



D4.03: Second TaaS Prototype
 Grant Agreement No.: 732638
 Call: H2020-ICT-2016-2017
 Topic: ICT-13-2016
 Type of action: RIA



D4.03: Second TaaS Prototype

Work package	WP 4
Task	Task 4.3
Due date	31/12/2019
Submission date	18/02/2022
Deliverable lead	TUB
Version	1.0
Authors	Nitesh Pandey (TUB), Alexander Willner (Fraunhofer), Gabriele Cerfoglio (Martel), Albert Yiu Quan Su (Sorbonne Université), Donatos Stavropoulos (CERTH)...
Reviewers	Alexander Willner (Fraunhofer)

Abstract	This deliverable describes the second prototype with the one-stop-shop with matchmaking capabilities, including the interactions between several components. A manual describes how customers and testbed providers can interact with the prototype.
Keywords	One-Stop-Shop, matchmaking, integration, testbed provider, spread-activation.



DISCLAIMER

The information, documentation and figures available in this deliverable are written by the Federation for FIRE Plus (Fed4FIRE+); project's consortium under EC grant agreement 732638 and do not necessarily reflect the views of the European Commission.

The European Commission is not liable for any use that may be made of the information contained herein.

COPYRIGHT NOTICE

© 2017-2021 Fed4FIRE+ Consortium

ACKNOWLEDGMENT



Co-funded by the
European Union



Co-funded by the
Swiss Confederation

This deliverable has been written in the context of a Horizon 2020 European research project, which is co-funded by the European Commission and the Swiss State Secretariat for Education, Research and Innovation. The opinions expressed and arguments employed do not engage the supporting parties.

Project co-funded by the European Commission in the H2020 Programme		
Nature of the deliverable:		R
Dissemination Level		
PU	Public, fully open, e.g. web	X
CL	Classified, information as referred to in Commission Decision 2001/844/EC	
CO	Confidential to FED4FIRE+ project and Commission Services	

- R: Document, report (excluding the periodic and final reports)
- DEM: Demonstrator, pilot, prototype, plan designs
- DEC: Websites, patents filing, press & media actions, videos, etc.
- OTHER: Software, technical diagram, etc.



EXECUTIVE SUMMARY

The most valuable assets testbeds have are actual knowledge and equipment. Therefore, the main objective of this deliverable is to describe a “Testbed as a Service (TaaS)” broker that allows potential customers to book added-value services that might be offered in a testbed in a federation. Such a one-stop-shop can be used to search, find and book a best-matched testbed according to the customer requirements. This might allow new revenue streams to be created.

This deliverable analyzes and describes the needed capabilities that such a system should have in this project. This analysis includes the tools and technologies used to implement the system and provides an overview of a proposed matchmaking algorithm. The implementation is based on the gap analysis conducted in deliverable D4.01 [1]. Therefore, an overview of the deliverable D4.01 will be discussed in brief in the next section of this deliverable.

To sum up, an introduction to the individual components used for building a TaaS broker is given, followed by a description of their interactions and a manual on how to use the TaaS broker is given. We conclude with an outlook.



TABLE OF CONTENTS

Disclaimer 2

Copyright notice 2

Acknowledgment..... 2

1 INTRODUCTION..... 8

1.2 OBJECTIVES 8

1.3 SUMMARY OF THE FIRST DELIVERABLE D4.01 8

1.4 CONCLUSION 9

2 DESIGN 10

2.1 SELECTION OF TOOLS 10

2.1.1 MySlice 10

2.1.2 Odo 12

2.1.3 Odo Implementation Details 14

2.2 MARKETPLACE..... 16

2.3 ARCHITECTURE DESCRIPTION 19

2.4 DESCRIPTION OF MATCHMAKING CAPABILITIES..... 20

2.4.1 Gensim Doc2vec algorithm 20

2.4.2 Model Training 22

2.4.3 Model Training Hyperparameters 23

3 MANUAL..... 24

3.1 SERVICE PUBLICATION..... 24

3.1.1 Register the Testbed Provider 24

3.1.2 Register the Services 26

3.2 SERVICE SELECTION 28

3.2.1 Query Demand..... 29

3.2.2 Structured Search 30

3.2.3 Selection Summary 32

3.3 LEAD CREATION 34

3.4 EXECUTION 39

3.4.1 Feedback between Customer and Vendor..... 39

3.4.2 Customer Accepts the Quotation..... 40

3.5 CONCLUSION 41

3.5.1 The Sales Order Becomes an Invoice 41

3.5.2 Invoice and a Confirmation of Customer’s Payment 42





3.5.3	Evaluation	43
4	CONCLUSION/FUTURE WORK.....	44
4.1	MARKETPLACE.....	44
4.2	MATCHMAKING	44
4.3	MYSLICE	46
4.4	ODOO	46
4.5	Pricing.....	46
5	WORKS CITED.....	47



LIST OF FIGURES

Figure 1: The initial Integration Plan for the Prototype.	9
Figure 2: The initial Integration Plan for the Prototype.	10
Figure 3: Configuration of an Incoming Mail Server on Odoo.	14
Figure 4: Configuration of Sales Channel.	15
Figure 5: Current Start Page (after login) of the Marketplace Component.	16
Figure 6: Structure and Data Flow in Flux [5].	17
Figure 7: Structure and Data Flow of the Marketplace Plug-in.	17
Figure 8: Architecture with all Components.	19
Figure 9: Paragraph Vector - Distributed Memory [11]	21
Figure 10: Paragraph Vector - Distributed Bag of Words [11]	21
Figure 11: Model Training Workflow	22
Figure 12: Registration of the FUSECO Playground Testbeds	25
Figure 13: Registering the Services of FUSECO Playground	26
Figure 14: Creating/Adding a new service	27
Figure 15: Overview of Activities for Service Publication.	28
Figure 16: Example of the Service Selection.	29
Figure 17: Searching services for testing by using matchmaking.	30
Figure 18: Searching for testbeds' resources	31
Figure 19: Login to the market using IMEC accounts	32
Figure 20: Request further Information after the Selection Phase.	33
Figure 21: Overview of Activities for Lead Creation.	34
Figure 22: Odoo notification via email	35
Figure 23: Odoo Log-in Page.	35
Figure 24: Vendor message page	36
Figure 25: Odoo Backend – Vendor's New Quotation View.	36
Figure 26	37
Figure 27	37
Figure 28: Odoo Backend – Vendor's Email View.	38
Figure 29: Odoo Frontend – Customer's Welcome Page.	38
Figure 30: Overview of Activities for Execution.	39
Figure 31: Odoo Frontend – Email & Message Section.	39
Figure 32: Odoo Frontend - Payment Confirmation View.	40
Figure 33: Overview of Activities for Conclusion.	41
Figure 34: Odoo Backend – Vendor's Sales Order View.	41
Figure 35: Odoo Backend – Vendor's Invoice View.	42
Figure 36: Odoo Frontend – Customer's Payment List Page.	42
Figure 37: Odoo Frontend – Ontology based Structured Search Engine.	45



ABBREVIATIONS

BSS	Business Support System
CRM	Customer Relationship Management
eTOM	Enhanced Telecom Operations Map
FanTaaStic	Testbeds-as-a-Service for the EIT ICT Labs
FedSM	Federated IT Service Management
FIRE	Future Internet Research and Experimentation
IIoT	Industrial Internet of Things
KPI	Key Performance Indicator
MVC	Model View Controller
NLP	Natural Language Processing
OSS	Operations Support System
QoS	Quality of Service
RFQ	Request For Quote
SFA	Slice Federation Architecture
SLA	Service Level Agreements
OPC UA	Open Platform Communication Unified Architecture
TaaS	Testbed as a Service
TSN	Time Sensitive Networking



1 INTRODUCTION

1.1 OBJECTIVES

In a number of conducted studies, the concept “Testbed as a Service” has been covered before. Some work suggests to provide brokering capabilities, in order to allow resource providers to offer different kinds of services in federated infrastructures. For instance, FanTaaStic was a project with the EIT ICT Labs initiative, that aimed at brokering services towards large-scale European experimental facilities. Based on similar concepts, within WP4 aims at building a sustainable marketplace, that is, both allowing testbed providers to offer their own services, and allowing the customers to find and book them easily. As a result, testbed providers might be able to create an additional source of revenue. The objectives can be summarised as follows: allowing new revenue streams, creating an open, self-sustainable marketplace for services, and focus on intelligent matchmaking and allow on-demand booking.

This deliverable introduces a prototype that implements these points and thus makes it possible to offer the testbeds as a service. The starting point of this report is the gap analysis that emerged from deliverable D4.1 [1]. The gap analysis showed the lessons learned from similar projects and it’s also presented various tools with which the implementation can be done.

The remainder of this document is structured as follows. Since the results of the analysis have a great impact on this report and on TaaS, it will be listed here again. The design section describes the tools that make up the prototype and how they interact with one another. The manual section describes how the customer, the vendor and the federation can use the TaaS broker. Finally, there is a brief summary and outlook for each individual component and for the TaaS Prototype.

1.2 SUMMARY OF THE FIRST DELIVERABLE D4.01

The gap analysis from deliverable D4.1 showed, on the basis of previous or similar projects, what needs to be considered and what experiences can be taken from them. This included identifying some user groups, functions that must be made available by the system, and what the initial plan for implementation might look like. Figure 1 shows this initial plan.

As can be seen in the figure, the idea was to extend the existing systems with new components so that they meet the new requirements. The Figure 1 also shows the connection to the tasks in Work Package 3, where each testbed has an application with which it can load its services into the Semantic Directory. This in turn enables users to search for them via the TaaS portal. In addition, it should be possible to import data from existing databases of the testbeds. The Semantic Directory is responsible for processing the uploaded data. It provides the appropriate results on the user requests and should be able to integrate all existing data, such as registered users or testbeds, like the applications for the testbeds.

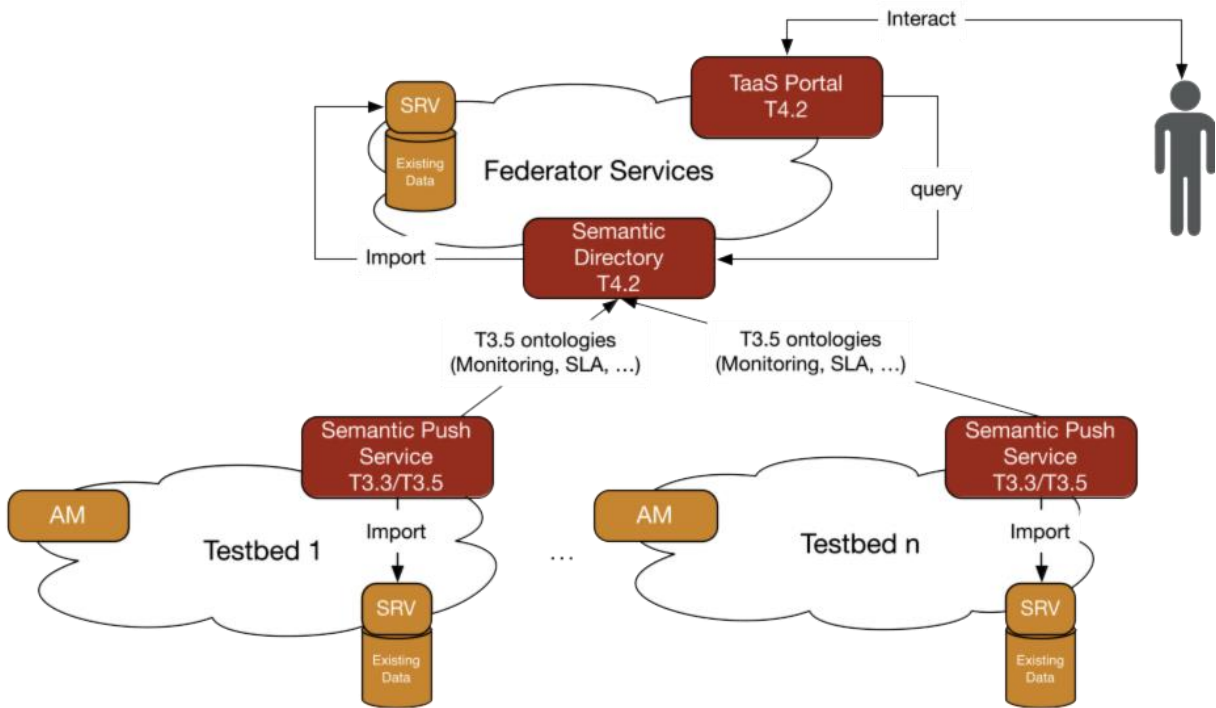


Figure 1: The initial Integration Plan for the Prototype.

1.3 CONCLUSION

Hence, this deliverable presents a prototype and implementation of the Task T4.2. Using D4.01 as starting point, the first prototype has been developed. The extent to which the initial plan was implemented is examined and explained in more detail in the Design section. In the following sections we are going to present the **Design** concept in detail, the **Manual** with a detailed description on the usage of the One-Stop-Shop system, the **Conclusion** with the entire deliverable in a nutshell and the **Future Works** with some possible developments and chances in order to make the TaaS broker more efficient.

2 DESIGN

This chapter deals in detail with the selected tools which will be discussed in the next sections. It includes a detailed description and its purpose. It also describes how the tools are connected in order to build the architecture of the first prototype. At the end, the matchmaking system will be presented.

2.1 SELECTION OF TOOLS

Before considering the prototype, all components are introduced. Each of them has its own task, which takes care of the relationship between customers and testbed providers (vendors). In addition, the architecture of the components are examined in more detail and certain design decisions are explained clearly.

