
Business API Ecosystem Documentation

Release latest

January 31, 2017

1	Index	3
1.1	Installation and Administration Guide	3
1.2	User and Programmer Guide	27
1.3	Plugins Guide	100

This project is part of [FIWARE](#) and has been made in collaboration with the [TM Forum](#).

The Business API Ecosystem is a joint component made up of the FIWARE Business Framework and a set of APIs (and its reference implementations) provided by the TMForum. This component allows the monetization of different kind of assets (both digital and physical) during the whole service life cycle, from offering creation to its charging, accounting and revenue settlement and sharing. The Business API Ecosystem exposes its complete functionality through TMForum standard APIs; concretely, it includes the catalog management, ordering management, inventory management, usage management, billing, customer, and party APIs.

The Business API Ecosystem is not a single software repository, but it is composed of different projects which work coordinately to provide the complete functionality.

Concretely, the Business API Ecosystem is made of the following components:

- *Reference implementations of TM Forum APIs*: Reference implementation of the catalog management, ordering management, inventory management, usage management, billing, customer, and party APIs.
- *Business Ecosystem Charging Backend*: Is the component in charge of processing the different pricing models, the accounting information, and the revenue sharing reports. With this information, the Business Ecosystem Charging Backend is able to calculate amounts to be charged, charge customers, and pay sellers.
- *Business Ecosystem RSS*: Is in charge of distributing the revenues originated by the usage of a given service among the involved stakeholders. In particular, it focuses on distributing part of the revenue generated by a service between the Business API Ecosystem instance provider and the Service Provider(s) responsible for the service. With the term “service” we refer to both final applications and backend application services (typically exposed through an API). Note that, in the case of composite services, more than one service provider may have to receive a share of the revenues.
- *Business Ecosystem Logic Proxy*: Acts as the endpoint for accessing the Business API Ecosystem. On the one hand, it orchestrates the APIs validating user requests, including authentication, authorization, and the content of the request from a business logic point of view. On the other hand, it serves a web portal that can be used to interact with the system.

Installation and Administration Guide The guide for maintainers that explains how to install it.

User and Programmer Guide The guide for users and programmers that explains how to use it.

Plugins Guide The guide for admins that cover the available plugins

1.1 Installation and Administration Guide

1.1.1 Introduction

This installation and administration guide covers the Business API Ecosystem version 5.4.1, corresponding to FIWARE release 5. Any feedback on this document is highly welcomed, including bugs, typos or things you think should be included but aren't. Please send them to the "Contact Person" email that appears in the [Catalogue page for this GEi](#). Or create an issue at [GitHub Issues](#)

The current version of the software has been tested under Ubuntu 14.04, Ubuntu 15.10, Ubuntu 16.04, Debian 7, Debian 8, and CentOS 7. THESE ARE THEREFORE CONSIDERED AS THE SUPPORTED OPERATING SYSTEMS.

1.1.2 Installation

Requirements

As described in the GErI overview, the Business API Ecosystem is not a single software, but a set of projects that work together for proving business capabilities. In this regard, this section contains the basic dependencies of the different components that made up the Business API Ecosystem.

Note: These dependencies are not mean to be installed manually in this step, as they will be installed throughout the documentation

TM Forum APIs and RSS requirements

- Java 8
- Glassfish 4.1

- MySQL 5.5

Charging Backend requirements

- Python 2.7
- MongoDB
- wkhtmltopdf

Logic Proxy requirements

- NodeJS 4.5.0+ (Including NPM)

Installing basic dependencies

Basic dependencies such as Java 8, Glassfish, MySQL, Python, etc. Can be installed using the package management tools provided by your operating system. However, in order to easy the installation process some scripts have been provided.

Note: The installation script may override some of the packages already installed in the system. so if you have software with common dependencies you may want to manually resolve them.

Installing basic dependencies using the script

In order to automate the installation of the basic dependencies, the script *setup_env.sh* has been provided. This script, located in the root directory, installs all the needed packages for Ubuntu, Debian, and CentOS systems.

Additionally, this script creates a directory */opt/biz-ecosystem* where Glassfish 4.1 and Node 6.9.1 are downloaded, creates a */etc/default/rss* directory used later for properties files, and sets up the PATH environment in your *.bashrc* file.

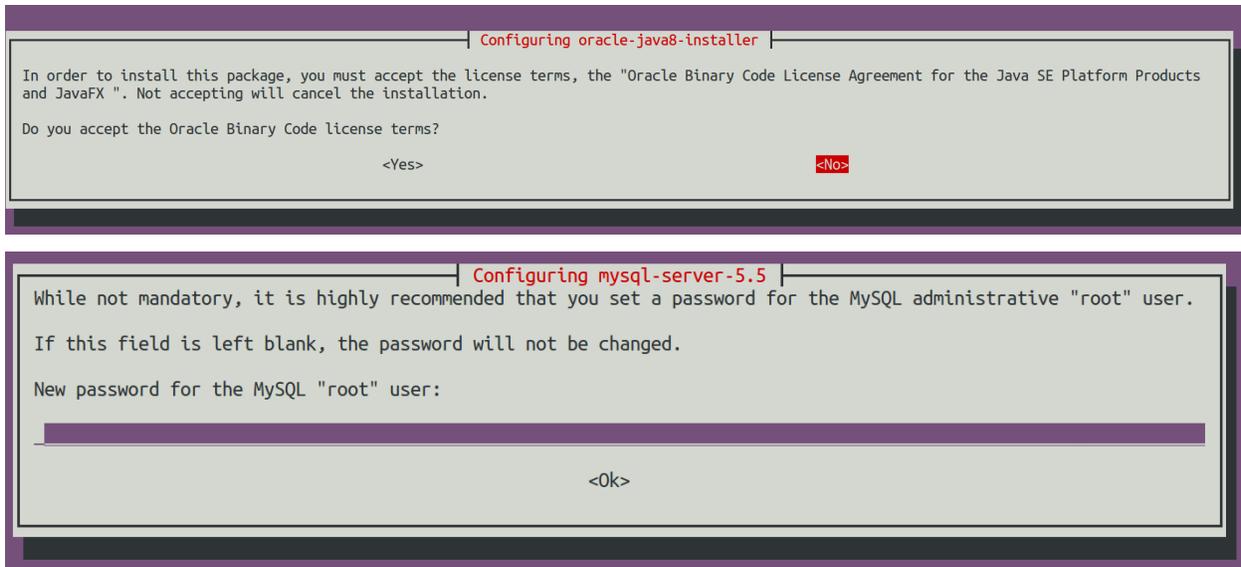
Note: The installation script changes the owner of all its created directories to your current user

To execute the script, run the following command from the root directory of the project

```
$ ./setup_env.sh
```

Note: Do not execute the script using *sudo*, for those tasks which require root privileges the script will prompt you to provide your *sudo* password

During the execution of the script you will be prompted some times in order to accept Oracle Java 8 terms and conditions and to provide MySQL root password.



Installing basic dependencies manually

Following, you can find how to install the basic dependencies without using the script. Be aware that some commands require to be executed as root.

APIs dependencies Java 8 Debian/Ubuntu

To install Java 8 in a Debian or Ubuntu system, it is needed to include the *webupd8team* repository. In an Ubuntu system this can be done directly with the following command:

```
$ sudo add-apt-repository ppa:webupd8team/java
```

In the case you have a Debian system the following commands have to be executed

```
$ sudo echo "deb http://ppa.launchpad.net/webupd8team/java/ubuntu trusty main" | tee /etc/apt/sources
$ sudo echo "deb-src http://ppa.launchpad.net/webupd8team/java/ubuntu trusty main" | tee -a /etc/apt
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys EEA14886
```

Then Java 8 can be installed using the following commands:

```
$ sudo apt-get update
$ sudo apt-get install -y oracle-java8-installer
$ sudo apt-get install -y oracle-java8-set-default
```

Java 8 CentOS 7

For a CentOS 7 system, the installation of Java 8 requires downloading the package from the official site

```
$ wget --no-cookies --no-check-certificate --header "Cookie: gpw_e24=http%3A%2F%2Fwww.oracle.com%2F;
$ tar xzf jdk-8u102-linux-x64.tar.gz
```

Then Java can be installed using *alternatives*

```
$ sudo alternatives --install /usr/bin/java java /opt/biz-ecosystem/jdk1.8.0_102/bin/java 2
$ sudo alternatives --config java

$ sudo alternatives --install /usr/bin/jar jar /opt/biz-ecosystem/jdk1.8.0_102/bin/jar 2
```

```
$ sudo alternatives --install /usr/bin/javac javac /opt/biz-ecosystem/jdk1.8.0_102/bin/javac 2
$ sudo alternatives --set jar /opt/biz-ecosystem/jdk1.8.0_102/bin/jar
$ sudo alternatives --set javac /opt/biz-ecosystem/jdk1.8.0_102/bin/javac
```

MySQL and Maven Debian/Ubuntu Once Java has been installed, the next step is installing MySQL and Maven

```
$ sudo apt-get install -y mysql-server mysql-client
$ sudo apt-get install -y maven
```

MySQL and Maven CentOS 7

For installing MySQL in CentOS, it is required to include the related repository before installing it

```
$ wget http://repo.mysql.com/mysql-community-release-el7-5.noarch.rpm
$ sudo rpm -ivh mysql-community-release-el7-5.noarch.rpm
$ sudo yum update

$ sudo yum install -y mysql-community-server
```

Then, for installing Maven

```
$ sudo wget http://repos.fedorapeople.org/repos/dchen/apache-maven/epel-apache-maven.repo -O /etc/yum.repos.d/epel-apache-maven.repo
$ sudo sed -i s/\$releasever/6/g /etc/yum.repos.d/epel-apache-maven.repo
$ sudo yum install -y apache-maven
```

Glassfish The next step is downloading and extracting Glassfish

```
$ wget http://download.java.net/glassfish/4.1/release/glassfish-4.1.zip
$ unzip glassfish-4.1.zip
```

Finally, it is required to download the MySQL connector for Glassfish and include it within the Glassfish *lib* directory

```
$ wget http://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-java-5.1.39.tar.gz
$ gunzip mysql-connector-java-5.1.39.tar.gz
$ tar -xvf mysql-connector-java-5.1.39.tar

$ cp mysql-connector-java-5.1.39/mysql-connector-java-5.1.39-bin.jar glassfish4/glassfish/lib
```

Charging Backend dependencies Python 2.7 Debian/Ubuntu

To install Python 2.7 and Pip in a Debian/Ubuntu distribution, execute the following command

```
$ sudo apt-get install -y python python-pip
```

Python 2.7 CentOS

Python 2.7 is included by default in CentOS 7. To install Pip it is required to include EPEL repository. All this stuff can be done executing the following commands

```
$ sudo rpm -iUvh http://dl.fedoraproject.org/pub/epel/7/x86_64/e/epel-release-7-5.noarch.rpm
$ sudo yum -y update
$ sudo yum install -y python-pip
```

MongoDB Debian/Ubuntu

To install MongoDB in a Debian/Ubuntu distribution, execute the following command

```
$ sudo apt-get install -y mongodb
```

MongoDB CentOS 7

To install MongoDB in CentOS it is needed to include its repository first. MongoDB can be installed executing the following commands

```
$ sudo echo "[mongodb]
name=MongoDB Repository
baseurl=http://downloads-distro.mongodb.org/repo/redhat/os/x86_64/
gpgcheck=0
enabled=1" > /etc/yum.repos.d/mongodb.repo

$ sudo yum install -y mongodb-org
```

Wkhtmltopdf Debian/Ubuntu

In Debian and Ubuntu Wkhtmltopdf is included in a package, so it can be directly installed with the following command

```
$ sudo apt-get install -y wkhtmltopdf
```

Wkhtmltopdf CentOS 7

In CentOS the Wkhtmltopdf RPM package has to be downloaded for installing it

```
$ wget http://download.gna.org/wkhtmltopdf/0.12/0.12.1/wkhtmltox-0.12.1_linux-centos7-amd64.rpm
$ sudo rpm -ivh wkhtmltox-0.12.1_linux-centos7-amd64.rpm
```

Logic Proxy Dependencies For installing Node and NPM it is needed to download the binaries from the official site and uncompress them

```
$ wget https://nodejs.org/dist/v6.9.1/node-v6.9.1-linux-x64.tar.xz
$ tar -xvf node-v6.9.1-linux-x64.tar.xz
```

Installing the Business API Ecosystem

As stated previously, the Business API Ecosystem is composed of different systems that need to be installed separately. In order to ease this process, it has been created an script **install.py** which can be used to automate the installation.

