOM_SOUP	250 kilogram per kilolitre	

Figure D.1: Available food product definitions for logged-in user



Edit a Food Product

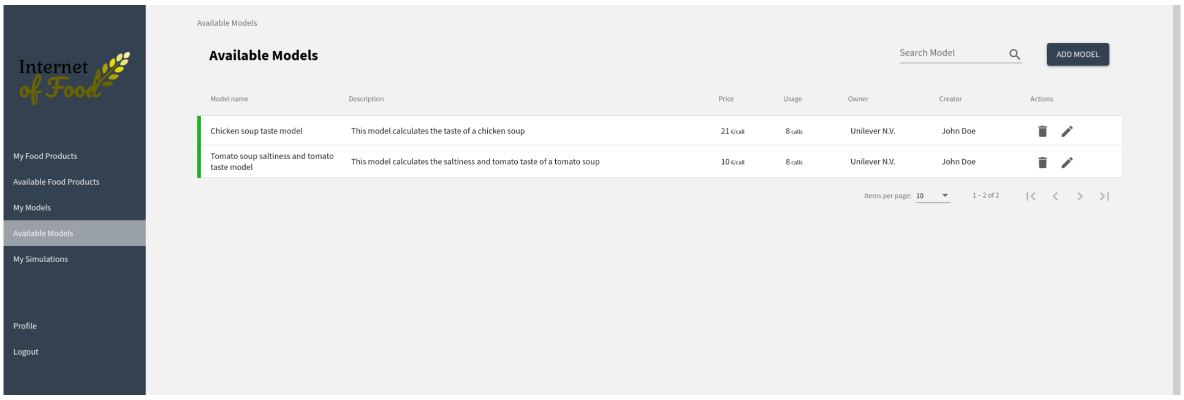
Ingredients

Name	Company Code	Standard Code	Amount	Unit
Tomato Sweet	10007	10007	60	%
Salt	10005	10005	3	%
Water	10009	10009	20	%
KCL	10006	10006	2	%
Sugar	10001	10001	20	%

All Ingredients

Name	Company Code	Standard Code	
Sugar	10001	10001	Add
FructoseSyrop	10002	10002	Add
Vinegar 10%	10003	10003	Add
Vinegar 12%	10004	10004	Add
Salt	10005	10005	Add
KCl	10006	10006	Add
Tomato Standard	10007	10007	Add
Tomato Sweet	10008	10008	Add
Water	10009	10009	Add

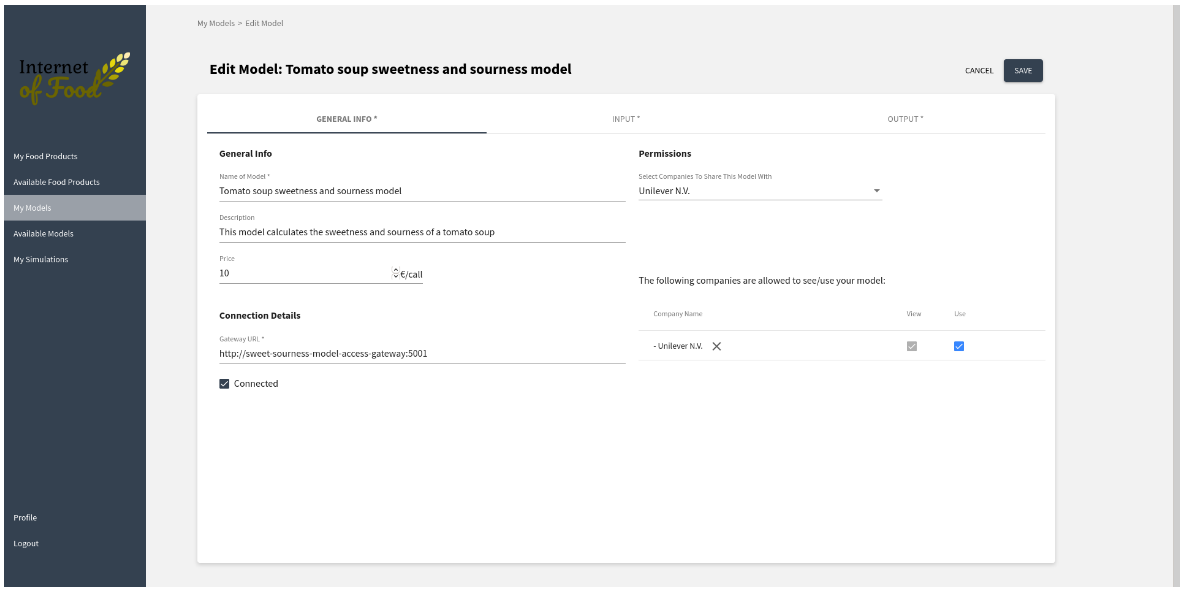
Figure D.2: Ingredient list of food product definition