2.1.1 MySlice

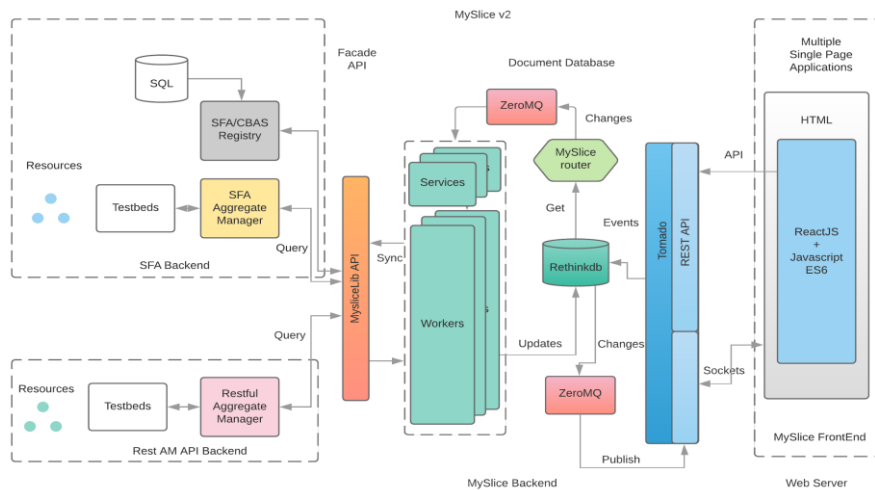


Figure 2: The initial Integration Plan for the Prototype.

The goal of the GUI component is to provide a simple tool to use, low barrier entry point to allow customers to discover available offerings (technical and non-technical). While the initial discovery phase should not require any prerequisite from the user (e.g., only a modern web browser or client for RESTful services), a handover to an authenticated service request is envisioned. One existing tool that can be reused and extended is MySlice, which has already been developed and used within a number of FIRE projects. It is envisioned to extend it

towards the specific requirements for WP4. Its architecture is depicted in Figure 2 and services are responsible for gathering data from the distributed sources of the architecture. The Web frontend interacts with a document database (RethinkDB), which is used as a caching system. It is also used to store data specific to the frontend. This real-time database offers the possibility to the services of the MySlice API to subscribe to events. As soon as the web frontend updates a record, the services are notified and can trigger events. This architecture allows decoupling the frontend from the complexity of processing results from distributed data sources (AMs, Registry).



This modular architecture allows it to be extended on both sides either to support new services or to enhance the web frontend. In the Fed4FIRE+ context it will be extended with matchmaking functionality and allow users to reserve not only Fed4FIRE+ specific physical/virtual resources but also to match specific user requests related to non-technical resources.

MySlice also provides the functionality to register and manage users. The communication with the testbeds is ensured using the SFA AM API or a Restful AM API. The communication with the Registry relies on the CBAS Registry API.

MySlice also provides support for different authentication mechanisms and is extended to support the Oauth module.

In the following subsections we are providing information about the most important MySlice components.

2.1.1.1 SFA/CBAS Registry

This module provides a registry service as specified by the SFA. It can be queried by the MySlice core as well as by some remote registries and provides information about the SFA objects which this domain is responsible for. MySlice implemented to work with the CBAS Registry is an extended version of the previous SFA Registry while still providing support for the latter.

The CBAS registry provides isolation in the management of the users and slices by integrating a Member and a Slice authority respectively, it also implements all of the GENI API calls that are missing in the SFA Registry, hence providing an efficient way to manage different entities in the federation.

2.1.1.2 MySliceLib

This module supports the interaction of MySlice with multiple Aggregate Managers that are part of a federation of testbeds. One of the key features is to transform users' requests into testbed-specific API calls, hence allowing MySlice to communicate with different Aggregate Managers using different protocols (XML RPC or Restful API). This module is used to send management, allocation and provisioning requests and to parse the results into user-friendly data.

2.1.1.3 MySlice Backend

This module supports all the advanced functionality provided by MySlice, such as, the distributed management of queries, asynchronous handling of the queries, caching of the results. To enable the asynchronous handling and distributed management of queries, a query is defined as an event and each event is handled by a specific service depending on the type of the events.

The modularity of MySlice facilitates the integration of new services to handle new types of events. The new services can be integrated as "add-ons" which can be enabled or disabled. In addition to this, MySlice provides the possibility to scale up the services, or also to run MySlice as a distributed system.

2.1.1.4 MySlice Frontend

This module is in charge of the frontend that exposes the underlying MySlice functionality to various users/experimenters. The frontend is built with Javascript ES6 and the Javascript framework React. The key of the frontend is that every object in the frontend is defined as a component that is reusable, besides it also facilitates the integration of new components.

The marketplace is implemented as a plug-in component to the MySlice GUI.

2.1.1.5 System Descriptions

The different components of MySlice and the registry run on different servers hosted and managed by SU (Sorbonne Université). The servers use a GNU/Linux Ubuntu distribution as an operating system. The distributed system is enabled by the modularity of the Myslice software.

MySlice and its components are built and developed in Python 3 language for its backend. And for the frontend and RESTful API, the Python web framework “Tornado”. A web framework is a support to develop web applications or web services.

2.1.2 Odoo

The Fed4Fire+ Federation requirements are:

- ➔ Customer Quotation Management.
- ➔ Customer Billing Management.
- ➔ Customer Helpdesk Management.

A common method to manage a company's interaction with current and potential customers is to use a Customer Relationship Management (CRM) system. Such system compiles data from a range of different communication channels, including a company's website, telephone, email, live chat, marketing materials, and more recently, social media. Thanks to CRM systems, businesses learn more about their target audiences and how to best cater to their needs.

There are several Open Source CRMs, such as

- ➔ vTiger (<https://www.vtiger.com>)
- ➔ SugarCRM (<https://www.sugarcrm.com>)
- ➔ Odoo (<https://www.odoo.com>)

Odoo was the choice within WP4 for three interesting features:

- ➔ Customer Billing Management and Customer Helpdesk Management APIs. These APIs make the integration between Odoo, MySlice and the Marketplace faster and simpler. For example, a customer (the user) can create a request for a quotation from the Fed4FIRE catalogue quickly and easily as an online purchase. Under the hood, Odoo creates a lead, sends the request to a vendor/Federation (TestBed provider) and comes back to the customer with the vendor's response, via message or email. Customers and vendors do not need to install any software or app on their computers.
- ➔ Multi-company functionality. Both Federation and vendors can manage billing, accounting and so forth directly. They only need an Odoo account, or alternatively an account from an external Identity Provider, thanks to Odoo's OAuth2 support. The software will be the same for all of them.
- ➔ Online payment services. PayPal, Stripes and other electronic payment methods are available in order to simplify and speed up the business between the customers and the vendors

Furthermore, Odoo is an all-in-one management software that offers a range of business applications that form a complete suite of enterprise management applications targeting companies of all sizes. Odoo is an all-in-one business software which includes accounting, billing, CRM, website/e-commerce, manufacturing, warehouse and inventory.

2.1.2.1 System Description

Odoo runs on a server hosted by SU (Sorbonne Université). The server uses a GNU/Linux Ubuntu distribution as an operating system.

There are two different editions of Odoo: Community and Enterprise versions. The Community is a free and open source version, the Enterprise extends the Community version with commercial features and paid services. Currently, we use the Community version.

Odoo is a combination of a frontend and backend portion. The frontend is where the User Interface that customers and vendors will interact with is served. The homepage, the log-in and sign-up pages all belong to the frontend for example. This is how the customers communicate with the vendors (e.g. the testbed providers and Federation) or check their bookings, their bills, their payments etc.

The backend part is aimed only at the vendors and the Federation. From the backend they have access to Odoo's API to integrate their own services, as well as a dashboard where they can manage their own business such as customer messages, requests for quotation (RFQ), invoices etc.

Odoo is built and developed in the Python language, with its UI also making use of Javascript (jQuery) and the standard styling of web pages with HTML and CSS.

A major advantage of Odoo is the modules, formerly just known as "add-ons". In this way, it is possible to add and remove features without editing the core of Odoo. Everyone can develop some modules or get them from a dedicated place where all are collected [7]. The modules are free or paid.

One very useful module that comes by default with the latest versions of Odoo enables Oauth2 support. This allows users to log into Odoo using an external Identity Provider, which include the commonly used ones like Google, but also any arbitrary Oauth2 compliant IdP, like the one provided by imec.

2.1.2.2 Dataset Description

For our simulation we need three things:

- ➔ A customer email address is required for three reasons:
 - for the customer to log into Odoo (frontend).
 - for receiving all Odoo notifications.
 - for a direct communication between the customer and the vendor

- ➔ A vendor email address is required for three reasons:
 - for the vendor to access Odoo (backend).
 - for receiving all Odoo notifications.
 - for a direct communication between the customer and the vendor.

- ➔ A list of vendors' services and products:
 - Marketplace/MySlice (market.fed4fire.eu) and Odoo (crm.fed4fire.eu) are synchronized through Odoo's native backend API (see External API section for further details). So, when a service or a product is created, updated or deleted in Odoo, it will be created, updated or deleted in the Marketplace/MySlice automatically.

2.1.3 Odoo Implementation Details

2.1.3.1 Details of the Simulation Environment

Odoo runs on a server, provides access through a web interface, accessible via any web browser. The login page can be found at: crm.fed4fire.eu. After logging in, both customers and vendors can access their accounts and start their doing their work. No specific software, tools or devices are required, just a computer with a web browser (Firefox, Chrome, Internet Explorer etc.) and an internet connection.

2.1.3.2 Integration

2.1.3.2.1 External API

Odoo provides its own native Application Programming Interface (API). From Odoo’s own documentation:

“Odoo is usually extended internally via modules, but many of its features and all of its data are also available from the outside for external analysis or integration with various tools. Part of the Model Reference [8] API is easily available over XML-RPC [9] and accessible in a variety of languages (Python, Ruby, PHP, Java – author’s note).”

It is not the goal of this document to explain the API in detail, but it is worth noting:

- ➔ It is usable either from a written program or through a command line interface (CLI)
- ➔ It is accessible from a variety of languages, independently of the OS or the browser, so the compatibility with other systems is granted.
- ➔ The response of the API is compatible with almost all languages/software because the API uses a JSON data format that grants universal compatibility.

2.1.3.2.2 Opportunistic Odoo Settings

Odoo can be set to send an email notification to an associated email address when it receives one for a given account. It is also possible to associate different email addresses with different email accounts. In our case, we use the first scenario. We will briefly describe how that is set up in Odoo.

The first step is to set the Incoming Email Server with all required parameters (Figure 3).

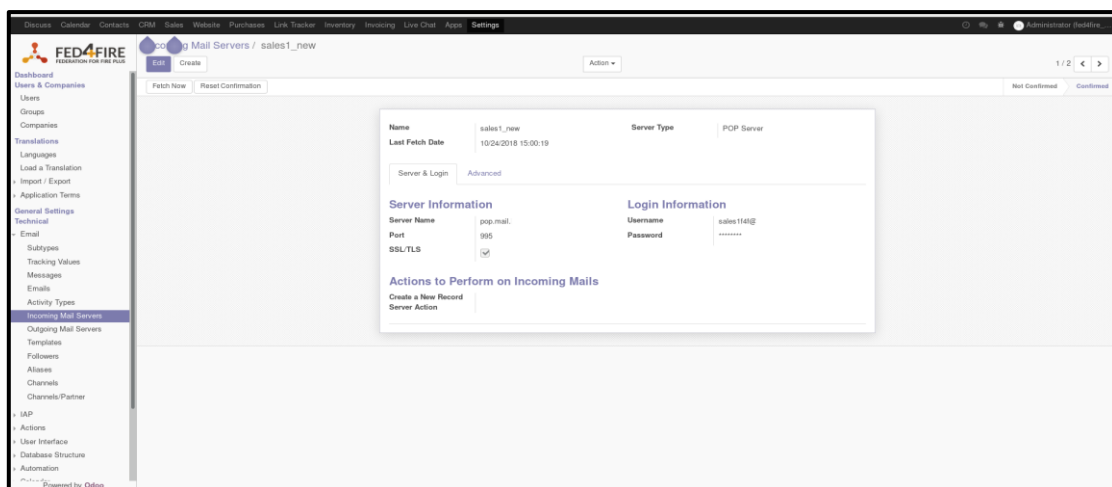


Figure 3: Configuration of an Incoming Mail Server on Odoo.

The second step is to associate the vendor’s email address with the email address of the incoming email server (Figure 4).

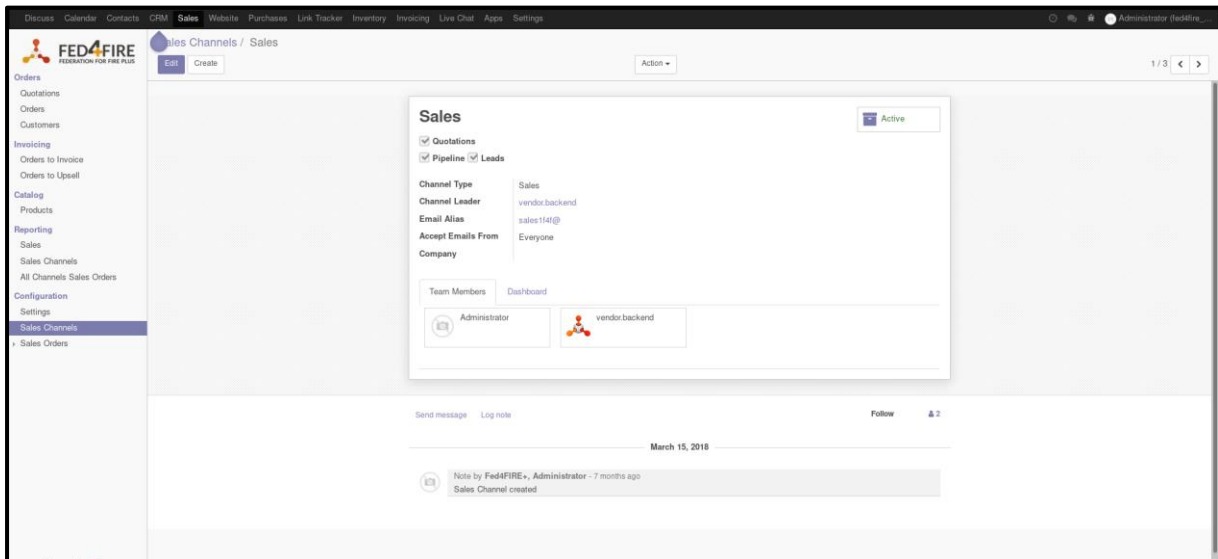


Figure 4: Configuration of Sales Channel.