Installing the Business API Ecosystem using the script

The script *install.py* is located at the root of the Business API Ecosystem project. This script provides functionality to automate the installation of the software. Concretely, it downloads all the APIs and components, compiles and deploys the APIs, and installs python and node libraries.

This script depends on Python3 to work. If you have used the *setup_env.sh* script, Python 3 is already installed. Otherwise, you can install Python 3 using the following commands:

Debian/Ubuntu

```
$ sudo apt-get install -y python3
$ sudo apt-get install -y python3-pip
```

CentOS 7

```
$ sudo yum -y install scl-utils
$ sudo rpm -Uvh https://www.softwarecollections.org/en/scls/rhscl/python33/epel-7-x86_64/download/rhscl-python33-3.6.5-1.el7.x86_64.rpm
$ sudo yum -y install python33
```

Additionally, *install.py* specs the binaries of Glassfish and Node to be included in the PATH, and need to be accessible by the user using the script. This can be done with the following commands (Note that the commands are supposing both or them are installed at */opt/biz-ecosystem*)

```
$ export PATH=$PATH:/opt/biz-ecosystem/glassfish4/glassfish/bin
$ export PATH=$PATH:/opt/biz-ecosystem/node-6.9.1-linux-x64/bin

$ sudo chown -R <your_user>:<your_user> /opt/biz-ecosystem
```

If you have used *setup_env.sh**, the Glassfish installation directory already belongs to your user. In addition, the export PATH command has been included in your bashrc, so to have Node and Glassfish in the PATH execute the following command:

```
$ source ~/.bashrc
```

Moreover, *install.py* requires Glassfish, MySQL and MongoDB to be up and running.

Debian/Ubuntu

```
$ asadmin start-domain
$ sudo service mysql restart
$ sudo service mongodb restart
```

CentOS 7

```
$ asadmin start-domain
$ sudo systemctl start mysqld
$ sudo systemctl start mongod
```

Finally, during the deployment of the RSS API, the script saves the properties file in the default RSS properties directory. If you have used *setup_env.sh* this directory already exists. Otherwise, you have to manually create the directory */etc/default/rss*, being required to have root privileges to create it. Moreover, this directory must be accessible by the user executing the script. To do that

```
$ sudo mkdir /etc/default/rss
$ sudo chown <your_user>:<your_user> /etc/default/rss
```

The script *install.py* creates the different databases as well as the connection pools and resources. In this regard, after the execution of the script all the APIs will be already configured. You can specify the database settings by modifying the script and updating DBUSER, DBPWD, DBHOST, and DBPORT, which by default contains the following configuration.

```
DBUSER = "root"
DBPWD = "toor"
DBHOST = "localhost"
DBPORT = 3306
```

To make a complete installation of the Business API Ecosystem, execute the following command

```
$ ./install.py all
```

In addition to the *all* option, *install.py* also provides several options that allows to execute parts of the installation process, so you can have more control over it. Concretely, the script provides the following options:

- **clone**: Downloads from GitHub the different components of the Business API Ecosystem
- **maven**: Compiles the downloaded APIs using Maven
- **tables**: Creates the required databases in MySQL
- **persistence**: Builds persistence.xml files of the different APIs

- **pools:** Creates database pools in Glassfish
- **resources:** Creates database resources in Glassfish
- **redeploy:** Deploys APIs and RSS war files in Glassfish
- **proxy:** Installs proxy Node libs
- **charging:** Installs charging Python libs

Installing the Business API Ecosystem Manually

Installing TM Forum APIs The different reference implementations of the TM Forum APIs used in the Business API Ecosystem are available in GitHub:

- [Catalog Management API](#)
- [Product Ordering Management API](#)
- [Product Inventory Management API](#)
- [Party Management API](#)
- [Customer Management API](#)
- [Billing Management API](#)
- [Usage Management API](#)

The installation for all of them is similar. The first step is cloning the repository and moving to the correct release

```
$ git clone https://github.com/FIWARE-TMForum/DSPRODUCTCATALOG2.git
$ cd DSPRODUCTCATALOG2
$ git checkout v5.4.1
```

Once the software has been downloaded, it is needed to create the connection to the database. To do that, the first step is editing the `src/main/resources/META-INF/persistence.xml` to have something similar to the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
  <persistence-unit name="DSProductCatalogPU" transaction-type="JTA">
    <jta-data-source>jdbc/pcatv2</jta-data-source>
    <exclude-unlisted-classes>>false</exclude-unlisted-classes>
    <properties>
      <property name="javax.persistence.schema-generation.database.action" value="drop-and-create">
    </properties>
  </persistence-unit>
</persistence>
```

Note that you should provide in the tag `jta-data-source` the name you want for your database connection resource, taking into account that it must be unique for each API.

The next step is creating the database for you API.

```
$ mysql-u <user> -p<passwd> "CREATE DATABASE IF NOT EXISTS <database>"
```

Note: You have to provide your own credentials and the selected database name to the previous command.

Once that the database has been created, the next step is creating the connection pool in Glassfish. To do that, you can use the following command:

```
$ asadmin create-jdbc-connection-pool --restype java.sql.Driver --driverclassname com.mysql.jdbc.Driver
```

Note: You have to provide your own database credentials, the database host, the database port, the database name of the one created previously, and a name for your pool

The last step for creating the database connection is creating the connection resource. To do that, execute the following command:

```
$ asadmin create-jdbc-resource --connectionpoolid <poolname> <jndiname>
```

Note: You have to provide the name of the pool you have previously created and a name for your resource, which has to be the same as the included in the *jta-data-source* tag of the *persistence.xml* file of the API.

When the database connection has been created, the next step is compiling the API sources with Maven

```
$ mvn install
```

Finally, the last step is deploying the generated war file in Glassfish

```
$ asadmin deploy --contextroot <root> --name <root> target/<WAR.war>
```

Note: You have to provide the wanted context root for the API, a name for it, and the path to the war file

Installing the RSS [The RSS sources can be found in GitHub](#)

The first step for installing the RSS component is downloading it and moving to the correct release

```
$ git clone https://github.com/FIWARE-TMForum/business-ecosystem-rss.git
$ cd business-ecosystem-rss
$ git checkout v5.4.1
```

Then, the next step is copying, *database.properties* and *oauth.properties* files to its default location at */etc/default/rss*

```
$ sudo mkdir /etc/default/rss
$ sudo chown <your_user>:<your_user> /etc/default/rss
$ cp properties/database.properties /etc/default/rss/database.properties
$ cp properties/oauth.properties /etc/default/rss/ouath.properties
```

Note: You have to include your user when changing *rss* directory owner

Once the properties files have been copied, they should be edited in order to provide the correct configuration params:

database.properties

```
database.url=jdbc:mysql://localhost:3306/RSS
database.username=root
database.password=root
database.driverClassName=com.mysql.jdbc.Driver
```

oauth.properties

```
config.grantedRole=Provider
config.sellerRole=Seller
config.aggregatorRole=aggregator
```

Note: The different params included in the configuration file are explained in detail in the Configuration section

Once the properties files have been edited, the next step is compiling the sources with Maven

```
$ mvn install
```

Finally, the last step is deploying the generated war file in Glassfish

```
$ asadmin deploy --contextroot DSRevenueSharing --name DSRevenueSharing fiware-rss/target/DSRevenueS
```

Installing the Charging Backend The Charging Backend sources can be found in [GitHub](#)

The first step for installing the charging backend component is downloading it and moving to the correct release

```
$ git clone https://github.com/FIWARE-TMForum/business-ecosystem-charging-backend.git
$ cd business-ecosystem-charging-backend
$ git checkout v5.4.1
```

Once the code has been downloaded, it is recommended to create a virtualenv for installing python dependencies (This is not mandatory).

```
$ virtualenv virtenv
$ source virtenv/bin/activate
```

To install python libs, execute the *python-dep-install.sh* script

```
$ ./python-dep-install.sh
```

Note: If you have not created and activated a virtualenv you will need to execute the script using sudo

Installing the Logic Proxy The Logic Proxy sources can be found in [GitHub <https://github.com/FIWARE-TMForum/business-ecosystem-logic-proxy>](https://github.com/FIWARE-TMForum/business-ecosystem-logic-proxy)

The first step for installing the logic proxy component is downloading it and moving to the correct release

```
$ git clone https://github.com/FIWARE-TMForum/business-ecosystem-logic-proxy.git
$ cd business-ecosystem-logic-proxy
$ git checkout v5.4.1
```

Once the code has been downloaded, Node dependencies can be installed with npm as follows

```
$ npm install
```

1.1.3 Configuration

At this step, the different components of the Business API Ecosystem are installed. In the case of the TMForum APIs and the RSS, this installation process has already required to configure their database connection before their deployment, so they are already configured. Nevertheless, this section contains an explanation of the function of the different settings of the RSS properties files.

Configuring the RSS

The RSS has its settings included in two files located at `/etc/default/rss`. The file `database.properties` contains by default the following fields:

```
database.url=jdbc:mysql://localhost:3306/RSS
database.username=root
database.password=root
database.driverClassName=com.mysql.jdbc.Driver
```

This file contains the configuration required in order to connect to the database.

- `database.url`: URL used to connect to the database, this URL includes the host and port of the database as well as the concrete database to be used
- `database.username`: User to be used to connect to the database
- `database.password`: Password of the database user
- `database.driverClassName`: Driver class of the database. By default MySQL

The file `oauth.properties` contains by default the following fields (It is recommended not to modify them)

```
config.grantedRole=Provider
config.sellerRole=Seller
config.aggregatorRole=aggregator
```

This file contains the name of the roles (registered in the idm) that are going to be used by the RSS.

- `config.grantedRole`: Role in the IDM of the users with admin privileges
- `config.sellerRole`: Role in the IDM of the users with seller privileges
- `config.aggregatorRole`: Role of the users who are admins of an store instance. In the context of the Business API Ecosystem there is only a single store instance, so you can safely ignore this flag

Configuring the Charging Backend

The Charging Backend creates some objects and connections in the different APIs while working, so the first step is configuring the different URLs of the Business API Ecosystem components by modifying the file `services_settings.py`, which by default contains the following content:

```
CATALOG = 'http://localhost:8080/DSPProductCatalog'
INVENTORY = 'http://localhost:8080/DSPProductInventory'
ORDERING = 'http://localhost:8080/DSPProductOrdering'
BILLING = 'http://localhost:8080/DSBillingManagement'
RSS = 'http://localhost:8080/DSRevenueSharing'
USAGE = 'http://localhost:8080/DSUsageManagement'
AUTHORIZE_SERVICE = 'http://localhost:8004/authorizeService/apiKeys'
```

This settings points to the different APIs accessed by the charging backend. In particular:

- `CATALOG`: URL of the catalog API including its path
- `INVENTORY`: URL of the inventory API including its path
- `ORDERING`: URL of the ordering API including its path
- `BILLING`: URL of the billing API including its path
- `RSS`: URL of the RSS including its path
- `USAGE`: URL of the Usage API including its path

- `AUTHORIZE_SERVICE`: Complete URL of the usage authorization service. This service is provided by the logic proxy, and is used to generate API Keys to be used by accounting systems when providing usage information.

Once the services have been configured, the next step is configuring the database. In this case, the charging backend uses MongoDB, and its connection can be configured modifying the `DATABASES` setting of the `settings.py` file.