Available Models

Model name	Description	Price	Usage	Owner	Creator	Actions
Chicken soup taste model	This model calculates the taste of a chicken soup	21 €/call	8 calls	Unilever N.V.	John Doe	
Tomato soup saltiness and tomato taste model	This model calculates the saltiness and tomato taste of a tomato soup	10 €/call	8 calls	Unilever N.V.	John Doe	

Figure D.3: List of available model for logged-in user



Edit Model: Tomato soup sweetness and sourness model

General Info

Name of Model *: Tomato soup sweetness and sourness model

Description: This model calculates the sweetness and sourness of a tomato soup

Price: 10 €/call

Connection Details

Gateway URL *: http://sweet-sourness-model-access-gateway:5001

☑ Connected

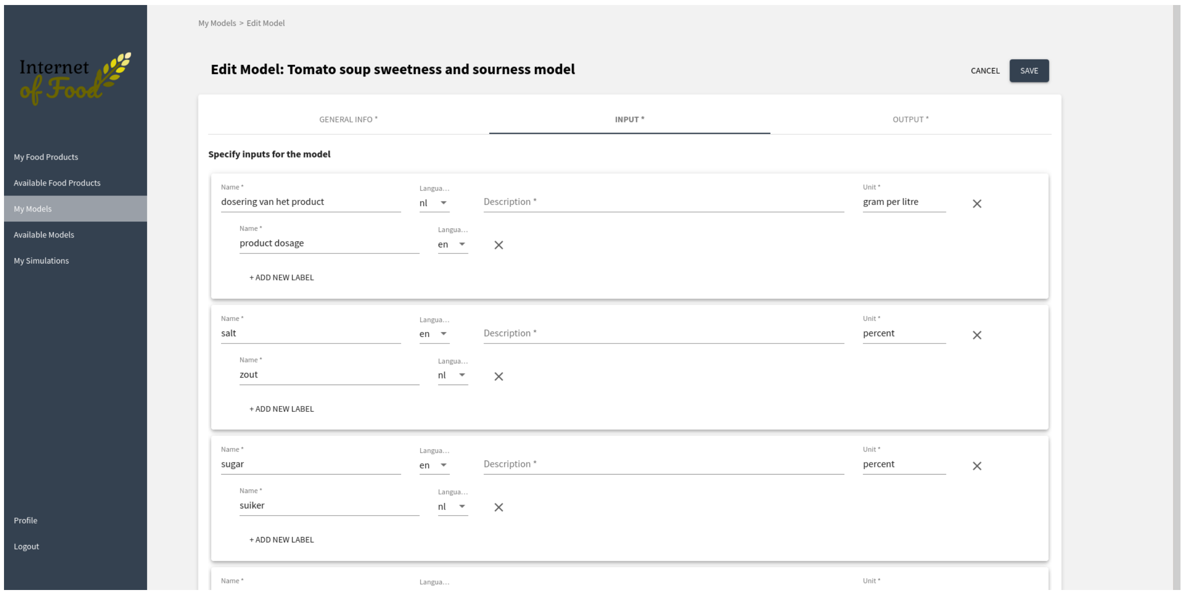
Permissions

Select Companies To Share This Model With: Unilever N.V.

The following companies are allowed to see/use your model:

Company Name	View	Use
- Unilever N.V.	✓	✓

Figure D.4: Edit page for model properties and related permissions



Edit Model: Tomato soup sweetness and sourness model

Specify inputs for the model

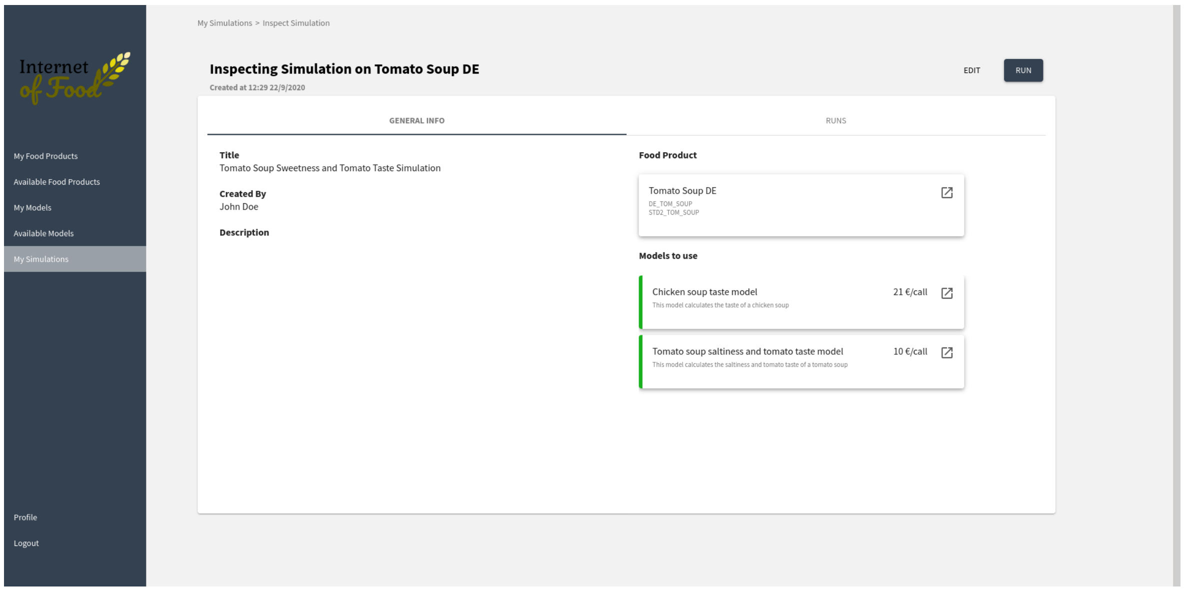
Name *: dosering van het product — Langua...: nl — Description * — Unit *: gram per litre
Name *: product dosage — Langua...: en

Name *: salt — Langua...: en — Description * — Unit *: percent
Name *: zout — Langua...: nl

Name *: sugar — Langua...: en — Description * — Unit *: percent
Name *: suiker — Langua...: nl

Name * — Langua... — Unit *

Figure D.5: Edit page of model parameters



Inspecting Simulation on Tomato Soup DE
Created at 12:29 22/9/2020

Title
Tomato Soup Sweetness and Tomato Taste Simulation

Created By
John Doe

Description

Food Product

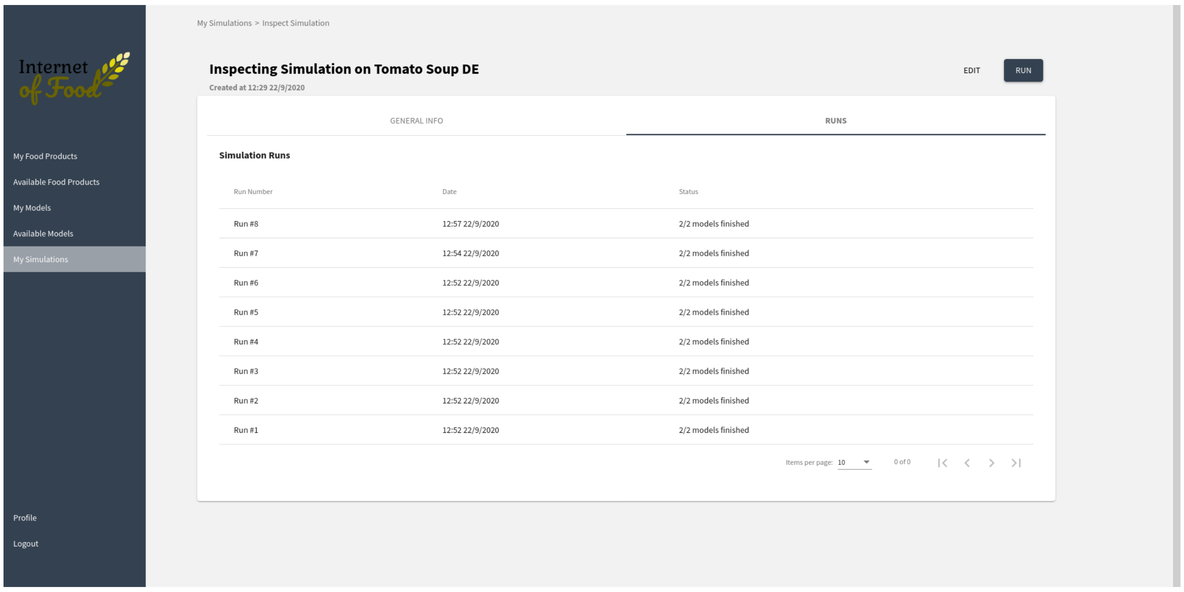
Tomato Soup DE
DE_TOM_SOUP
STD2_TOM_SOUP

Models to use

Chicken soup taste model — 21 €/call
This model calculates the taste of a chicken soup

Tomato soup saltiness and tomato taste model — 10 €/call
This model calculates the saltiness and tomato taste of a tomato soup