The third step is optional, but we can add other followers, meaning other people (contacts) available to Odoo, regardless of whether they are customers, vendors etc. But with this, confirmation emails for example will be automatically sent out upon receiving a request from a client.

2.1.3.2.3 Oauth2 support for external Identity Providers

As previously mentioned, Odoo provides Oauth2 support thanks to an included module. This allows for the use of an external Identity Provider, and thus allowing users to log into Odoo using an external account. An example of this would be how a Google account nowadays can be used to log into several websites, without the need to register a new account for every website.

With our current Odoo deployment, users log in using an account they register with iMec. Upon landing on Odoo’s login page, users click the login button, are redirected to iMec’s login page, and upon logging in there are then brought back to Odoo, fully logged in. Their profile details are carried over with them as well.

2.2 MARKETPLACE

In comparison to the other components presented, which are reused, the Marketplace component was newly developed for this project. The Marketplace component offers on-demand booking for the customers. There are two main features enabled with the implementation of this component. The first one is to enable testbed providers to register themselves and their services and the second one is to enable customers to search and book services.



Figure 5: Current Start Page (after login) of the Marketplace Component.

The Marketplace component is developed as a plug-in for MySlice, because it also uses the framework ReactJS for the frontend part and therefore it is integrated with MySlice. Figure 5 shows the current version of the Marketplace component integrated in MySlice.

FireRabbit is the current internal project name but will be renamed to Marketplace for the next prototype.

For the design of the pages, we use Google's Material Design [3]. Material-UI [2] implements this design for React components and is available as open source software. That is the reason why it is also used in the development of the Marketplace plug-in.

As already mentioned, the Marketplace plug-in was developed with the React framework. React is a JavaScript framework developed by Facebook's developers to build single-page or mobile applications. A single-page application loads all the necessary components at the beginning as a single load and dynamically rewrites the page and changes the views upon user's interaction. Single-page applications are meant to improve the user experience, while reducing the successive page loading and reloading.

React provides a template language that allows you to build simple views and also implements a mechanism that efficiently updates the views as changes occur.

The framework allows developers to define the remaining technology stack themselves, as it does not prescribe any dependencies in this respect. Looking at the MVC model, this means that React provides the view part but not the model or controller. For the missing components, a pattern is used that also comes from the Facebook's developers. It is called Flux [5] and does not follow the MVC model to allow unidirectional data flow.

Figure 6 shows the structure and the data flow that run through the components.

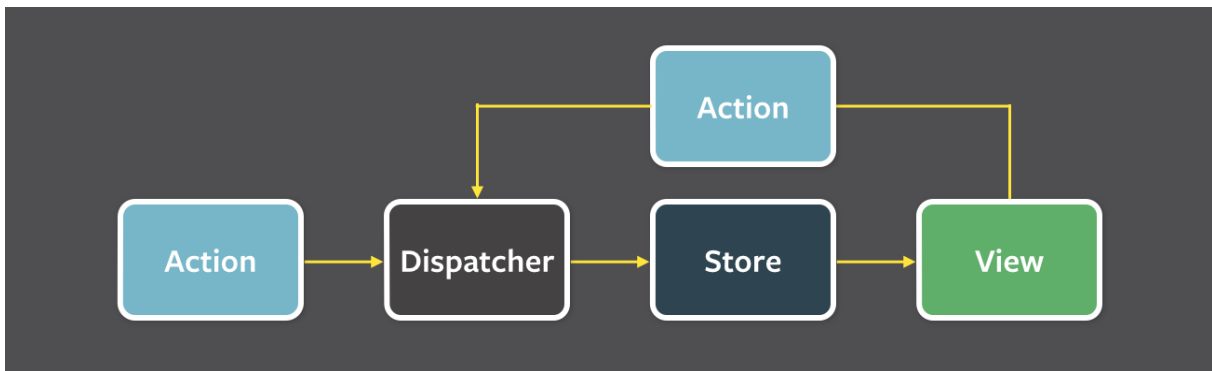


Figure 6: Structure and Data Flow in Flux [5].

Unidirectional data flow means that when a user interacts with a view, it transmits an action to the store through a central dispatcher. Stores hold data of the application and update the corresponding views in the event of a change.

The structure for the Marketplace plug-in is as follows:

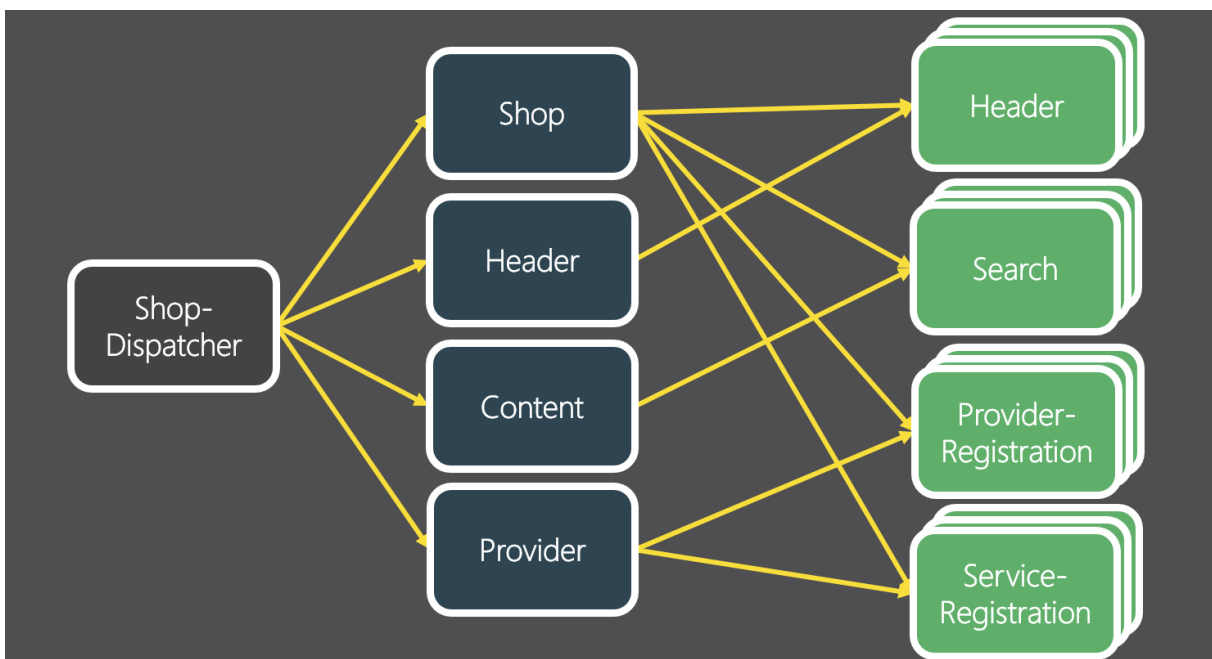


Figure 7: Structure and Data Flow of the Marketplace Plug-in.

The colour coding corresponds to the previous figure. The actions have been removed from the image to keep good readability. Since each flux-based application should only have one dispatcher, it also has only the shop dispatcher that forwards all actions to the stores. There are currently four stores that manage the states of different parts of the application. The shop “store” is an exception, as it manages data that affects the entire application. For example, the current theme or loaded configuration file.

If the theme changes, all views are notified and re-rendered. The other stores are limited to individual parts, such as registering the testbed providers and their services. As with the stores, there are several views for one area. The header view contains the logo and the menu from which

D4.03: Second TaaS Prototype



the other pages can be accessed. Search view contains the text field for the search and all subsequent components, such as the list of results and the summary of the booking.

In the first TaaS prototype, the Marketplace is using a simple keyword searching tool to compute the matchmaking result. However, for the second TaaS prototype, a new component is implemented to handle more complex matchmaking queries. This new component, described in section 2.4, is implemented by TUB and integrated with MySlice. The matchmaking component is deployed as a Docker Container and hosted by SU. It exposes a Restful API, hence the Marketplace can query this matchmaking component and retrieve complex search results. The Marketplace frontend is adapted to display the results to the users.



2.3 ARCHITECTURE DESCRIPTION

After the subcomponents of the prototype have been presented, the architecture of the three components is shown in this section.

Figure 8 shows how the different user groups interact with the components and how the components themselves interact with one other. The Marketplace plug-in is part of the MySlice component and Fed4Fire+ Offerings represent the Knowledge Graph.

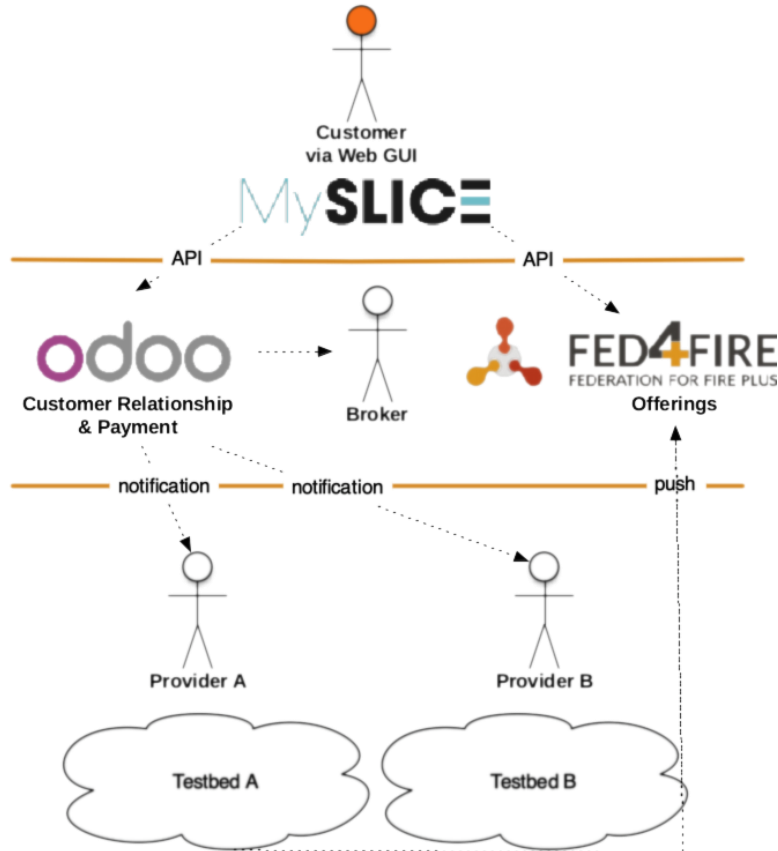


Figure 8: Architecture with all Components.

2.4 DESCRIPTION OF MATCHMAKING CAPABILITIES

For the second prototype, a Natural Language Processing approach is adopted. Natural language processing (NLP) received a lot of attention from academia and industry over the recent decade, benefiting from the introduction of new algorithms for processing the vast corpora of digitized text. A set of language modeling and feature learning techniques called word embeddings became increasingly popular for NLP tasks. The NLP approaches also have been taken place in the search engine domain to understand the user query and provide an answer based on the query. Word2vec [10] became one of the most famous algorithms for word embeddings, offering a numeric representation of any word, followed by doc2vec [11], which performed the same task for a paragraph or document.

The doc2vec algorithm is applied to build sophisticated search engine for distributed tested and services. There are several algorithms which can be applied to build the domain-specific search engine. Later, the evaluation will take place which algorithms are suitable for domain-specific search engine.

2.4.1 Gensim Doc2vec algorithm

Numeric representation of text documents is a challenging task in machine learning. Such a representation may be used for many purposes, for example, document retrieval, web search, spam filtering, topic modelling, etc. However, there are not many good techniques to do this. Many tasks use the well-known but simplistic method of a bag of words (BOW), but the outcomes will be mostly mediocre since BOW loses many subtleties of a possible good representation, e.g. consideration of word order. but the Do2vec algorithm can handle these problems easily.

The goal of doc2vec is to create a numeric representation of a document, regardless of its length. But unlike words, documents do not come in logical structures such as words, so another method has to be found. The concept that Mikilov and Le have [10, 11] used was simple, yet clever: they have used the word2vec model, and added another vector (Paragraph ID below) and solved the problem.

Paragraph Vector - Distributed Memory (PV-DM)

This is the Paragraph Vector model analogous to Word2Vec CBOW. The doc-vectors are obtained by training a neural network on the synthetic task of predicting the center word based an average of both context word-vectors and the full document's doc-vector.

- Assign and randomly initialize paragraph vector for each doc
- Predict next word using context words + paragraph vector
- Slide context window across doc but keep paragraph vector fixed
- Updating is done via SGD and backprop.

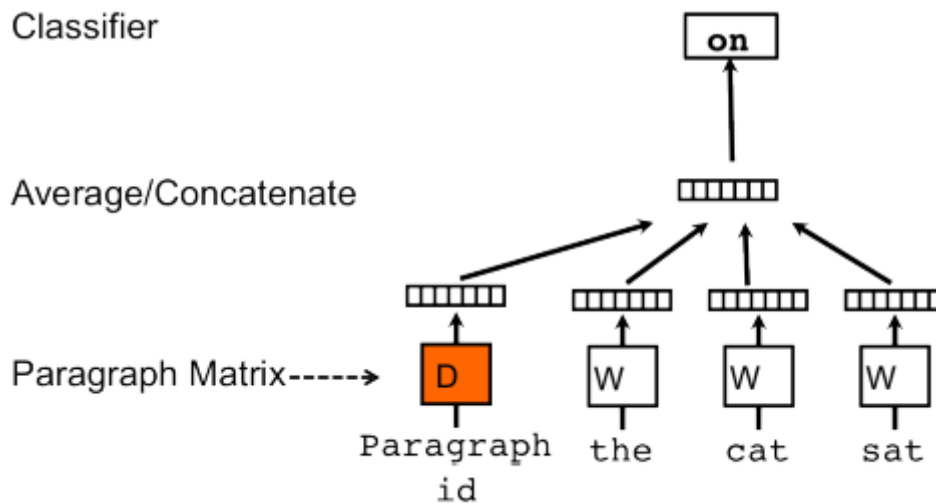


Figure 9: Paragraph Vector - Distributed Memory [11]

Paragraph Vector - Distributed Bag of Words (PV-DBOW)

This is the Paragraph Vector model analogous to Word2Vec SG. The doc-vectors are obtained by training a neural network on the synthetic task of predicting a target word just from the full document's doc-vector. It is also common to combine this with skip-gram testing, using both the doc-vector and nearby word-vectors to predict a single target word, but only one at a time.