```
DATABASES = {
    'default': {
        'ENGINE': 'django_mongodb_engine',
        'NAME': 'wstore_db',
        'USER': '',
        'PASSWORD': '',
        'HOST': '',
        'PORT': '',
        'TEST_NAME': 'test_database',
    }
}
```

This setting contains the following fields:

- `ENGINE`: Database engine, must be fixed to `django_mongodb_engine`
- `NAME`: Name of the database to be used
- `USER`: User of the database. If empty the software creates a non authenticated connection
- `PASSWORD`: Database user password. If empty the software creates a non authenticated connection
- `HOST`: Host of the database. If empty it uses the default `localhost` host
- `PORT`: Port of the database. If empty it uses the default `27017` port
- `TEST_NAME`: Name of the database to be used when running the tests

Once the database connection has been configured, the next step is configuring the name of the IdM roles to be used by updating `settings.py`

```
ADMIN_ROLE = 'provider'
PROVIDER_ROLE = 'seller'
CUSTOMER_ROLE = 'customer'
```

This settings contain the following values:

- `ADMIN_ROLE`: IDM role of the system admin
- `PROVIDER_ROLE`: IDM role of the users with seller privileges
- `CUSTOMER_ROLE`: IDM role of the users with customer privileges

The Charging Backend component is able to send email notifications to the users when they are charged or receive a payment. In this way, it is possible to provide email configuration in the `settings.py` file by modifying the following fields:

```
WSTOREMAILUSER = 'email_user'
WSTOREMAIL = 'wstore_email'
WSTOREMAILPASS = 'wstore_email_passwd'
SMTPSERVER = 'wstore_smtp_server'
SMTPPORT = 587
```

This settings contain the following values: * `WSTOREMAILUSER`: Username used for authenticating in the email server * `WSTOREMAIL`: Email to be used as the sender of the notifications * `WSTOREMAILPASS`: Password of the user for authenticating in the email server * `SMTPSERVER`: Email server host * `SMTPPORT`: Email server port

Note: The email configuration is optional. However, the field `WSTOREMAIL` must be provided since it is used internally for RSS configuration

Additionally, the Charging Backend is the component that charges customers and pays providers. For this purpose it uses PayPal. For configuring paypal, the first step is setting `PAYMENT_METHOD` to `paypal` in the `settings.py` file

```
PAYMENT_METHOD = 'paypal'
```

Then, it is required to provide PayPal application credentials by updating the file `src/wstore/charging_engine/payment_client/paypal_client.py`

```
PAYPAL_CLIENT_ID = ''
PAYPAL_CLIENT_SECRET = ''
MODE = 'sandbox' # sandbox or live
```

This settings contain the following values:

- `PAYPAL_CLIENT_ID`: Id of the application provided by PayPal
- `PAYPAL_CLIENT_SECRET`: Secret of the application provided by PayPal
- `MODE`: Mode of the connection. It can be `sandbox` if using the PayPal sandbox for testing the system. Or `live` if using the real PayPal APIs

Moreover, the Charging Backend is the component that activates the purchased services. In this regard, the Charging Backend has the possibility of signing its acquisition notifications with a certificate, so the external system being offered can validate that is the Charging Backend the one making the request. To use this functionality it is needed to configure the certificate and the private Key to be used by providing its path in the following settings of the `settings.py` file

```
NOTIF_CERT_FILE = None
NOTIF_CERT_KEY_FILE = None
```

Finally, the last step is creating the context of the Charging Backend by creating two sites. First, create the external site by executing the following command. Note that you have to provide the real URL where the proxy will be running.

```
$ ./manage.py createsite external http://<proxy_path>:<proxy_port>/
```

Then, you have to create the local site by providing the real URL where the Charging Backend will be running as follows

```
$ ./manage.py createsite local http://localhost:<charging_port>/
```

The Charging Backend uses a Cron task to check the status of recurring and usage subscriptions, and for paying sellers. The periodicity of this tasks can be configured using the `CRONJOBS` setting of `settings.py` using the standard Cron format

```
CRONJOBS = [
    ('0 5 * * *', 'django.core.management.call_command', ['pending_charges_daemon']),
    ('0 6 * * *', 'django.core.management.call_command', ['resend cdrs'])
]
```

Once the Cron task has been configured, it is necessary to include it in the Cron tasks using the command:

```
$ ./manage.py crontab add
```

It is also possible to show current jobs or remove jobs using the commands:

```
$ ./manage.py crontab show
$ ./manage.py crontab remove
```

Configure Apache for running the Charging Backend

The Charging Backend is a Django App that can be deployed in different ways. In this case, this installation guide covers two different mechanisms: using the Django *runserver* command (as seen in *Running the Charging Backend* section) or deploying it using an Apache server. This section explains how to configure Apache and the Charging Backend to do the later.

The first step is installing Apache and mod-wsgi. In Ubuntu/Debian:

```
$ sudo apt-get install apache2 libapache2-mod-wsgi
```

Or in CentOS:

```
$ sudo yum install httpd mod_wsgi
```

The next step is populating the file *src/wsgi.py* provided with the Charging Backend

```
import os
import sys

path = 'charging_path/src'
if path not in sys.path:
    sys.path.insert(0, path)

os.environ['DJANGO_SETTINGS_MODULE'] = 'settings'

import django.core.handlers.wsgi
application = django.core.handlers.wsgi.WSGIHandler()
```

If you are using a virtualenv, then you will need to include its activation in your *wsgi.py* file, so it should look similar to the following:

```
import os
import sys
import site

site.addsitedir('virtualenv_path/local/lib/python2.7/site-packages')
path = 'charging_path/src'
if path not in sys.path:
    sys.path.insert(0, path)

os.environ['DJANGO_SETTINGS_MODULE'] = 'settings'

# Activate your virtual env
activate_env=os.path.expanduser('virtualenv_path/bin/activate_this.py')
execfile(activate_env, dict(__file__=activate_env))

import django.core.handlers.wsgi
application = django.core.handlers.wsgi.WSGIHandler()
```

Note: Pay special attention to *charging_path* and *virtualenv_path* which have to point to the Charging Backend and the virtualenv paths respectively.

Once WSGI has been configured in the Charging Backend, the next step is creating a virtualhost in Apache. To do that, you can create a new site configuration file in the Apache related directory (located in `/etc/apache2/sites-available/` in an Ubuntu/Debian system or in `/etc/httpd/conf.d` in a CentOS system) and populate it with the following content:

```
<VirtualHost *:8006>
    WSGIDaemonProcess char_process
    WSGIScriptAlias / charging_path/src/wsgi.py
    WSGIProcessGroup char_process
    WSGIPassAuthorization On

    WSGIApplicationGroup %{GLOBAL}
</VirtualHost>
```

Note: Pay special attention to `charging_path` which have to point to the Charging Backend path.

Depending on the version of Apache you are using, you may need to explicitly allow the access to the directory where the Charging Backend is deployed in the configuration of the virtualhost. To do that, add the following lines to your virtualhost:

Apache version < 2.4

```
<Directory charging_path/src>
    Order deny,allow
    Allow from all
</Directory>
```

Apache version 2.4+

```
<Directory charging_path/src>
    Require all granted
</Directory>
```

Once you have included the new virtualhost configuration, the next step is configuring Apache to listen in the selected port (8006 in the example). To do that, edit `/etc/apache2/ports.conf` in Ubuntu/Debian or `/etc/httpd/conf/httpd.conf` in CentOS and include the following line:

```
Listen 8006
```

Then, in Ubuntu/Debian systems, enable the site by linking the configuration file to the `sites-enabled` directory:

```
ln -s ../sites-available/001-charging.conf ../sites-enabled/001-charging.conf
```

Once you have the site enabled, restart Apache. In Ubuntu/Debian

```
$ sudo service apache2 restart
```

Or in CentOS

```
$ sudo apachectl restart
```

Note: Ensure that the directory where the Changing Backend is installed can be accessed by the Apache user (`www-data` in Ubuntu/Debian, and `apache` in CentOS)

Configuring the Logic Proxy

The first step for configuring the proxy is creating the configuration file by copying `config.js.template` to `config.js`

```
$ cp config.js.template config.js
```

The first setting to be configured is the port where the proxy is going to run, this setting is located in *config.js*

```
config.port = 80;
```

If you want to run the proxy in HTTPS you can update *config.https* setting

```
config.https = {
  enabled: false,
  certFile: 'cert/cert.crt',
  keyFile: 'cert/key.key',
  caFile: 'cert/ca.crt',
  port: 443
};
```

In this case you have to set *enabled* to true, and provide the paths to the certificate (*certFile*), to the private key (*keyFile*), and to the CA certificate (*caFile*).

Then, it is possible to modify some of the URLs of the system. Concretely, it is possible to provide a prefix for the API, a prefix for the portal, and modifying the login and logout URLs

```
config.proxyPrefix = '';
config.portalPrefix = '';
config.logInPath = '/login';
config.logOutPath = '/logout';
```

Additionally, the proxy is the component that acts as the front end of the Business API Ecosystem, both providing a web portal, and providing the endpoint for accessing to the different APIs. In this regard, the Proxy has to have the OAuth2 configuration of the FIWARE IDM.

To provide OAuth2 configuration, an application has to be created in an instance of the FIWARE IdM (e.g <https://account.lab.fiware.org>), providing the following information:

- URL: `httphttps://<proxy_host>:<proxy_port>`
- Callback URL: `httphttps://<PROXY_HOST>:<PROXY_PORT>/auth/fiware/callback`
- Create a role *Seller*

Once the application has been created in the IdM, it is possible to provide OAuth2 configuration by modifying the following settings

```
config.oauth2 = {
  'server': 'https://account.lab.fiware.org',
  'clientID': '<client_id>',
  'clientSecret': '<client_secret>',
  'callbackURL': 'http://<proxy_host>:<proxy_port>/auth/fiware/callback',
  'roles': {
    'admin': 'provider',
    'customer': 'customer',
    'seller': 'seller'
  }
};
```

In this settings, it is needed to include the IDM instance being used (*server*), the client id given by the IdM (*clientID*), the client secret given by the IdM (*clientSecret*), and the callback URL configured in the IdM (*callbackURL*)

Moreover, the Proxy uses MongoDB for maintaining some info, such as the current shopping cart of a user. you can configure the connection to MongoDB by updating the following setting:

```
config.mongodb = {
  server: 'localhost',
  port: 27017,
  user: '',
  password: '',
  db: 'belp'
};
```

In this setting you can configure the host (*server*), the port (*port*), the database user (*user*), the database user password (*password*), and the database name (*db*).

As already stated, the Proxy is the component that acts as the endpoint for accessing the different APIs. In this way, the proxy needs to know the URLs of them in order to redirect the different requests. This endpoints can be configured using the following settings

```
config.endpoints = {
  'catalog': {
    'path': 'DSPProductCatalog',
    'host': 'localhost'
    'port': '8080',
    'appSsl': false
  },
  'ordering': {
    'path': 'DSPProductOrdering',
    'host': 'localhost'
    'port': '8080',
    'appSsl': false
  },
  ...
}
```

The setting *config.endpoints* contains the specific configuration of each of the APIs, including its *path*, its *host*, its *port*, and whether the API is using SSL or not.

Note: The default configuration included in the config file is the one used by the installation script, so if you have used the script for installing the Business API Ecosystem you do not need to modify these fields

Finally, there are two fields that allow to configure the behaviour of the system while running. On the one hand, *config.revenueModel* allows to configure the default percentage that the Business API Ecosystem is going to retrieve in all the transactions. On the other hand, *config.usageChartURL* allows to configure the URL of the chart to be used to display product usage to customers in the web portal.

1.1.4 Final steps

The Business API Ecosystem, allows to upload some product attachments and assets to be sold. These assets are uploaded by the Charging Backend that saves them in the file system, jointly with the generated PDF invoices.

In this regard, the directories *src/medi*a*, **src/media/bills*, and *src/media/assets* must exist within the Charging Backend directory, and must be writable by the user executing the Charging Backend.

```
$ mkdir src/media
$ mkdir src/media/bills
$ mkdir src/media/assets
$ chown -R <your_user>:<your_user> src/media
```

Additionally, the Business API Ecosystem uses indexes for efficiency and pagination. In this regards, the directory *indexes* must exist within the Logic Proxy directory, and must be writable by the user executing it.

```
$ mkdir indexes
$ chown -R <your_user>:<your_user> indexes
```

You can populate at any time the indexes directory using the *fill_indexes.js* script provided with the Logic Proxy.