Figure D.6: Inspect page for simulation



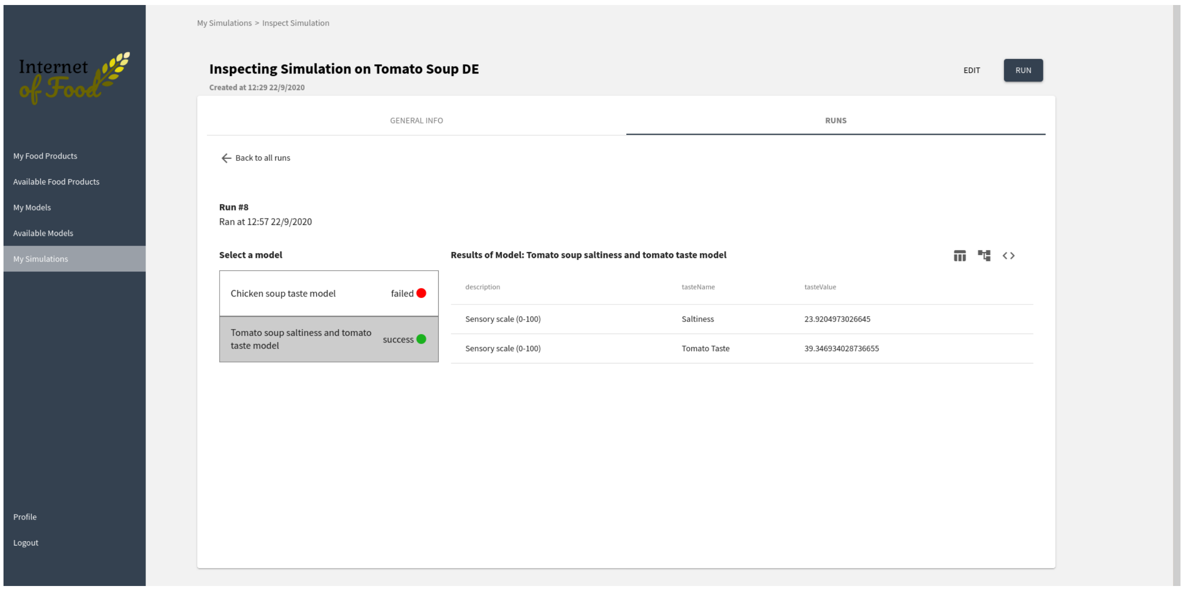
Inspecting Simulation on Tomato Soup DE
Created at 12:29 22/9/2020

Simulation Runs

Run Number	Date	Status
Run #8	12:57 22/9/2020	2/2 models finished
Run #7	12:54 22/9/2020	2/2 models finished
Run #6	12:52 22/9/2020	2/2 models finished
Run #5	12:52 22/9/2020	2/2 models finished
Run #4	12:52 22/9/2020	2/2 models finished
Run #3	12:52 22/9/2020	2/2 models finished
Run #2	12:52 22/9/2020	2/2 models finished
Run #1	12:52 22/9/2020	2/2 models finished

Items per page: 10 0 of 0

Figure D.7: Page showing list of all executed simulations



Inspecting Simulation on Tomato Soup DE
Created at 12:29 22/9/2020

Run #8
Ran at 12:57 22/9/2020

Select a model

Chicken soup taste model	failed 🔴
Tomato soup saltiness and tomato taste model	success 🟢

Results of Model: Tomato soup saltiness and tomato taste model

description	tasteName	tasteValue
Sensory scale (0-100)	Saltiness	23.9204973026645
Sensory scale (0-100)	Tomato Taste	39.346934028736655

Figure D.8: Simulation result page showing results from multiple models

About Author

Hossain Muhammad Muctadir received his bachelor’s degree in Information Technology with a Software Engineering major from the University of Dhaka, Bangladesh (2013) and master’s degree in Software Systems Engineering from RWTH Aachen University, Germany (2017). He worked with FEV GmbH for his master's thesis where he studied Software Product Line (SPL) in the context of the automotive industry and implemented a similarity analysis framework that helps to identify SPLs from existing software components. Hossain has been working as a software developer for more than five years in different organizations, both in Bangladesh and Germany. During his professional career, he developed desktop and web applications as well as augmented reality applications for HoloLens. He also worked in research projects since his bachelor’s studies and made several publications.

Mr. Muctadir joined Technical University of Eindhoven from October 2018 as a Professional Doctorate in Engineering (PDEng) trainee in the Software Technology program offered by the Stan Ackermans Institute. During his traineeship, Mr. Muctadir participated in the development of several software intensive products in collaboration with global players like ASML and Philips Healthcare. In January 2020, he started working on his ten-month long PDEng graduation project in the context of the Internet of Food project, which is a consortium of leading food production companies and research organizations.