- Only use paragraph vector (No word vectors).
- Take window of words in paragraph and randomly sample which one to predict using paragraph vector (Ignores word ordering)
- Simpler and more memory efficient.
- DM typically outperforms DBOW (but Dm + DBOW is even better)

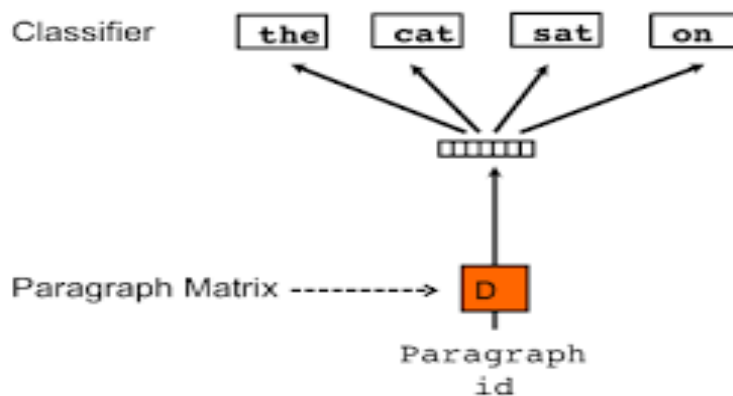


Figure 10: Paragraph Vector - Distributed Bag of Words [11]



2.4.2 Model Training

- Prepare training data for domain related}
- Apply preprocessing (such as remove stop-words, remove numbers, removing punctuation, accent marks and other diacritics, removing white spaces, expanding abbreviations, Lemmatization etc.
- Apply Gensim doc2vec model on preprocessed data
- Train the model with hyperparameter

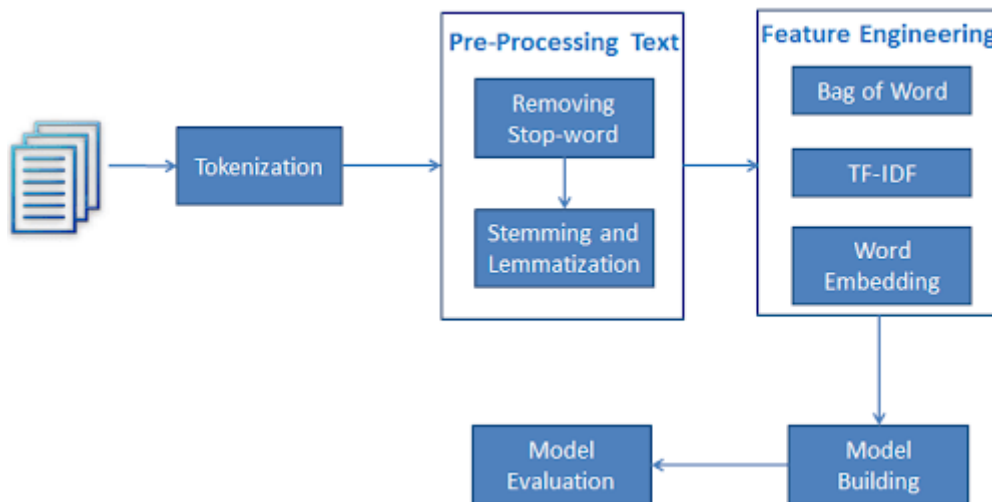


Figure 11: Model Training Workflow

2.4.3 Model Training Hyperparameters

Hyperparameters Name	Values
vector_size	20000
dm	1
Window	2
alpha	0.025
seed	47
min_alpha	0.0001
min_count	5
sampling_threshold	1e-5
worker_count	1
iter	200

As you can see, the user first enters his query in the form of text. Using a Natural language processing based search engine, all entries related to the queries are listed. The result of the query based on the NLP model, which was trained on domain-specific datasets. This learning-based on statistical learning, which uses word embeddings as features.

An algorithm for this technology works through natural language processing and machine learning technology.

Finally, the original and newly found hits are delivered or shown to the user. This approach is thus implemented in the prototypes.

3 MANUAL

This chapter describes the operation of the prototype. The operation is separated into five steps, based on the activities a customer goes through to book services.

- ➔ Service Publication – It allows testbed providers to register themselves and their services.
- ➔ Service Selection – It provides customers the opportunity to query services and to select them out of a list of best matches.
- ➔ Lead Creation (Feedback) – it notifies testbed providers about requests and allows both sides (customer and vendor) to communicate with one other.
- ➔ Execution (Bilateral incl. Legal and Financial Aspects) – It includes the negotiation between the customer and the testbed provider and the service execution.
- ➔ Conclusion – It covers the invoice creation and the payment.

The following sections describe these steps in detail and with examples. This includes how the different components are communicating with one another and how the customers and testbed providers can interact with the current version of the prototype.

3.1 SERVICE PUBLICATION

((public web sites / knowledge graph))

The testbed owners decide to provide their services to the customers through Fed4Fire. To move forward with this purpose, the sophisticated search engine came to light to offer its services. This search engine provides the searched query information along with the testbed names. For the registration of services, a description of the query, an image, and the type of service are given.

The information is provided through machine learning and the NLP approach. In addition, they are saved in Odoo in order to easily manage the administrative and financial aspects.


The following sections will describe the steps in order to register the testbeds and services by the services provider.


3.1.1 Register the Testbed Provider

The following example shows how to register the testbed provider and its services in the current prototype. The scenario is as follows:

"The Fraunhofer Institute FOKUS would like to register the FUSECO Playground Testbed. Dr. Hermann Schmik will serve as contact person. To begin, they register the services *OPC UA Compliance Test* and *IIoT Consultant Alexander Willner*."

In the first step, the testbed provider will be registered. To do this, go to the Register Provider page from the menu previously presented in Figure 6.


FIRERabbit



Register your testbed

Before you can use the advantages of Fed4FIRE+ we need some information of you. Please make sure that your information are correct. Otherwise it may affect the status of your account.

Company

location type

Germany **Non Profit**

Please select your location Please select your type of business

name:

FUSECO Playground

Representative

title:

Dr.

Please select your title (optional)

<small>First name</small>	<small>Middle name</small>	<small>Surname(s)</small>
Hermann		Schmik
<small>(optional)</small>	<small>(optional)</small>	<small>(optional)</small>

REGISTER

Version 0.4.6

Figure 12: Registration of the FUSECO Playground Testbeds

Figure 12 shows the page with the information from the example above. After entering information and clicking on the register button, information is distributed in the prototype.



3.1.2 Register the Services

In the second step, the services are registered. To do this, go to the *Register Service* page.



The screenshot shows the FIRERabbit interface for registering services. At the top, there is a logo for FIRERabbit and a hamburger menu icon. Below the logo, the text "Register your services" is displayed. There are three service entries, each with a plus icon on the left and a dropdown arrow on the right:

- OPC UA Compliance Test
- IIoT consultant Alexander Willner
- ADD SERVICE

Below the service entries is a "REGISTER" button. At the bottom of the page, the text "Version 0.4.6" is displayed.

Figure 13: Registering the Services of FUSECO Playground

As can be seen in Figure 13, the service *OPC UA Compliance Test* and *IIoT Consultant Alexander Willner* were entered for this testbed. information is integrated into the system after clicking on Register, as with registering the testbed itself.

With these two steps, both the FUSECO Playground Testbed and its two services are added to the system and can be found and booked via the search engine using proper matchmaking, which leads to the next step in the manual.

As can be seen in Figure 14, we can add new services. For example, from the above Figure 13, OPC UA Compliance test is an application/service whereas, **IIoT Consultant Alexander Willner** is both a consultant/training provider and a service. So, if the testbed provider wants to introduce a new service, they can add it by filling the form and simply clicking **ADD SERVICE**. A new service added to the One-Stop-Shop can be registered later. Therefore, the service will also be available in the search summary.

Register your services

+ OPC UA Compliance Test ▼

+ IIoT Consultant Alexander Willner ▼

+ Add a Service ▲

Service Title

Service Title
(optional)

Consultant Title ▼
(optional)

First name <input type="text"/> <small>(optional)</small>	Middle name <input type="text"/> <small>(optional)</small>	Surname(s)/Last name <input type="text"/> <small>(optional)</small>
<input type="button" value="Choose File"/> No file chosen	<input type="button" value="Upload Image"/>	

Service Description

Service Description
(optional)

Else upload your description here:

Service Pricing

Price <input type="text"/> <small>(optional)</small>	Currency <input type="text"/> ▼ Please select currency
---	--

ADD SERVICE

Figure 14: Creating/Adding a new service



3.2 SERVICE SELECTION

This section will mainly focus on the Service Selection which is the second step to achieve in order to book services. A customer can use the unstructured search engine to find services that match their needs. The customer interacts with this engine to send queries which will be processed in the backend. In this case, processing is handled by the NLP-based matchmaking system described in section 2.4, in order to improve the accuracy of the result. The plug-in searches for all entries that match the query into the Knowledge Graph store and returns them to Marketplace. Figure 15 shows these steps:

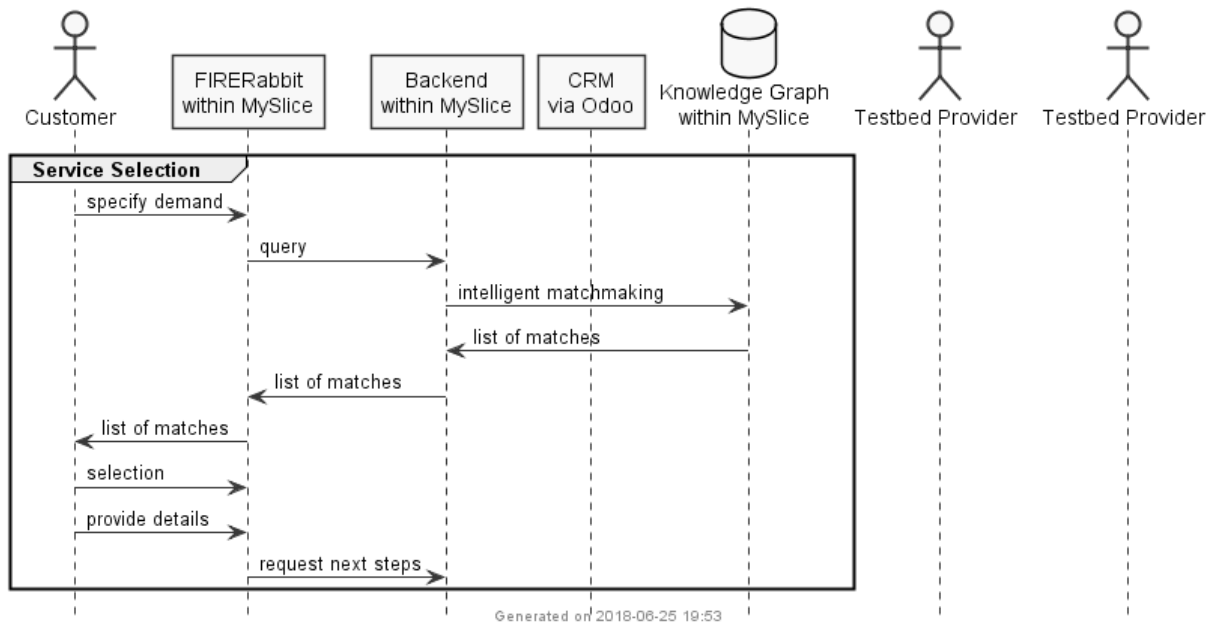


Figure 15: Overview of Activities for Service Publication.

After the matchmaking system has sent the list of matches to Marketplace, the latter processes the list into a human-readable format and displays it in a list. The customer can select the desired services from this list. After the selection, the customer is asked to provide additional information for his booking. This includes his idea or project and his details for any further questions. The reason why the idea should be included is that it allows the selected testbed provider to see what the customer is planning and, if necessary, to give advice, such as selecting another service. Once information has been entered and the selection posted, information is forwarded to the backend.