```
$ node fill_indexes.js
```

1.1.5 Running the Business API Ecosystem

Running the APIs and the RSS

Both the TM Forum APIs and the RSS are deployed in Glassfish; in this regard, the only step for running them is starting Glassfish

```
$ asadmin start-domain
```

Running the Charging Backend

The Charging Backend creates some objects and connections on startup; in this way, the Glassfish APIs must be up and running before starting it.

Using Django runserver

The Charging Backend can be started using the *runserver* command provided with Django as follows

```
$ ./manage.py runserver 127.0.0.1:<charging_port>
```

Or in background

```
$ nohup ./manage.py runserver 127.0.0.1:<charging_port> &
```

Note: If you have created a virtualenv when installing the backend or used the installation script, you will need to activate the virtualenv before starting the Charging Backend

Using Apache

If you have deployed the charging backend in Apache, you can start it with the following command in a Debian/Ubuntu system

```
$ sudo service apache2 start
```

Or in a CentOS system

```
$ sudo apachectl start
```

Running the Logic Proxy

The Logic Proxy can be started using Node as follows

```
$ node server.js
```

Or if you want to start it in background:

```
$ nohup node server.js &
```

1.1.6 Installing Asset Plugins

The Business API Ecosystem is intended to support the monetization of different kind of digital assets. The different kind of assets that may be wanted to be monetized will be heterogeneous and potentially very different between them.

Additionally, for each type of asset different validations and activation mechanisms will be required. For example, if the asset is a CKAN dataset, it will be required to validate that the provider is the owner of the dataset. Moreover, when a customer acquires the dataset, it will be required to notify CKAN that a new user has access to it.

The huge differences between the different types of assets that can be monetized in the Business API Ecosystem makes impossible to include its validations and characteristics as part of the core software. For this reason, it has been created a plugin based solution, where all the characteristics of an asset type are implemented in a plugin that can be loaded in the Business API Ecosystem.

To include an asset plugin execute the following command in the Charging Backend:

```
$ ./manage.py loadplugin ckandataset.zip
```

It is possible to list the existing plugins with the following command:

```
$ ./manage.py listplugins
```

To remove an asset plugin, execute the following command providing the plugin id given by the *listplugins* command

```
$ ./manage.py removeplugin ckan-dataset
```

Note: For specific details on how to create a plugin and its internal structure, have a look at the Business API Ecosystem Programmer Guide

At the time of writing, the following plugins are available:

- **WireCloud Component:** Allows the monetization of WireCloud components, including Widgets, operators, and mashups
- **Accountable Service :** Allows the monetization of services protected by the **Accounting Proxy**, including Orion Context Broker queries
- **CKAN Dataset :** Allows the monetization of CKAN datasets

1.1.7 Sanity check Procedures

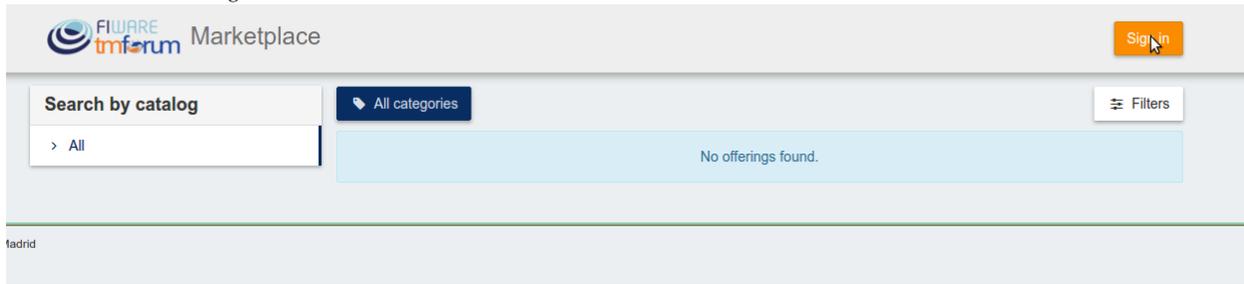
The Sanity Check Procedures are the steps that a System Administrator will take to verify that an installation is ready to be tested. This is therefore a preliminary set of tests to ensure that obvious or basic malfunctioning is fixed before proceeding to unit tests, integration tests and user validation.

End to End Testing

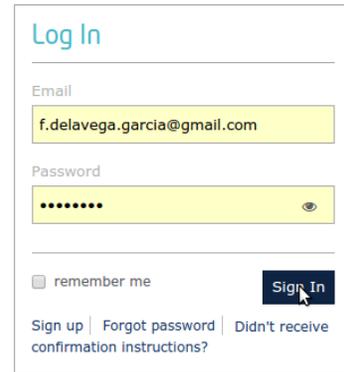
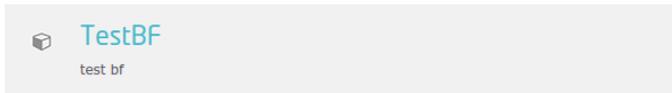
Please note that the following information is required before starting with the process: * The host and port where the Proxy is running * A valid IdM user with the *Seller* role

To Check if the Business API Ecosystem is running, follow the next steps:

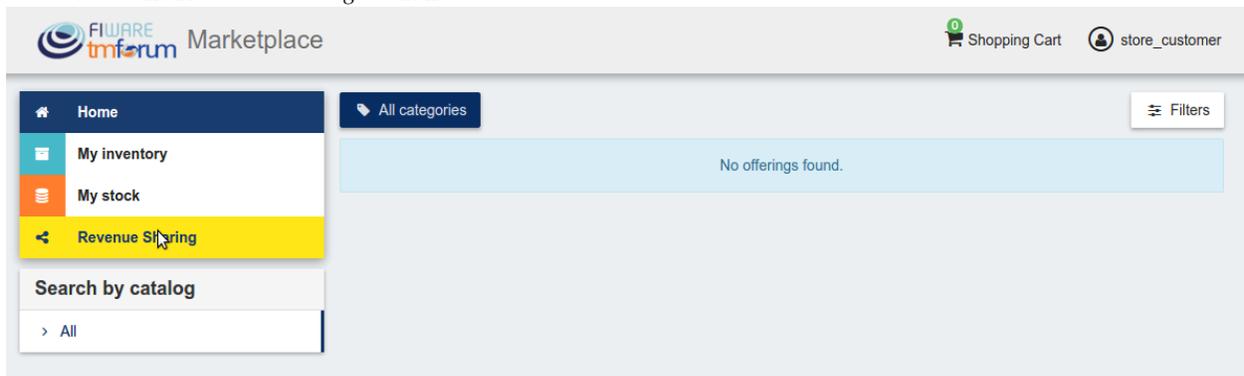
1. Open a browser and enter to the Business API Ecosystem
2. Click on the *Sign In* Button



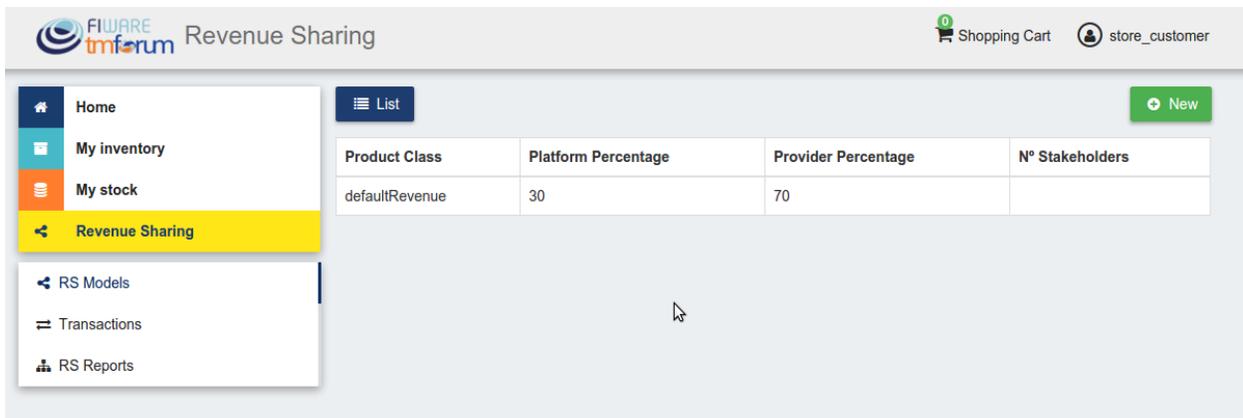
3. Provide your credentials in the IdM page



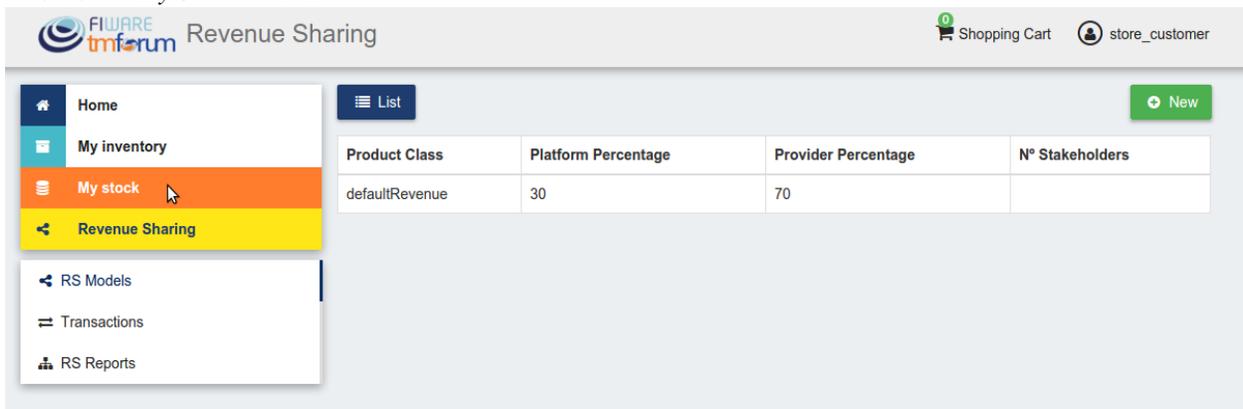
4. Go to the *Revenue Sharing* section



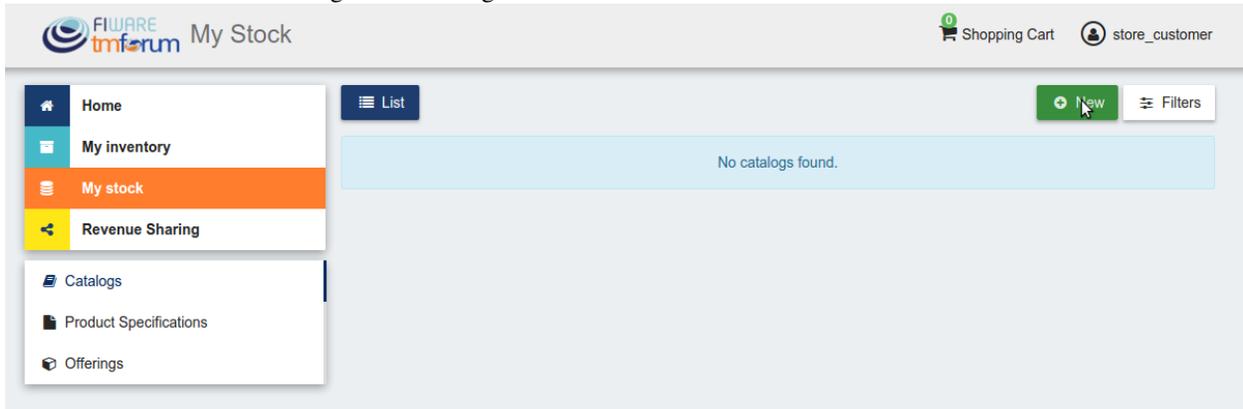
5. Ensure that the default RS Model has been created



6. Go to *My Stock* section



7. Click on *New* for creating a new catalog



8. Provide a name and a description and click on *Next*. Then click on *Create*

FIWARE tmforum My Stock Shopping Cart 0 store_customer

- Home
- My inventory
- My stock
- Revenue Sharing
- Catalogs
- Product Specifications
- Offerings

List New

New catalog

1 General **Step 1: General**

2 Finish

Enter a name
New Catalog

Enter a description (optional)
This is a new example catalog

Next

FIWARE tmforum My Stock Shopping Cart 0 store_customer

- Home
- My inventory
- My stock
- Revenue Sharing
- Catalogs
- Product Specifications
- Offerings

List New

New catalog

1 General

2 Finish **Step 2: Finish**

Name
New Catalog

Status
Active Launched Retired Obsolete

Description
This is a new example catalog

Create

FIWARE tmforum My Stock Shopping Cart 0 store_customer

- Home
- My inventory
- My stock
- Revenue Sharing
- Catalogs
- Product Specifications
- Offerings

List Details

New Catalog

About Parties Offerings

General

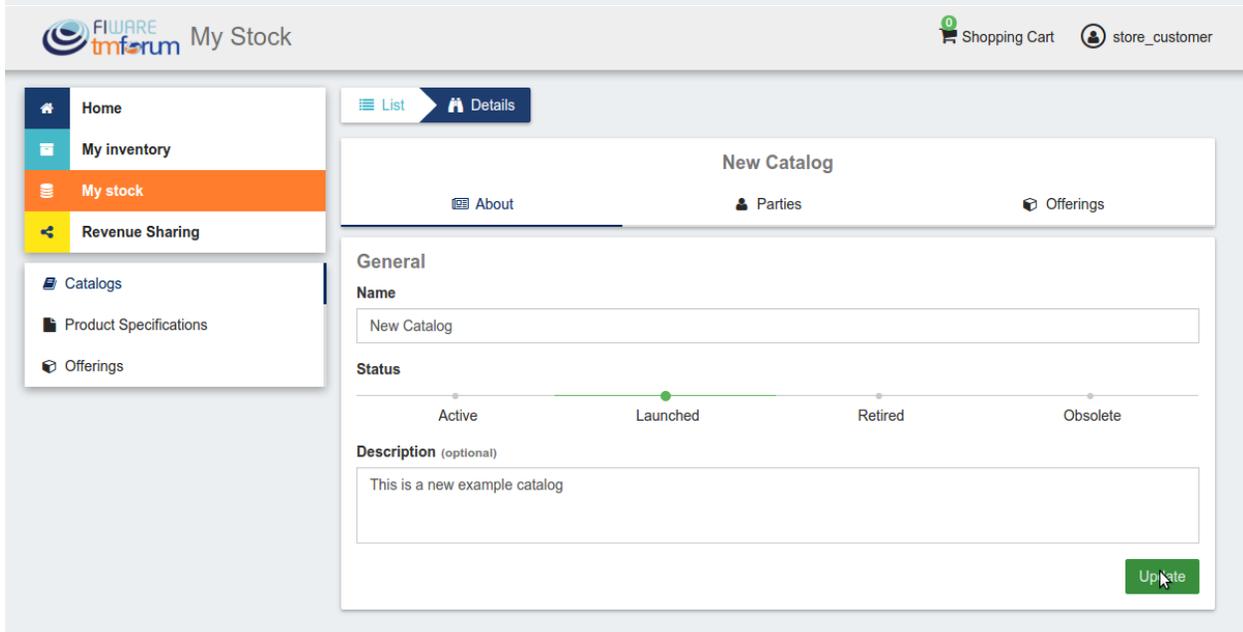
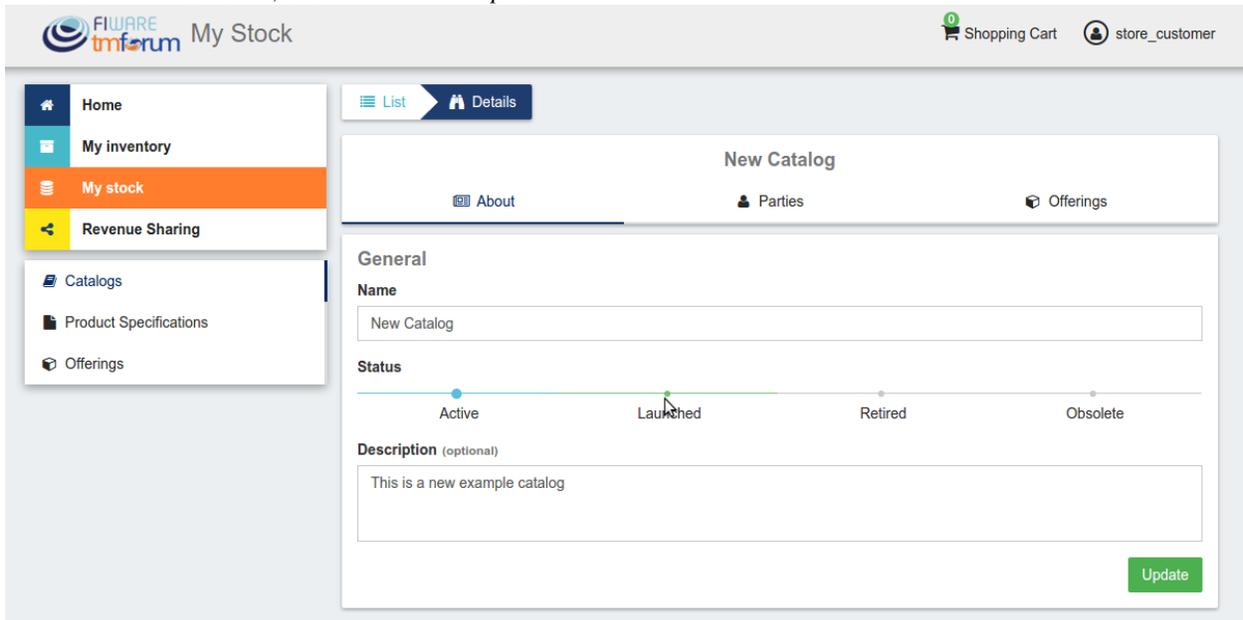
Name
New Catalog

Status
Active Launched Retired Obsolete

Description (optional)
This is a new example catalog

Update

9. Click on *Launched*, and then click on *Update*



10. Go to *Home*, and ensure the new catalog appears

The top screenshot shows the 'My Stock' interface. The left sidebar contains navigation links: Home, My inventory, My stock (highlighted), Revenue Sharing, Catalogs, Product Specifications, and Offerings. The main content area is titled 'New Catalog' and includes tabs for 'About', 'Parties', and 'Offerings'. The 'General' section has a 'Name' field with 'New Catalog', a 'Status' dropdown set to 'Launched', and a 'Description' field with 'This is a new example catalog'. An 'Update' button is at the bottom right.

The bottom screenshot shows the 'Marketplace' interface. The left sidebar has the same navigation links, with 'My stock' highlighted. Below the sidebar is a 'Search by catalog' section with 'All' and 'New Catalog' options. The main content area shows 'All categories' and a 'Filters' button. A light blue message box states 'No offerings found.'

List of Running Processes

We need to check that Java for the Glassfish server (APIs and RSS), python (Charging Backend) and Node (Proxy) are running, as well as MongoDB and MySQL databases. If we execute the following command:

```
ps -ewF | grep 'java|mongodb|mysql|python|node' | grep -v grep
```

It should show something similar to the following:

```
mongodb  1014    1  0 3458593 49996 0 sep08 ?          00:22:30 /usr/bin/mongod --config /etc/mongod
mysql    1055    1  0 598728 64884 2 sep08 ?          00:02:21 /usr/sbin/mysqld
francis+ 15932 27745  0 65187 39668  0 14:53 pts/24  00:00:08 python ./manage.py runserver 0.0.0.0
francis+ 15939 15932  1 83472 38968  0 14:53 pts/24  00:00:21 /home/user/business-ecosystem-chargin
francis+ 16036 15949  0 330473 163556 0 14:54 pts/25  00:00:08 node server.js
root     1572    1  0 1142607 1314076 3 sep08 ?          00:37:40 /usr/lib/jvm/java-8-oracle/bin/java -
```

Network interfaces Up & Open

To check the ports in use and listening, execute the command:

```
$ sudo netstat -nltp
```

The expected results must be something similar to the following:

```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.1:8006         0.0.0.0:*               LISTEN      15939/python
tcp        0      0 127.0.0.1:27017       0.0.0.0:*               LISTEN      1014/mongod
tcp        0      0 127.0.0.1:28017       0.0.0.0:*               LISTEN      1014/mongod
tcp        0      0 127.0.0.1:3306        0.0.0.0:*               LISTEN      1055/mysqld
tcp6       0      0 :::80                  :::*                    LISTEN      16036/node
tcp6       0      0 :::8686                :::*                    LISTEN      1572/java
tcp6       0      0 :::4848                :::*                    LISTEN      1572/java
tcp6       0      0 :::8080                :::*                    LISTEN      1572/java
tcp6       0      0 :::8181                :::*                    LISTEN      1572/java
```

Databases

The last step in the sanity check, once we have identified the processes and ports, is to check that MySQL and MongoDB databases are up and accepting queries. We can check that MySQL is working, with the following command:

```
$ mysql -u <user> -p<password>
```

You should see something similar to:

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 174
Server version: 5.5.47-0ubuntu0.14.04.1 (Ubuntu)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

For MongoDB, execute the following command:

```
$ mongo <database> -u <user> -p <password>
```

You should see something similar to:

```
MongoDB shell version: 2.4.9
connecting to: <database>
>
```

1.1.8 Diagnosis Procedures

The Diagnosis Procedures are the first steps that a System Administrator will take to locate the source of an error in a GE. Once the nature of the error is identified with these tests, the system admin will very often have to resort to more concrete and specific testing to pinpoint the exact point of error and a possible solution. Such specific testing is out of the scope of this section.

Resource Availability

Memory use depends on the number of concurrent users as well as the free memory available and the hard disk. The Business API Ecosystem requires a minimum of 1024 MB of available RAM memory, but 2048 MB of free memory are recommended. Moreover, the Business API Ecosystem requires at least 15 GB of hard disk space.

Remote Service Access

N/A

Resource Consumption

Resource consumption strongly depends on the load, especially on the number of concurrent users logged in.

- Glassfish main memory consumption should be between 500 MB and 2048 MB
- MongoDB main memory consumption should be between 30 MB and 500 MB
- Python main memory consumption should be between 30 MB and 200 MB
- Node main memory consumption should be between 30 MB and 200 MB
- MySQL main memory consumption should be between 30 MB and 500 MB

I/O Flows

The only expected I/O flow is of type HTTP, on port defined in the Logic Proxy configuration file

1.2 User and Programmer Guide

1.2.1 Introduction

This user and programmer guide covers the Business API Ecosystem version 5.4.1, corresponding to FIWARE release 5. Any feedback on this document is highly welcomed, including bugs, typos or things you think should be included but aren't. Please send them to the "Contact Person" email that appears in the [Catalogue page for this GEi](#). Or create an issue at [GitHub Issues](#)

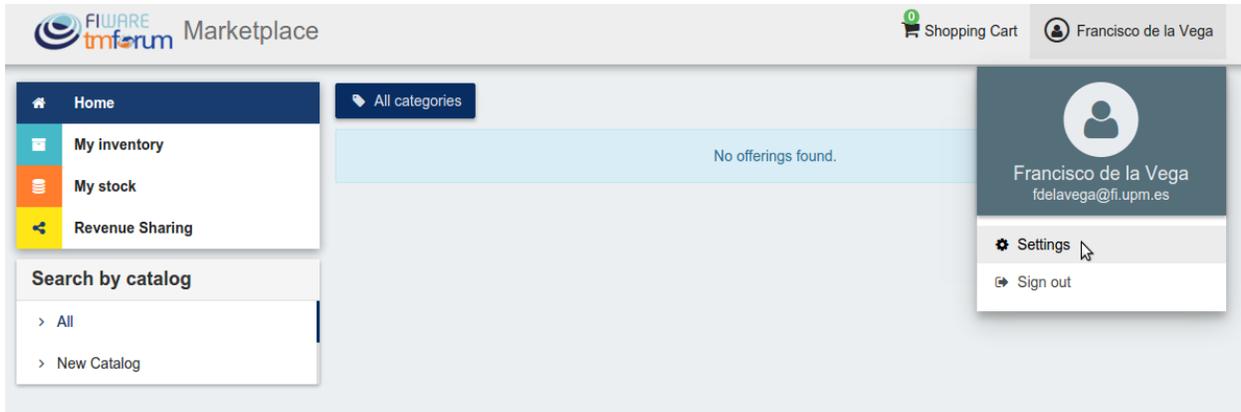
1.2.2 User Guide

This user guide contains a description of the different tasks that can be performed in the Business API Ecosystem using its web interface. This section is organized so the actions related to a concrete user role are grouped together.

Profile Configuration

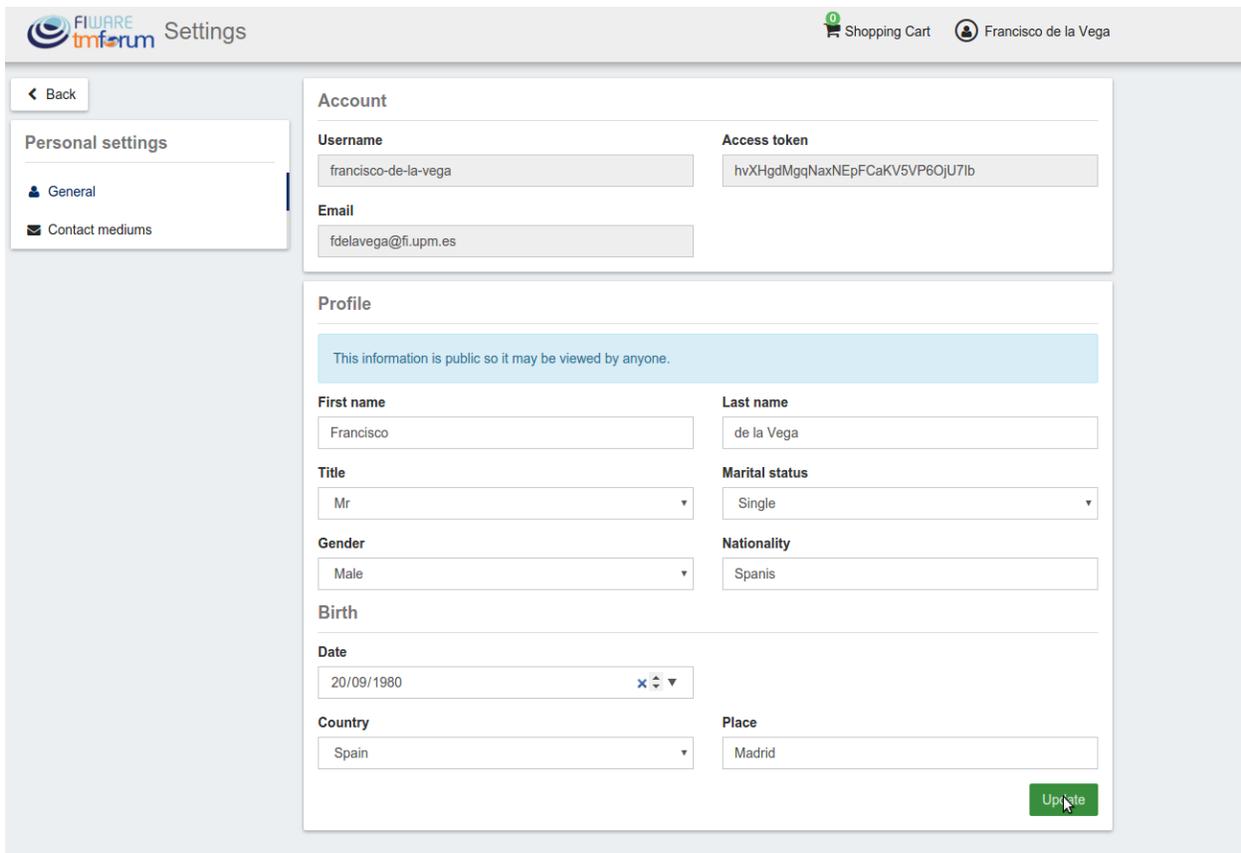
All the users of the system can configure their profile, so they can configure their personal information as well as their billing addresses and contact mediums.

To configure the user profile, the first step is opening the user *Settings* located in the user menu.



In the displayed view, it can be seen that some information related to the account is already included (*Username, Email, Access token*). This information is the one provided by the IdM after the login process.

To create the profile, fill in the required information and click on *Update*



Note: Only the *First name* and *Last name* fields are mandatory

Once you have created your profile, you can include contact mediums by going to the *Contact mediums* section.

The screenshot displays the 'Settings' page for a user named Francisco de la Vega. The page is divided into two main sections: 'Account' and 'Profile'. The 'Account' section contains fields for 'Username' (francisco-de-la-vega), 'Access token' (hvXHgdMgcNaxNEpFCaKV5VP60JU71b), and 'Email' (fdelavega@fi.upm.es). The 'Profile' section includes a warning that the information is public. It contains fields for 'First name' (Francisco), 'Last name' (de la Vega), 'Title' (Mr), 'Marital status' (Single), 'Gender' (Male), 'Nationality' (Spanis), 'Birth Date' (20/09/1980), 'Country' (Spain), and 'Place' (Madrid). A green 'Update' button is located at the bottom right of the profile section.

In the *Contact Medium* section, there are two different tabs. On the one hand, the *Shipping addresses* tab, where you can register the shipping addresses you will be able to use when creating orders and purchasing products.

To create a shipping address, fill in the fields and click on *Create*

Settings Shopping Cart Francisco de la Vega

Shipping addresses Business addresses

The shipping addresses will be used in your orders.

New shipping address

Email address

Email

Postal address

Street

Postcode **City**

State / Province **Country**

Telephone number

Type **Number**

[Create](#)

Once created, you can edit the address by clicking on the *Edit* button of the specific address, and changing the wanted fields.