3.2.1 Query Demand

The following example shows how to use the search engine to send matchmaking requests. Since the focus here is on the customer, the scenario already described is extended as follows:

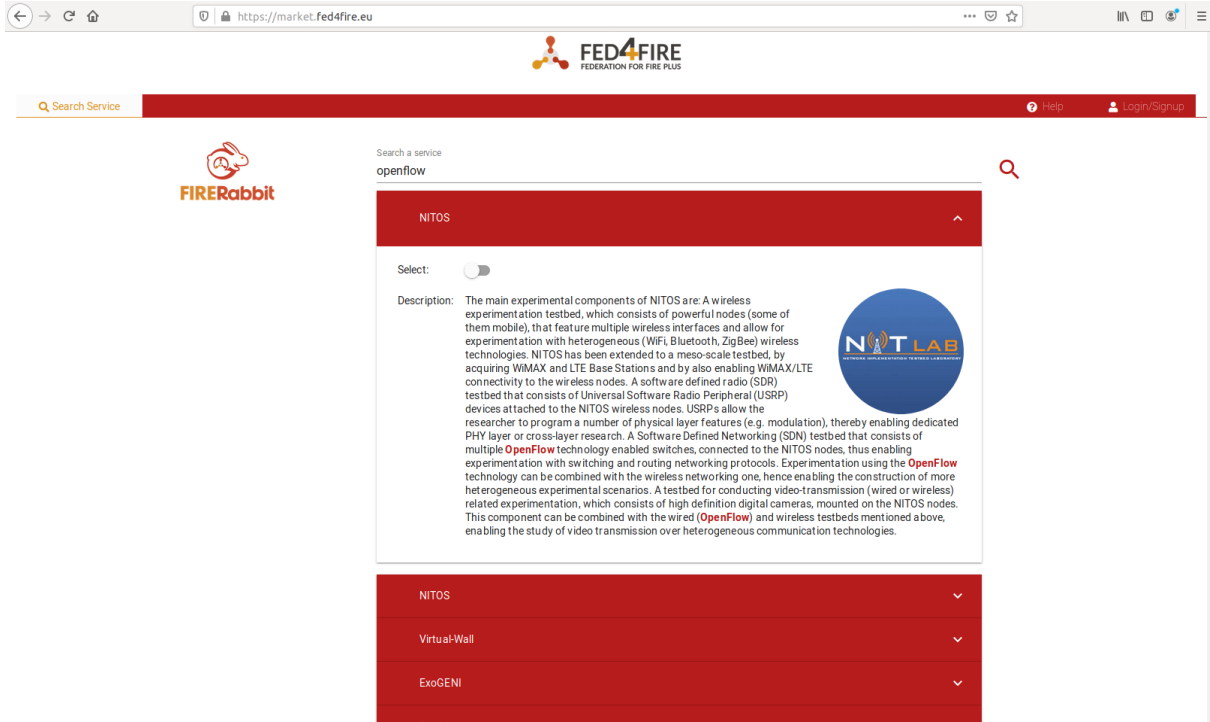


Figure 16: Example of the Service Selection.

First, the company calls up the Marketplace and enters "Openflow" as the search term. The matchmaking system finds five services for the search term that match the term. Figure 16 shows the list with the appropriate services and the company selection.

The Figure 16, summarizes the search and selection, as they differ slightly. As can be seen in the figure, three other services have been listed in addition to the NITOS Playground services that match the search term. As can be seen in the first service, each service has a description and picture in order to describe what its features are.

Another matchmaking capability of the system is the possibility for the user to search sentence based query, for example, he/she can write 'How to access testbeds?' in the search field and can find testbeds access setup facilities. Figure 17 shows a list of testing services offered by the One-Stop-Shop.

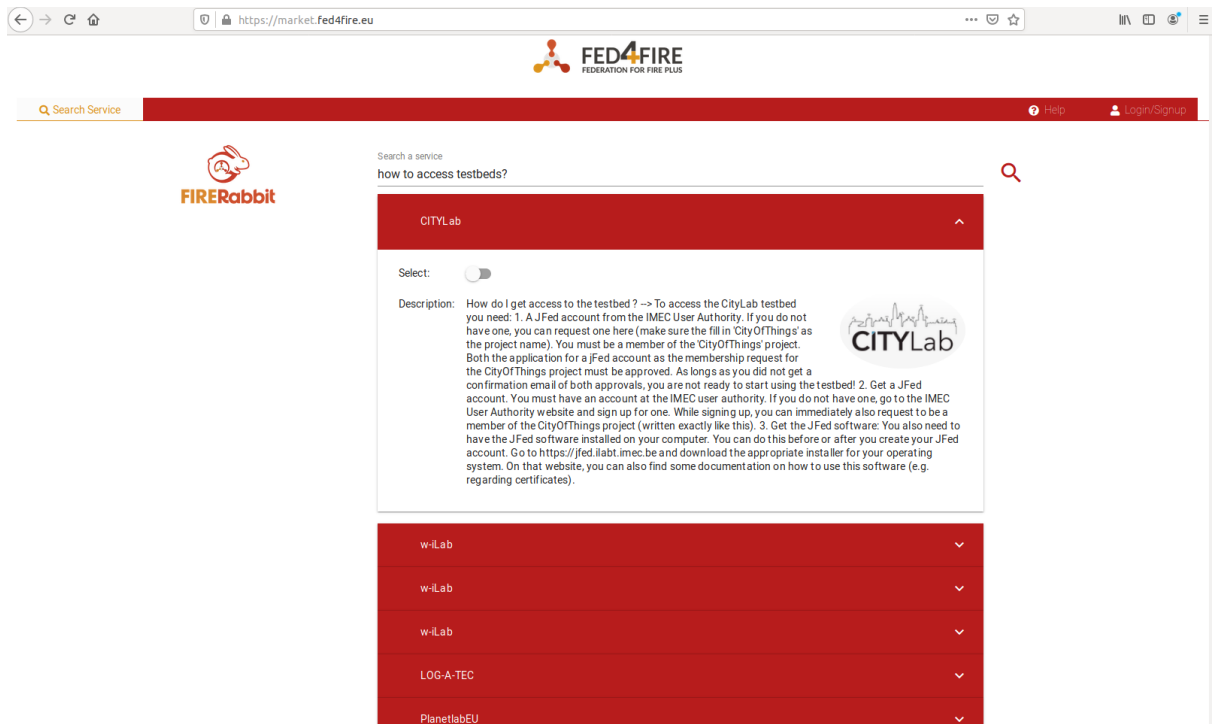


Figure 17: Searching services for testing by using matchmaking.

3.2.2 Structured Search

In addition to the NLP-based matchmaking system, an advanced search engine is implemented for customers to get detailed information of the testbeds’ resources. The Figure 18 describes this engine as a structured search engine.

The following example will describe how to use the engine to find testbeds’ resources that are located in France.

Customers provide to the engine with criteria for the following fields:

- ➔ Availability of testbed
- ➔ Node type
- ➔ Location
- ➔ Country

The query will then be processed in the backend, and the result will be the resources that match the criteria, displayed as lists of dropdown items regrouped by testbeds.

Customers can then select individual resources or the services for the next step described in the following section.



Structured search Unstructured search

Available TestBeds Countries CLEAR FILTER

Grid'5000 ^

Select:

Nodes:

	Node	Available	Testbed	Location
<input type="checkbox"/>	dahu-1	false	Grid'5000	long:5.76705, lat:45.19021, France
<input type="checkbox"/>	dahu-10	false	Grid'5000	long:5.76705, lat:45.19021, France
<input type="checkbox"/>	dahu-11	false	Grid'5000	long:5.76705, lat:45.19021, France
<input type="checkbox"/>	dahu-12	false	Grid'5000	long:5.76705, lat:45.19021, France
<input type="checkbox"/>	dahu-13	false	Grid'5000	long:5.76705, lat:45.19021, France
<input type="checkbox"/>	dahu-14	false	Grid'5000	long:5.76705, lat:45.19021, France

Figure 18: Searching for testbeds' resources



3.2.3 Selection Summary

Each element of the matchmaking results corresponds to a service. The customer can then select the services of interest by toggling the “Select” switch as shown in Figure 19.

After selecting the desired services and clicking on the “Next Steps” button, a summary of the selected services is shown. The customer will then be prompted to fill a form for additional information he/she may want to share, as well as his/her contact information.

Figure 21 shows this summary with some additional information, ready to be sent to the testbed providers.



Figure 19: Login to the market using IMEC accounts

Customers can also log into the market using their IMEC Account prior to the service selection. In this case the contact information will be filled automatically with the information provided to the IMEC Authority.

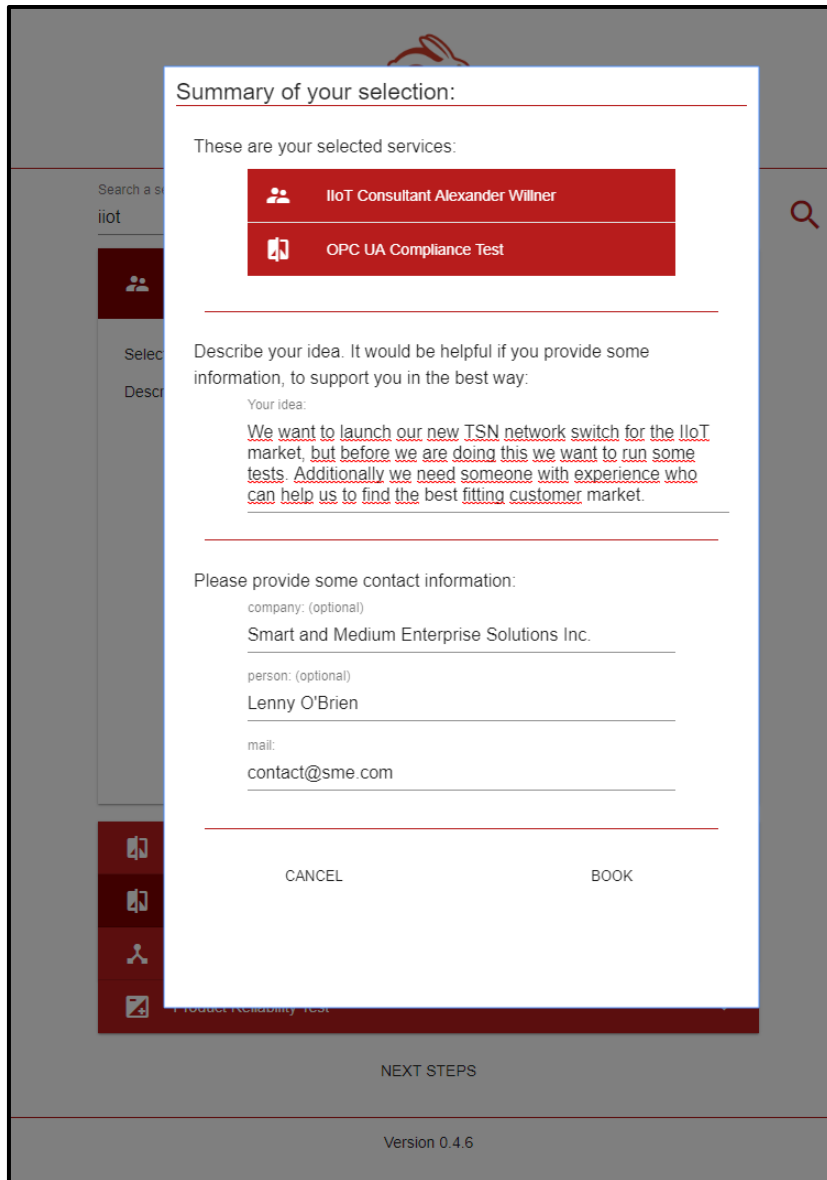


Figure 20: Request further Information after the Selection Phase.

After clicking on the “Book” button, the selection with the information is sent to the CRM, which will send a notification to the testbed providers.



3.3 LEAD CREATION

This last step completes the Service selection. The following activities are described in Lead Creation section.

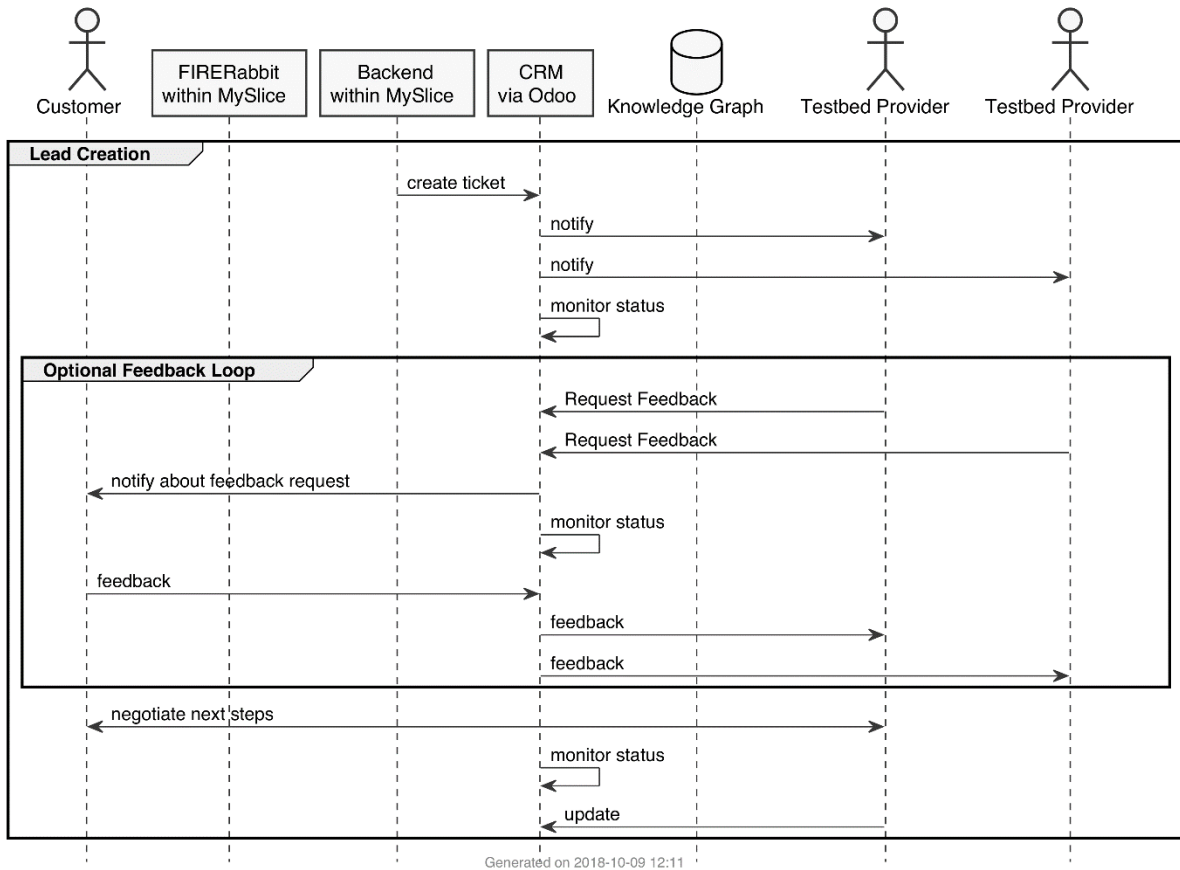


Figure 21: Overview of Activities for Lead Creation.

A customer books a service on market.fed4fire.eu. The vendor receives an Odoo notification via email. He clicks on the “View Lead/Opportunity” button (Figure 22) and sign-in to Odoo using their iMec account (Figure 23).





Figure 22: Odoo notification via email

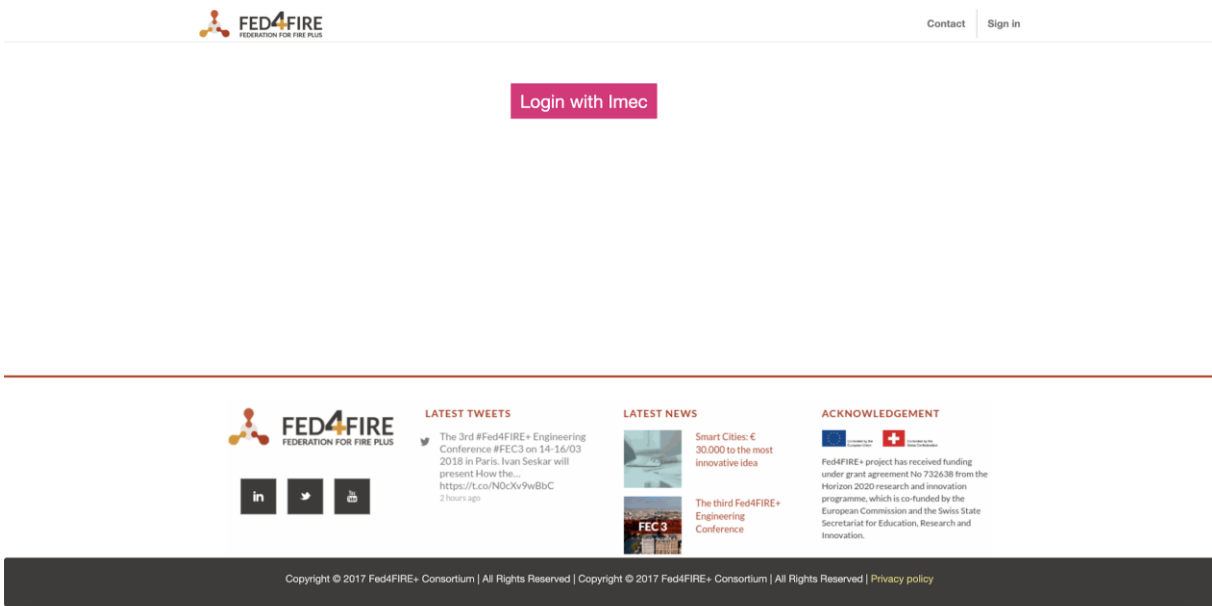


Figure 23: Odoo Log-in Page.

Figure 24 shows a page where the vendor can send a message to the customer (Send message button) or create an offer (New Quotation button).



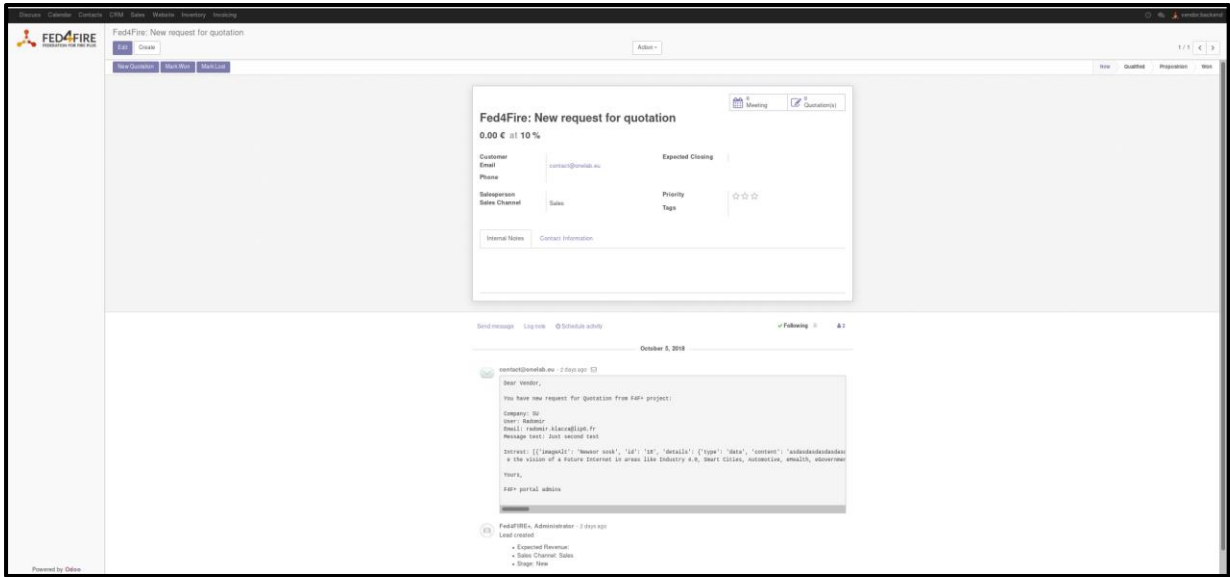


Figure 24: Vendor message page

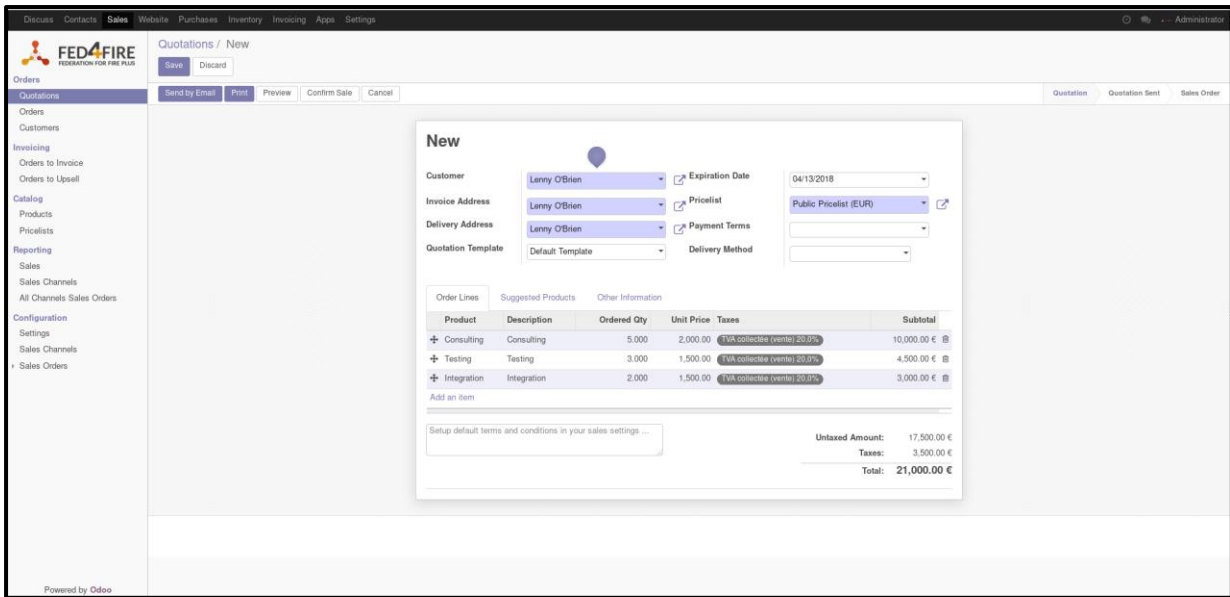


Figure 25: Odoo Backend – Vendor’s New Quotation View.

From this page (Figure 25) they can create a new quotation and fill in the form. In the case of a new customer, click on the drop-down list and select the “create and edit” option (Figure 26) and add a new contact to Odoo (Figure 25).

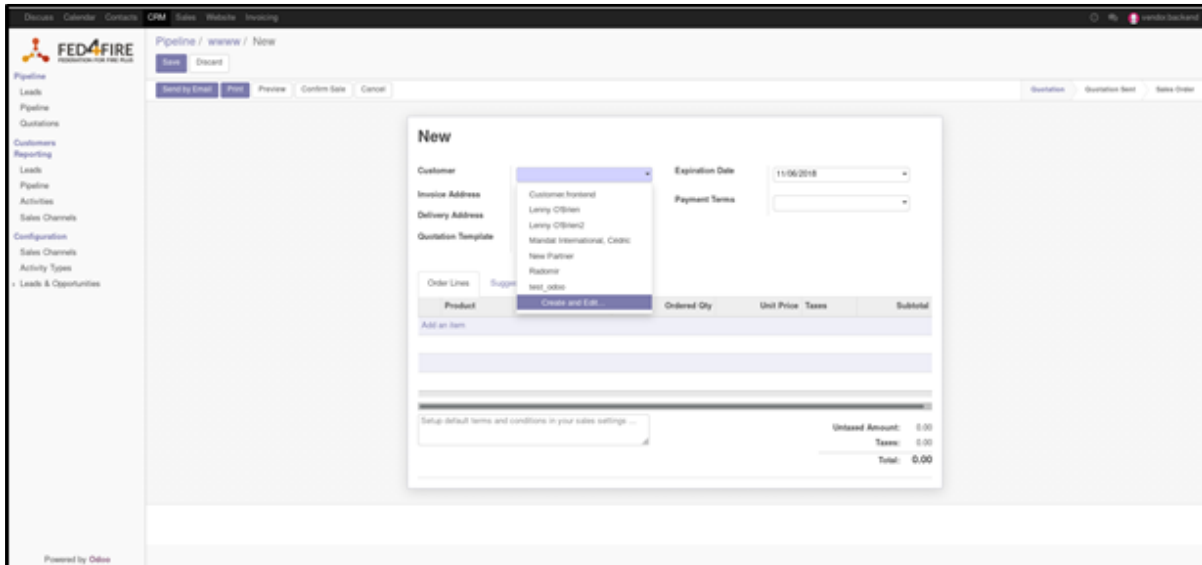


Figure 26

After a click on the “Send by email” button, the quotation will be saved automatically in Odoo (Figure 28).

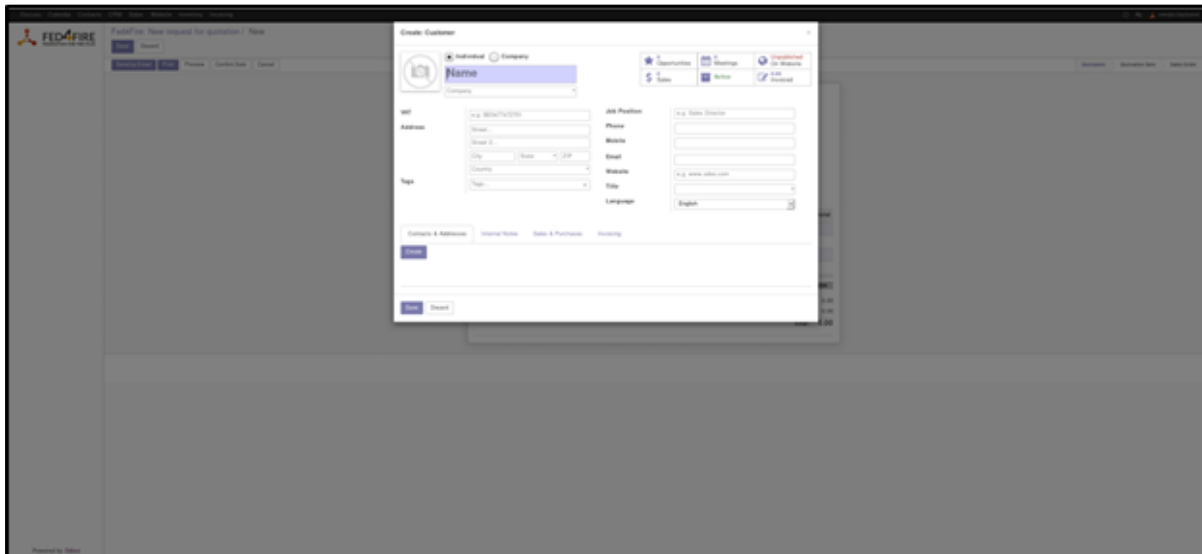


Figure 27



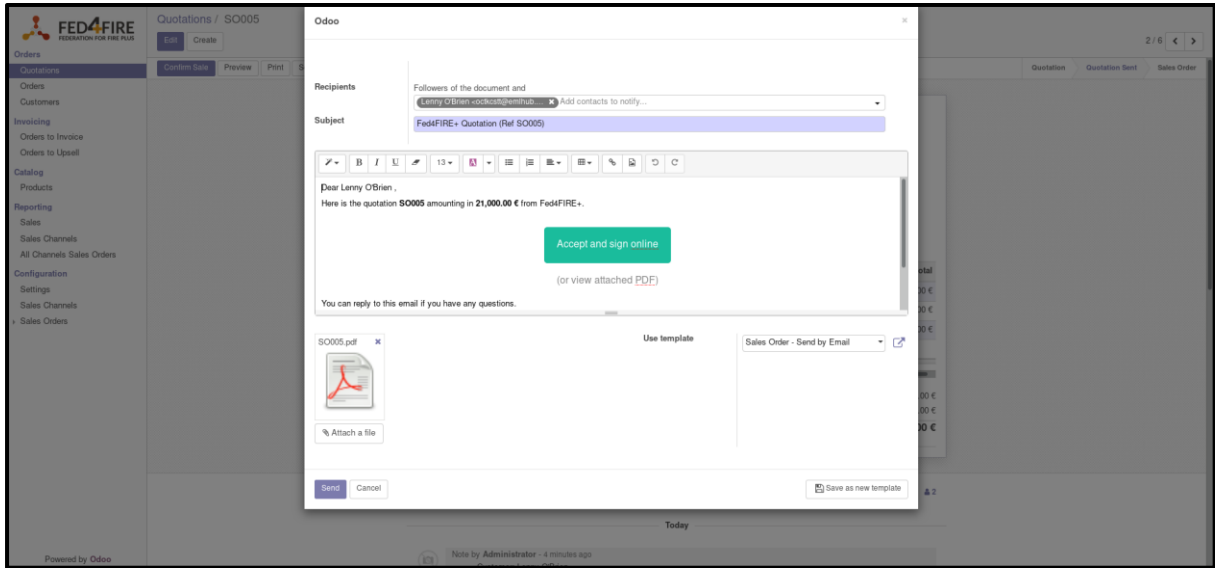


Figure 28: Odoo Backend – Vendor’s Email View.