Settings Shopping Cart Francisco de la Vega

Shipping addresses Business addresses

The shipping addresses will be used in your orders.

My shipping addresses

Email address	Postal address	Telephone number	Actions
fdelavega@fi.upm.es	Campus de Montegancedo S/N 28041 Madrid (Madrid) Spain	Mobile, +34611111111	Edit

New shipping address

Email address

Email

On the other hand, if you have the *Seller* role you can create *Business Addresses*, which can be used by your customers in order to allow them to contact you.

Email address	Postal address	Telephone number	Actions
fdelavega@fi.upm.es	Campus de Montegancedo S/N 28041 Madrid (Madrid) Spain	Mobile, +34611111111	

In the *Business Addresses* tab you can create, different kind of contact mediums, including emails, phones, and addresses. To create a contact medium, fill in the fields and click on *Create*

FIWARE **inforum** Settings Shopping Cart Francisco de la Vega

[← Back](#)

Shipping addresses **Business addresses**

This information is public so it may be viewed by anyone.

New business address

Medium
Email address

Email
fdelavega.provider@fi.upm.es

Create

FIWARE **inforum** Settings Shopping Cart Francisco de la Vega

[← Back](#)

Shipping addresses **Business addresses**

This information is public so it may be viewed by anyone.

My business addresses

Medium	Details	Actions
Email address	fdelavega.provider@fi.upm.es	 

New business address

Medium
Telephone number

Type
mobile

Number
+34 622222222

Create

Settings

Shopping Cart Francisco de la Vega

Shipping addresses Business addresses

This information is public so it may be viewed by anyone.

My business addresses

Medium	Details	Actions
Email address	fdelavega.provider@fi.upm.es	
Telephone number	mobile, +3462222222	

New business address

Medium: Postal address

Street: Campus de Montegancedo S/N

Postcode: 28041 City: Madrid

State / Province: Madrid Country: Spain

Create

You can *Edit* or *Remove* the contact medium by clicking on the corresponding button

Settings

Shopping Cart Francisco de la Vega

Shipping addresses Business addresses

This information is public so it may be viewed by anyone.

My business addresses

Medium	Details	Actions
Email address	fdelavega.provider@fi.upm.es	
Telephone number	mobile, +3462222222	
Postal address	Campus de Montegancedo S/N 28041 Madrid (Madrid) Spain	

New business address

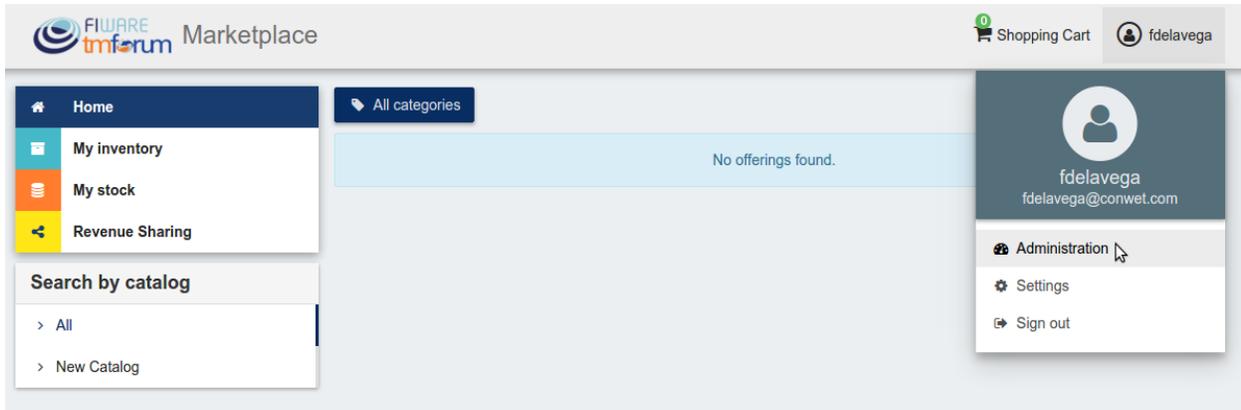
Medium: Email address

Email:

Create

Admin

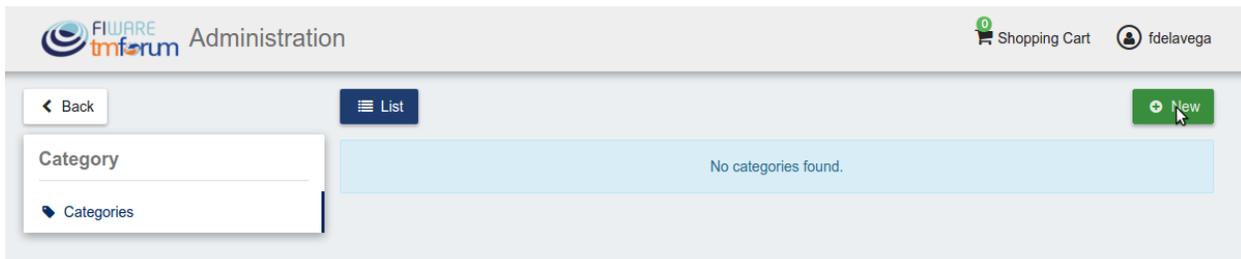
If the user of the Business API Ecosystem is an admin, he will be able to access the *Administration* section of the web portal. This section is located in the user menu.



Manage Categories

Admin users are authorized to create the system categories that can be used by *Sellers* to categorize their catalogs, products, and offerings.

To create categories, go to the *Administration* section, and click on *New*



Then, provide a name and an optional description for the category. Once the information has been included, click on *Next*, and then on *Create*

FIWARE Administration Administration Shopping Cart fdelavega

Back List New

Category Categories

New Category

1 General

2 Finish

Step 1: General

Enter a name

Enter a description (optional)

Choose a parent category

Next

FIWARE Administration Administration Shopping Cart fdelavega

Back List New

Category Categories

New Category

1 General

2 Finish

Step 2: Finish

Name

Status
 Active Launched Retired Obsolete

Description

Create

Categories in the Business API Ecosystem can be nested, so you can choose a parent category if you want while creating.

FIWARE Administration Shopping Cart fdelavega

Category: Categories

New Category

1 General
2 Finish

Step 1: General

Enter a name: VM Services

Enter a description (optional): VM Services category

Choose a parent category:

Name	Updated
Cloud Services	a minute ago

Next

Existing categories can be updated. To edit a category click on the category name.

FIWARE Administration Shopping Cart fdelavega

Category: Categories

List New

Status	Name	Updated
Launched	Cloud Services	a minute ago
Launched	Cloud Services / VM Services	a few seconds ago

Then edit the corresponding fields and click on *Update*.

FIWARE Administration Shopping Cart fdelavega

Category: Categories

List **Detail**

General