All quotations are stored in Odoo, so both the customer and the vendor can view them at any time (Figure 29).

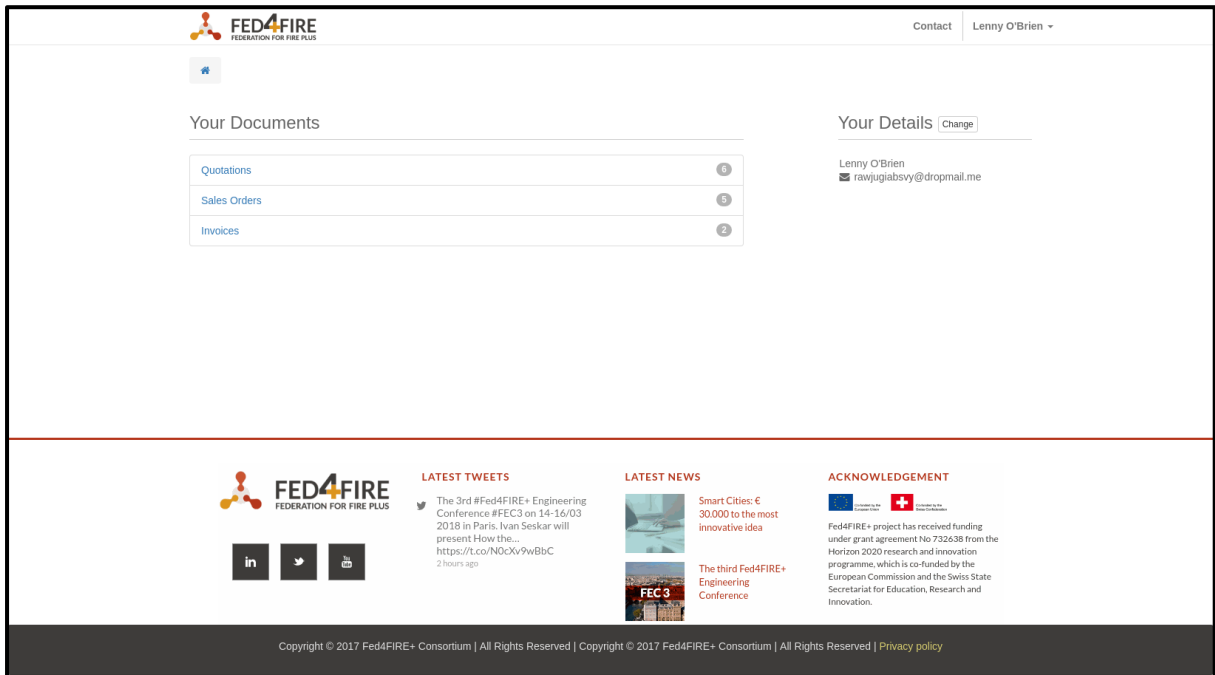


Figure 29: Odoo Frontend – Customer’s Welcome Page.



3.4 EXECUTION

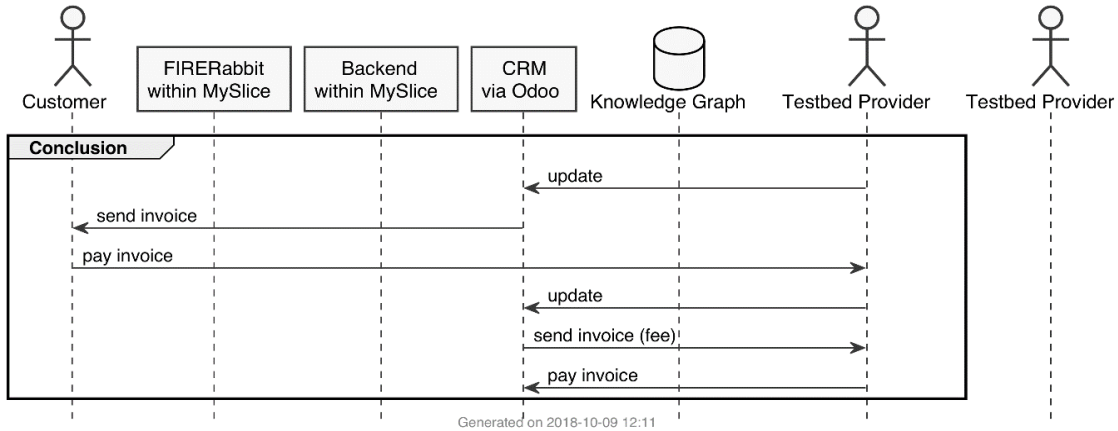


Figure 30: Overview of Activities for Execution.

After the request for quotation phase, the next steps are:

- ➔ Feedback between the customer and the vendor.
- ➔ The customer accepts the quotation.
- ➔ The quotation becomes a sale order.

3.4.1 Feedback between Customer and Vendor

In this phase, customer and vendor/Federation can exchange messages and emails, and can modify the quotation about the type of services, quantity and costs. All is stored in Odoo, so both documents and messages can be read, collected and shared with other people and printed anytime (Figure 31).

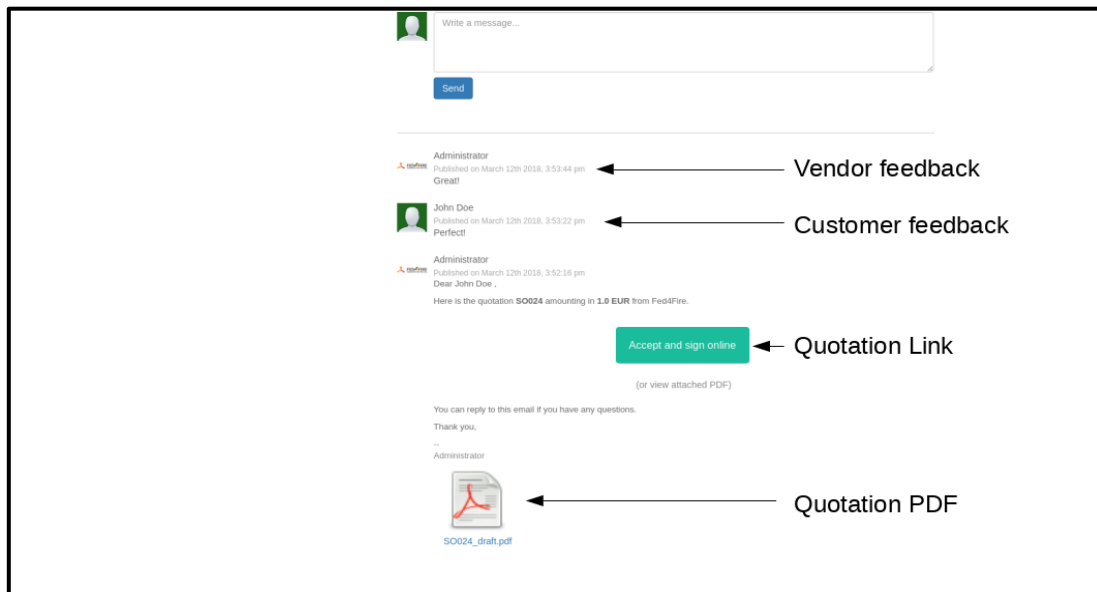


Figure 31: Odoo Frontend – Email & Message Section.



3.4.2 Customer Accepts the Quotation

When a customer accepts a quotation, it becomes a sales order. The customer can confirm his purchase in many ways such as wired payment, digital signatures and so on so forth. The most common method is through an online payment. The latter is the quickest and the most secure method for both customer and vendor/Federation (Figure 32).

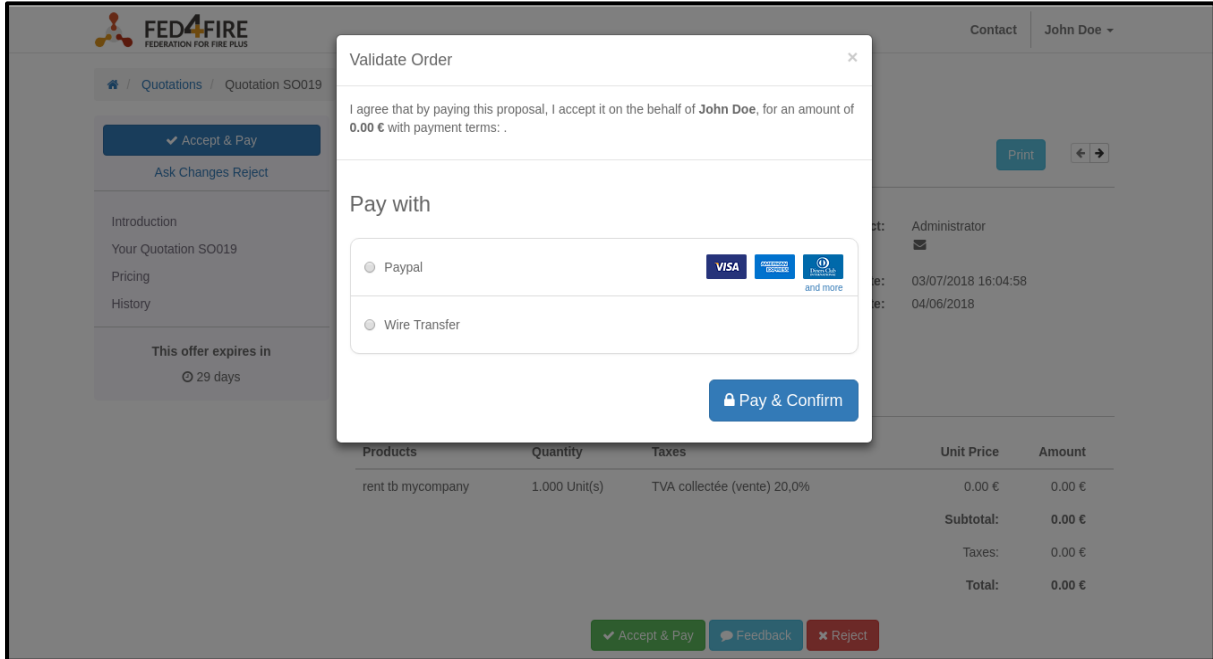


Figure 32: Odoo Frontend - Payment Confirmation View.

3.5 CONCLUSION

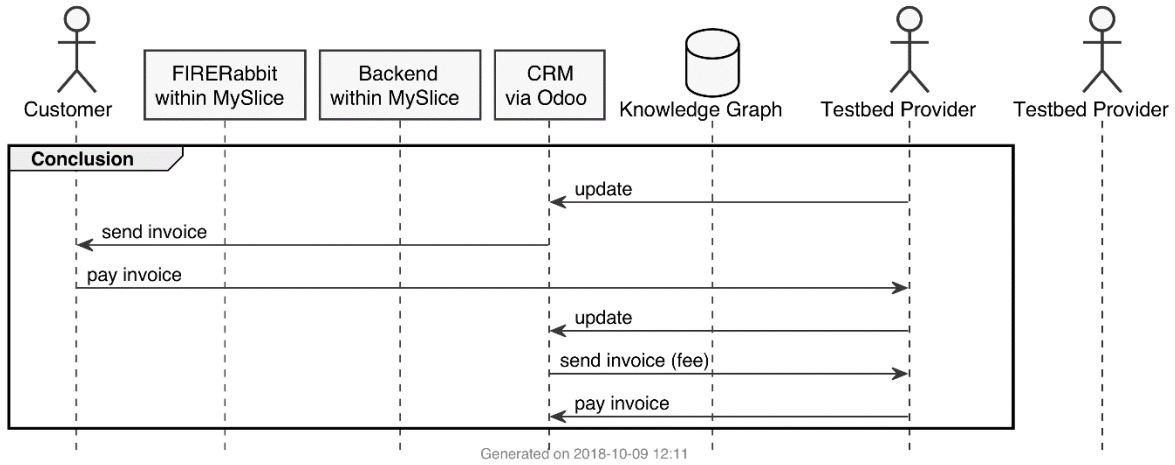


Figure 33: Overview of Activities for Conclusion.

In summary: the customer requests a resource through the marketplace, the testbed provider is notified through the CRM, and through it the provider can contact the customer and negotiate a deal. The customer can pay the provider, and the federation can charge a fee to the provider afterwards.

3.5.1 The Sales Order Becomes an Invoice

A sales order in Figure 34 is ready to become an invoice.

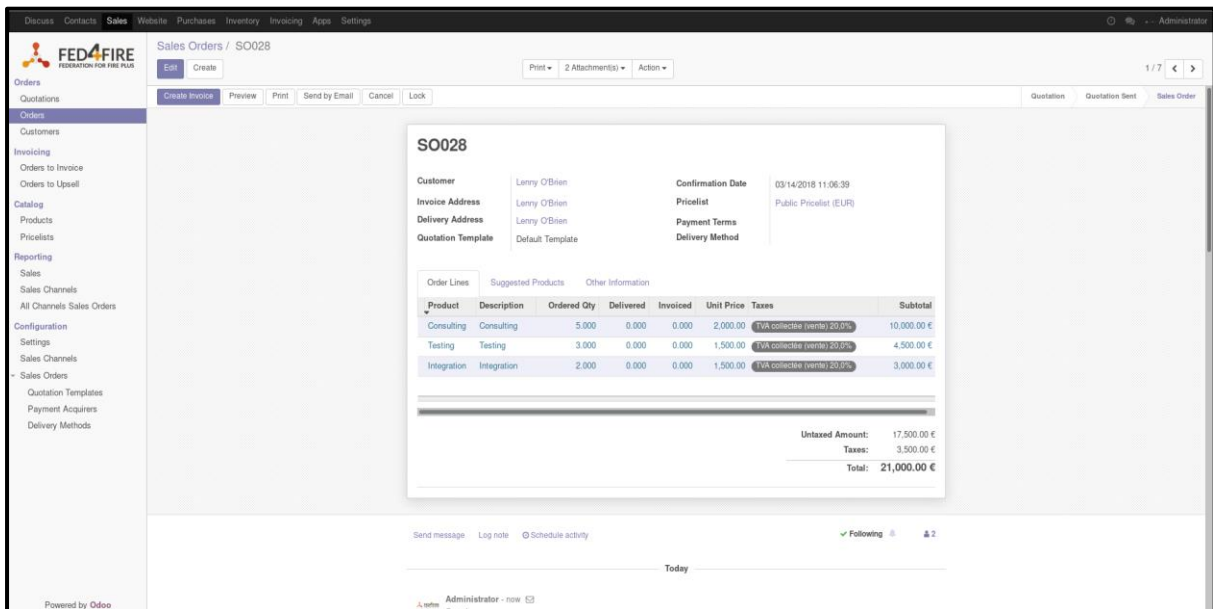


Figure 34: Odoo Backend – Vendor's Sales Order View.



3.5.2 Invoice and a Confirmation of Customer’s Payment

As noted above, the customer can confirm his order in two manners: either through a digital signature or through an online payment. In the first case, the vendor has to check that he has received the customer’s payment in his bank account (outside of Odoo), confirm the payment manually on Odoo, then send a payment confirmation to the customer (Figure 35). In the second case, the payment is confirmed automatically, an online payment provider sends a notification to the vendor and the electronic financial transaction (EFT) is done immediately.

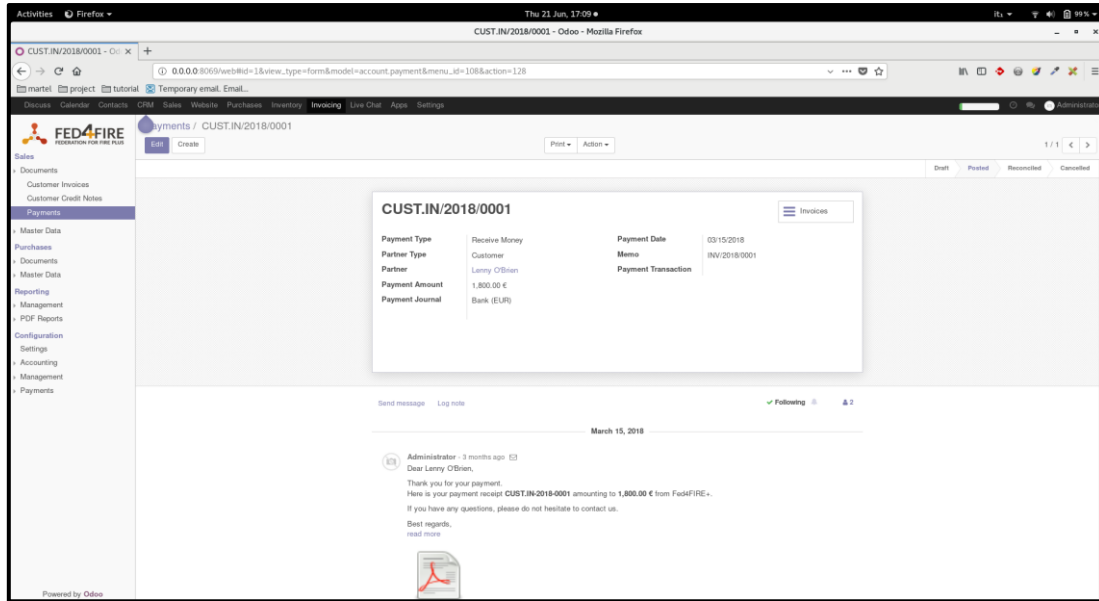


Figure 35: Odoo Backend – Vendor’s Invoice View.

Now the customer can get his service requested (Figure 36).

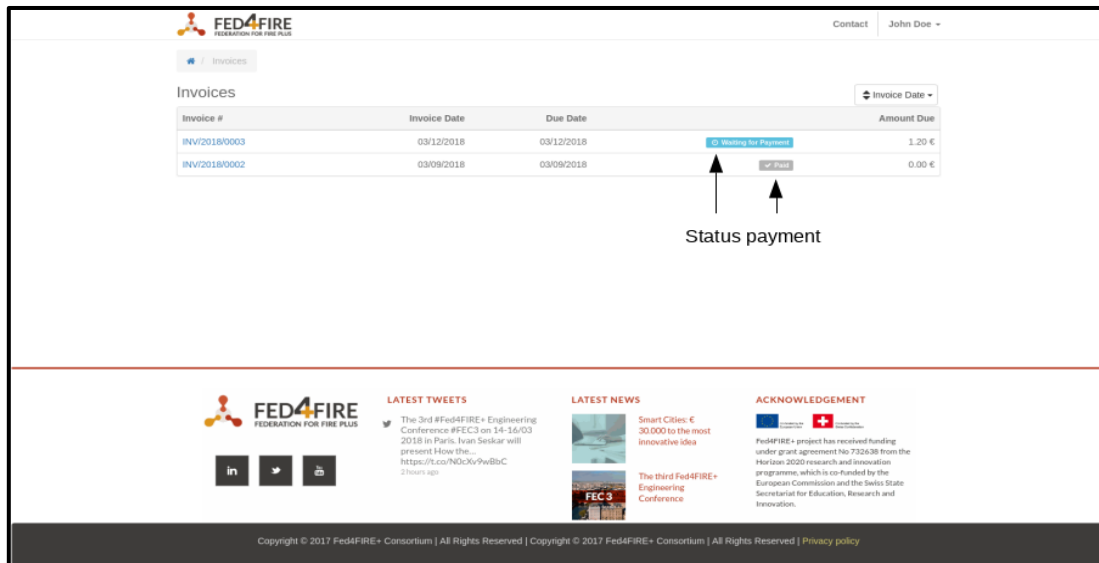


Figure 36: Odoo Frontend – Customer’s Payment List Page.



3.5.3 Evaluation

Through Odoo, External API and opportunistic settings, we can create an integration among Marketplace, MySlice and Odoo itself. In detail, the customer books a service/product on market.fed4fire.eu, filling in a digital form. Marketplace/MySlice system sends all data contained in the form to Odoo, which automatically stores them in its database. Odoo sends a notification to the vendor. Depending on what service/product the customer chooses, the Marketplace/MySlice system sends an email to the email address associated with the service/product selected.

For example, IIoT service is associated with the email address: sales1f4f@fed4fire.eu. The vendor's email address is associated with this email: vendorf4f@fed4fire.eu.

Now the customer selects IIoT service at market.fed4fire.eu, Marketplace/MySlice system sends an email to sales1f4f@fed4fire.eu.

Odoo receives the email, creates a lead/opportunity (the step before being an RFQ) in the associated vendor account and sends an email notification to vendor@fed4fire.eu.

The vendor gets the notification, logs in to his account (crm.fed4fire.eu) and converts the lead/opportunity to an RFQ. The RFQ is sent to the customer.

The customer gets an email with all RFQ data. He logs into his account (crm.fed4fire.eu) to manage it or to simply communicate with the vendor and the business starts.

4 CONCLUSION/FUTURE WORK

By combining existing tools, it was possible to create a prototype that has already met the first prototype requirements, i.e., only some components such as the marketplace or the matchmaking system have been developed. The next steps are outlined below.

4.1 MARKETPLACE

For the Marketplace component there is still a wide range of features that can be implemented. The Marketplace is fully integrated in MySlice, and with the integration of Oauth2 with Imec as IdP, a user can authenticate against Odoo and the Marketplace using his Imec credentials, the next steps are to improve the customer and testbed provider interaction as well as communication with some other components. The list of additional features:

- Integrating the Marketplace with the Unstructured and Structured search engines (APIs),
- Visually distinguish both search engines,
- Handover to Odoo (paid support) and jFed (reserve resources),
- Share the authentication cookie between the Marketplace and Odoo so users need to authenticate only once in order to access all the services,
- Extension of the user interface for logged-in user,
- Stronger integration of Odoo into the frontend.

4.2 MATCHMAKING

As with the Marketplace, the Matchmaking System also has features that can be added to it. Starting with the refinement of the presented architecture up to the replacement of individual steps by more suitable methods, as for example neural networks for the determination of the weights.

The list of additional features:

- Refinement of the current architecture in order to create an interoperable and intelligent system.
- Provide more data to the unstructured based search engine.
- Use Other Natural Language Processing (NLP) approaches for unstructured search engine.
- Further enhancements of matchmaking capabilities.
- Evaluate unstructured search engine.

Unstructured search is helpful at providing various information about the testbeds, however, it is hard to use when the user wants to receive information about the specifics of the testbeds like the information about the nodes that available, the number of nodes or other parts of infrastructure specific to a particular testbed. Ability to choose the testbed based on this characteristic can be vital in order for the user's decision to be appropriate.

Structured search Unstructured search

Available TestBeds France CLEAR FILTER

Countries

Grid'5000 ^

Select:

Nodes:

	Node	Available	Testbed	Location
<input checked="" type="checkbox"/>	dahu-2	true	Grid'5000	long:5.76705, lat:45.19021, France
<input type="checkbox"/>	dahu-24	true	Grid'5000	long:5.76705, lat:45.19021

Figure 37: Odoo Frontend – Ontology based Structured Search Engine.

In order to solve this problem an additional integrated structured search can be introduced. This way of ontological search should allow to filter the available testbeds according to the features and hardware they can provide and nodes according to their capabilities.

At this point in time several filters are available like:

- ➔ Location
- ➔ Hardware Type
- ➔ Country of origin
- ➔ Name
- ➔ Availability
- ➔ Features specific to some testbeds (special network interface for example)

A small prototype of this solution has already been implemented. Currently, the backend of the provided software is built into the backend of the unstructured search and it also uses flask.

One of the biggest challenges in providing this kind of service is gathering detailed information about the features of the testbeds. At this point in time the data is scraped and gathered from the <https://flsmonitor-api.fed4fire.eu/> which provides limited information about the testbeds and their nodes in form of rspec data. This information is, however, limited and in the future it was planned to contact the testbed providers in order to create a comprehensive ontological database and use it as a source of information.

4.3 MYSLICE

With the integration of MySlice with Oauth2 using iMec as IdP, it's now possible to synchronize users between the iMec Authority and MySlice, and create user account in the MySlice registry. The next steps will focus on :

- ➔ Improving the frontend to visually distinguish between customer and provider interface,
- ➔ Integrating MySlice with other Fed4Fire+ components.

4.4 ODOO

Odoo is currently integrated with the marketplace by creating leads via emails generated from the marketplace. Oauth support is provided with iMec's IdP. Following this, the future work includes the following:

- ➔ Better generation and organization of leads from various inquiries made by users about the testbeds
- ➔ Customize Odoo's look to match a unified look of all the Fed4Fire+ components
- ➔ Upgrade Odoo to the latest version, possibly make use of all it's newer CRM features and improvements. Update authentication to match.
- ➔ Investigate new options for invoicing and quotations, possibly direct payments
- ➔ Make use of the RPC API Odoo provides for creating leads, quotations and other models, to create more structured data programmatically.

4.5 PRICING

A pricing scheme based on the time and volume of resource usage will be implemented as a proof-of-concept in NITOS testbed. This module will monitor the usage of resources like CPU, RAM, Storage and mainly the interface traffic generated by the experiment execution. The rationale behind the pricing module is that the most valuable resource of a wireless testbed is the spectrum and its usage should be priced higher than anything else. Users usually spend a lot of time configuring and testing their experiment before being able to run it at full-scale and usually generate large volume of network traffic. To this end, a fair pricing scheme should take into consideration the downtime and should charge the experimenter only when the resources are actually used and occupy the limited and valuable spectrum. Moreover, this will provide incentive to users to reserve testbed resources that don't overcharge during the preparation period of an experiment.

5 WORKS CITED

- (1) Klacza, R., Vaissade, F., Willner, A., Ahmed, Z., & Rook, J. (2017). D4.01: TaaS Gap Analysis Report.
- (2) Nguyen, H., Tassinari, O., Brookes, M., Ross, K., Marks, N., Sebald, S., Crockett, T. (n.d.). Material-UI. Retrieved from <https://material-ui.com/>
- (3) Google. (n.d.). Design - Material Design. Retrieved from <https://material.io/design/>
- (4) Rocha, C., Schwabe, D., & Poggi de Aragao, M. (2004, May). A hybrid approach for searching in the semantic web. Proceedings of the 13th international conference on World Wide Web.
- (5) Facebook Inc. (2015). Flux - In Depth Overview. Retrieved from <https://facebook.github.io/flux/docs/in-depth-overview.html#content>
- (6) Fernandes, Joao, et al. "IoT Lab: Towards co-design and IoT solution testing using the crowd." 2015 International Conference on Recent Advances in Internet of Things (RIoT). IEEE, 2015.
- (7) Odoo Apps. Official Odoo apps store. Retrieved 10:24, November 02, 2018, from <https://apps.odoo.com/apps/browse>
- (8) Odoo, ORM API. Official Odoo Documentation. Retrieved 10:24, November 02, 2018, from www.odoo.com/documentation/11.0/reference/orm.html#reference-orm-model
- (9) "XML-RPC." Wikipedia, The Free Encyclopedia. 16 September 2018, at 15:33(UTC). <<https://en.wikipedia.org/wiki/XML-RPC>>.
- (10) Tomas Mikolov , Ilya Sutskever , Kai Chen, Greg Corrado , Jeffrey Dean, Distributed Representations of Words and Phrases and their Compositionality (2013).
- (11) Quoc Le (QVL@GOOGLE.COM), Tomas Mikolov (TMIKOLOV@GOOGLE.COM), Distributed Representations of Sentences and Documents