Name: VM Services

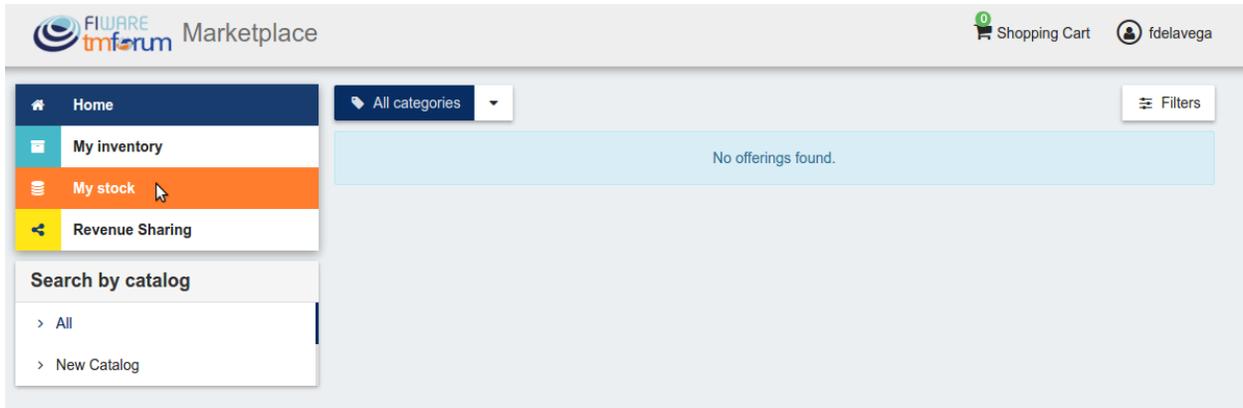
Status: Active **Launched** Retired Obsolete

Description (optional): VM Services category

Update

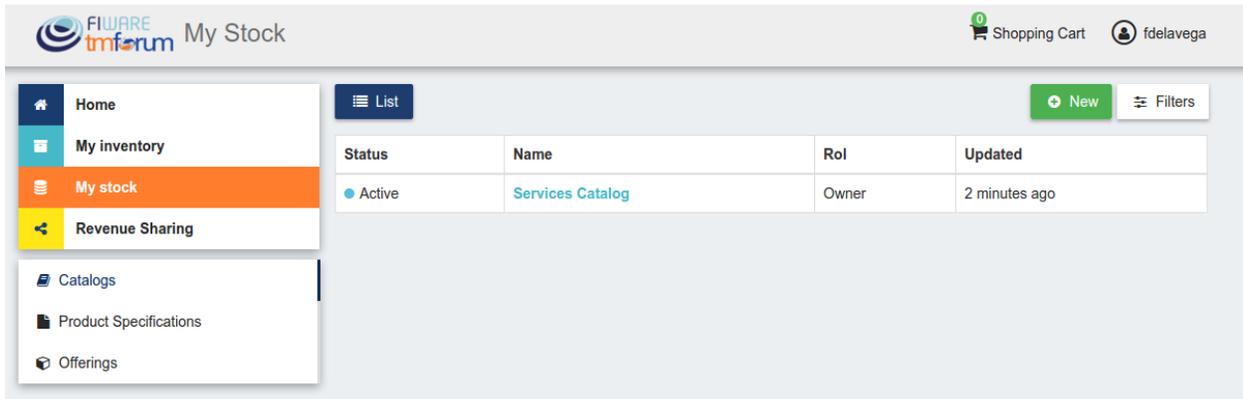
Seller

If the user of the Business API Ecosystem has the *Seller* role, he will be able to monetize his products by creating, catalogs, product specifications and product offerings. All these objects are managed accessing *My Stock* section.

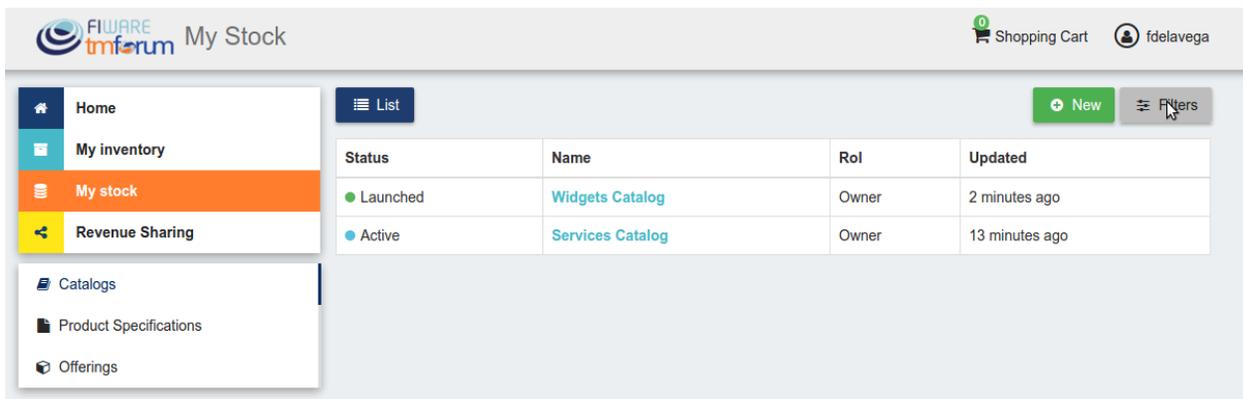


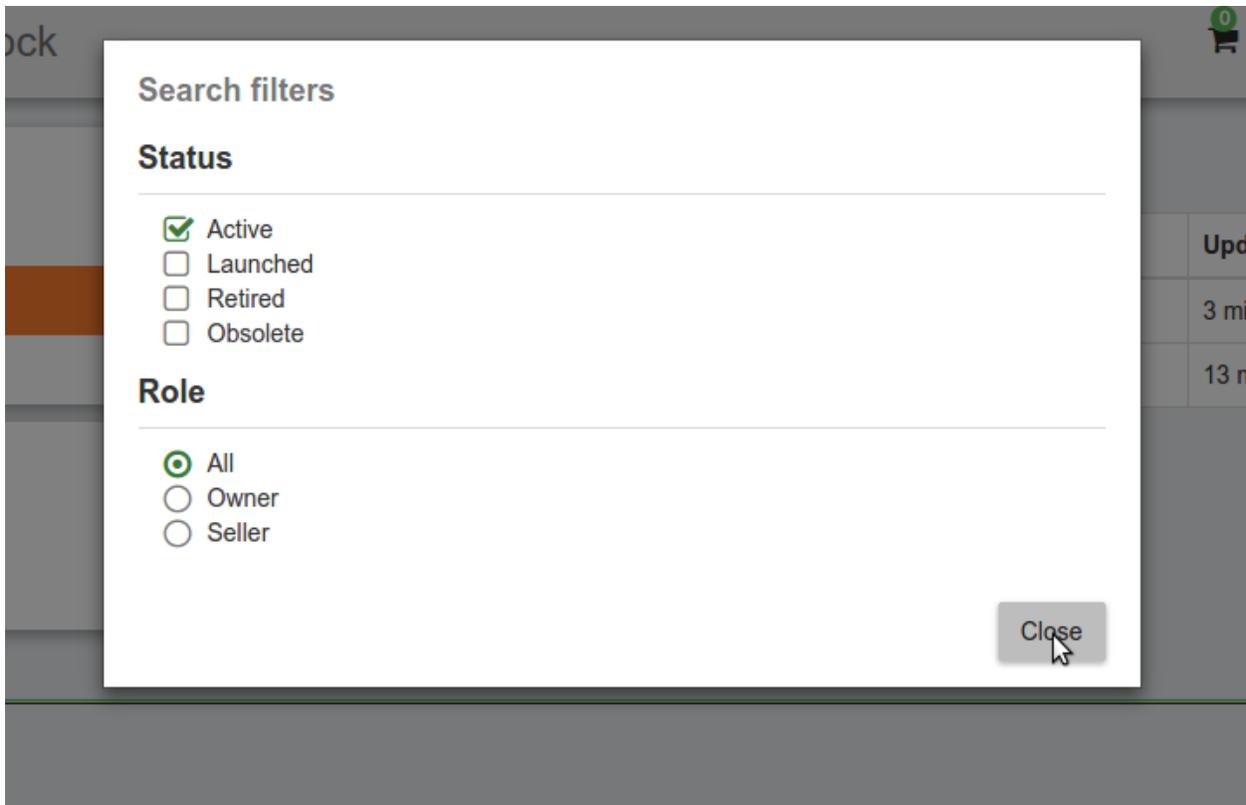
Manage Catalogs

The *Catalogs* section is the one that is open by default when the seller accesses *My Stock* section. This section contains the catalogs the seller has created.

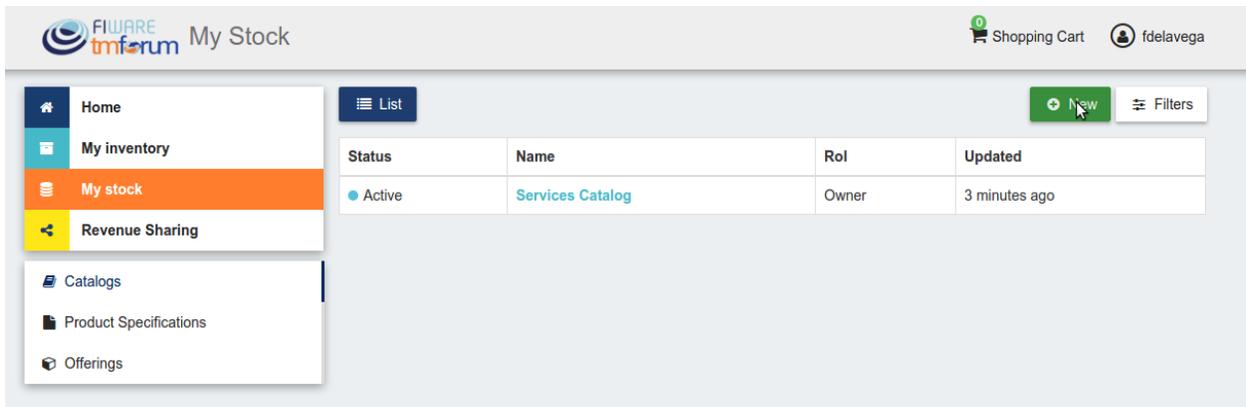


Additionally, it is possible to filter the shown catalogs by status and the role you are playing by clicking on *Filters*, choosing the required ones, and clicking on *Close*

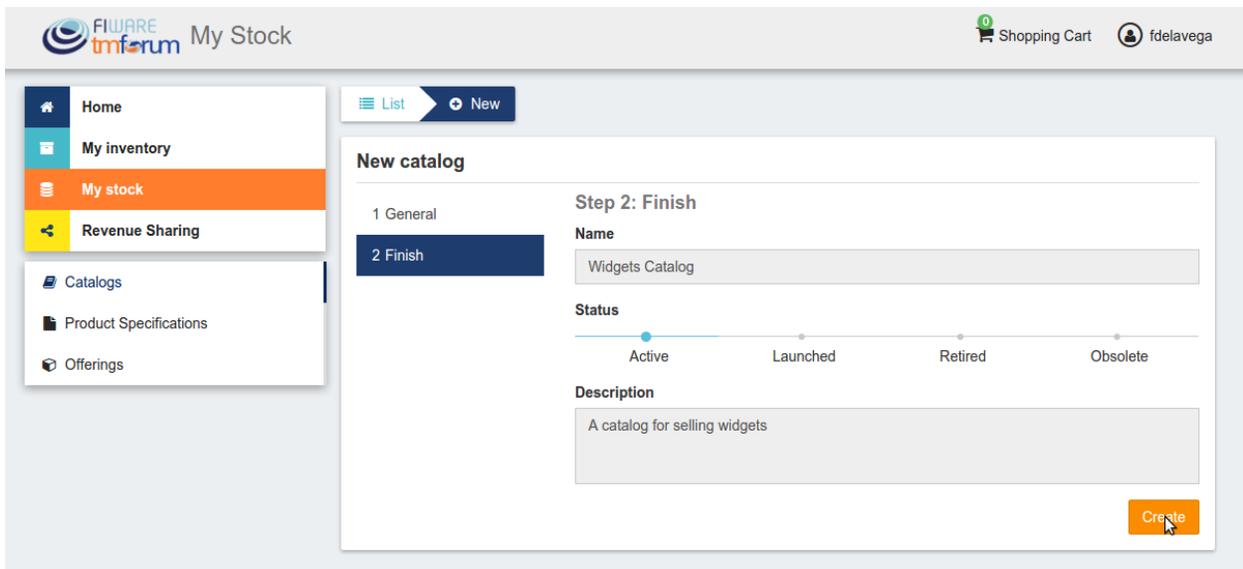
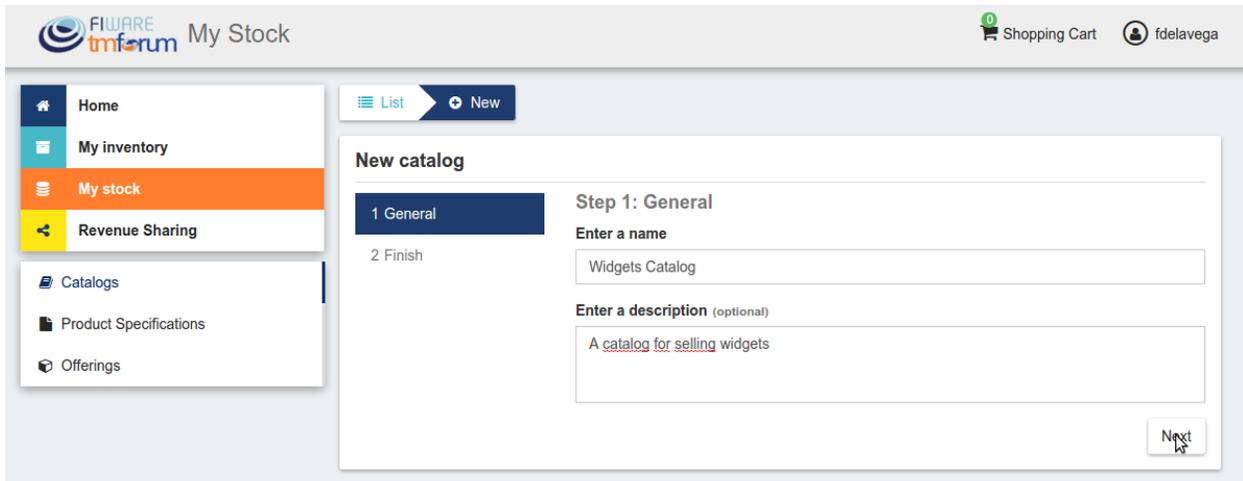




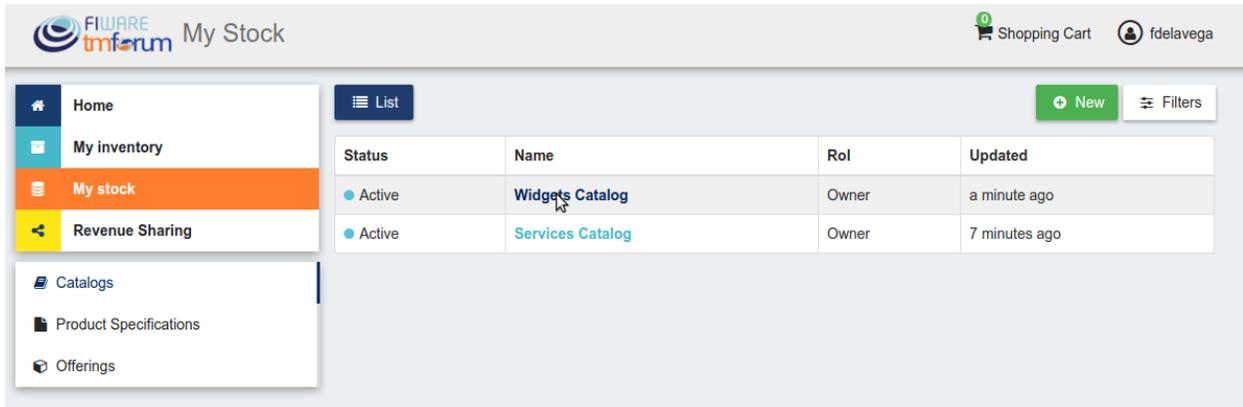
To create a new catalog click on the *New* button.



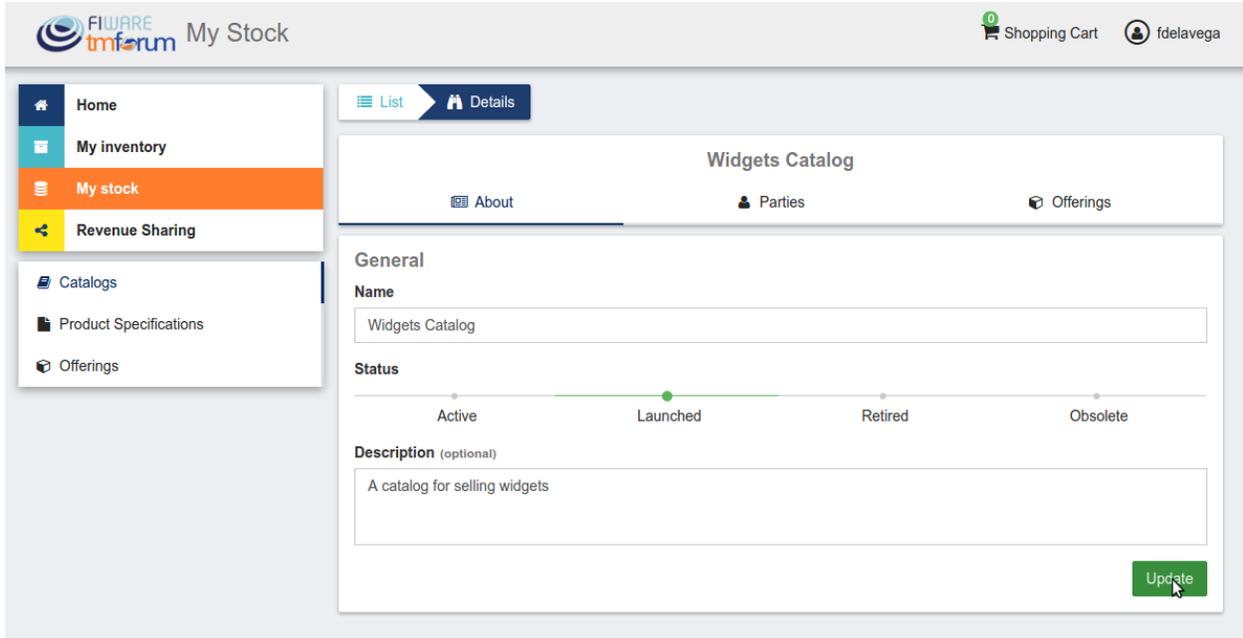
Then, provide a name and an optional description for the catalog. Once you have filled the fields, click on *Next*, and then on *Create*



Sellers can also update their catalogs. To do that, click on the name of the catalog to open the update view.

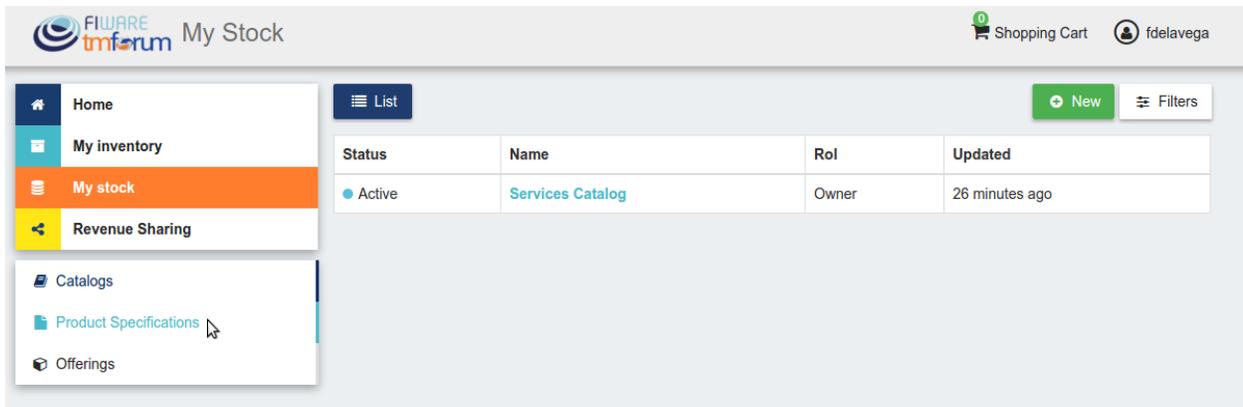


Then, update the fields you want to modify and click on *Update*. In this view, it is possible to change the *Status* of the catalog. To start monetizing the catalog, and make it appear in the *Home* you have to change its status to *Launched*



Manage Product Specifications

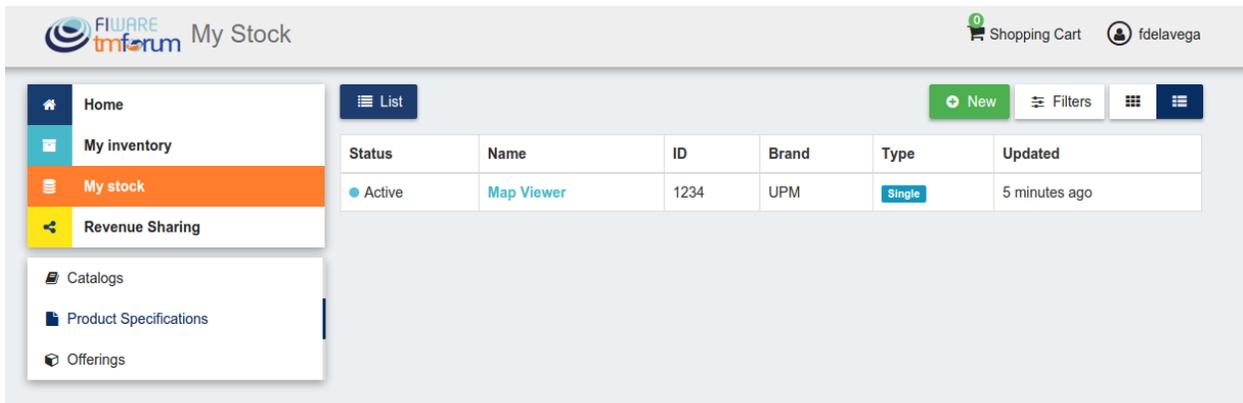
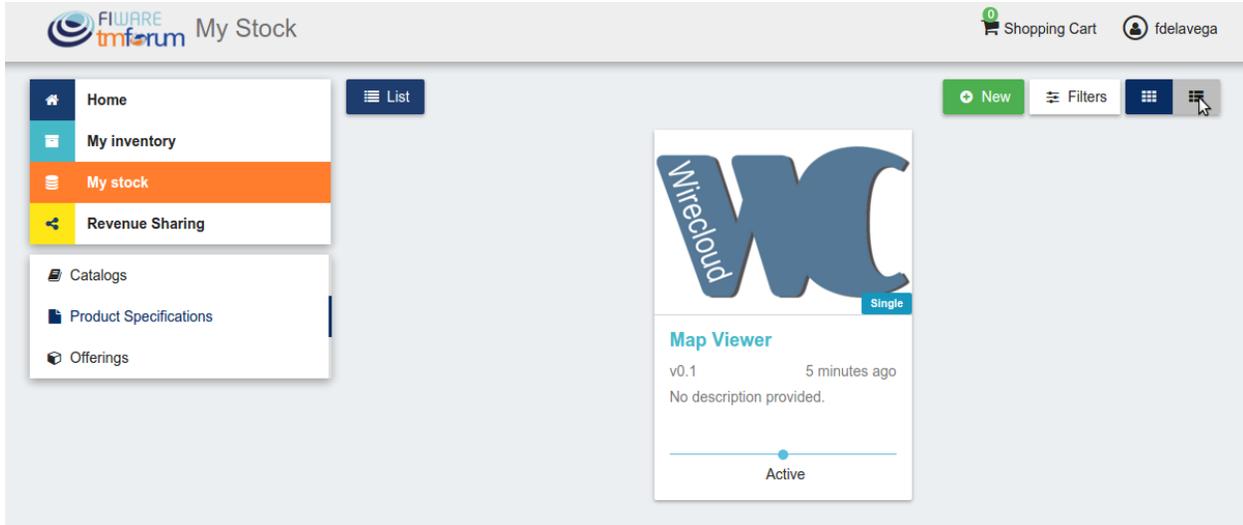
Product Specifications represent the product being offered, both digital and physical. To list your product specifications go to *My Stock* section and click on *Product Specifications*



The different product specifications can be filtered by status or by if they are bundles or not. To filter products, click on *Filters*, choose the appropriate ones, and click on *Close*

The screenshot displays the FIWARE InForm My Stock interface. The top navigation bar includes the FIWARE InForm logo, the text "My Stock", a shopping cart icon with a "0" badge, and the user name "fdelavega". A left sidebar menu contains items: Home, My inventory, My stock (highlighted), Revenue Sharing, Catalogs, Product Specifications, and Offerings. A "List" button is visible next to the sidebar. The main content area shows a "Wirecloud WC" card with a "Single" status, a "Map Viewer" section with version "v0.1", a timestamp "3 minutes ago", and the text "No description provided." Below this is a progress indicator with a blue dot and the word "Active". A "Filters" button is located in the top right of the main area. A modal window titled "Search filters" is open, showing two sections: "Status" with checkboxes for Active, Launched, Retired, and Obsolete; and "Type" with radio buttons for All (selected), Single, and Bundle. A "Close" button is in the bottom right of the modal.

Additionally, it is possible to switch between the grid view and the tabular view using the provided buttons.



To create a new product specification click on *New*



In the displayed view, provide the general information of the product spec. including its name, version, and an optional description. In addition, you have to include the product brand (Your brand), and an ID number which identifies the product in your environment. Then, click on *Next*.

The screenshot shows the 'New product' form in Step 1: General. The left sidebar contains navigation options: Home, My inventory, My stock (highlighted), Revenue Sharing, Catalogs, Product Specifications, and Offerings. The main content area has a 'List' button and a 'New' button. The form steps are listed on the left: 1 General (selected), 2 Bundle, 3 Assets, 4 Characteristics, 5 Attachments, 6 Relationships, 7 Terms & Conditions, and 8 Finish. The 'Step 1: General' form includes the following fields:

- Enter a name:** Basic Chart
- Enter a version:** 0.1
- Enter a brand:** UPM
- Enter an ID Number:** 1234
- Enter a description (optional):** A basic widget for showing charts

A 'Next' button is located at the bottom right of the form.

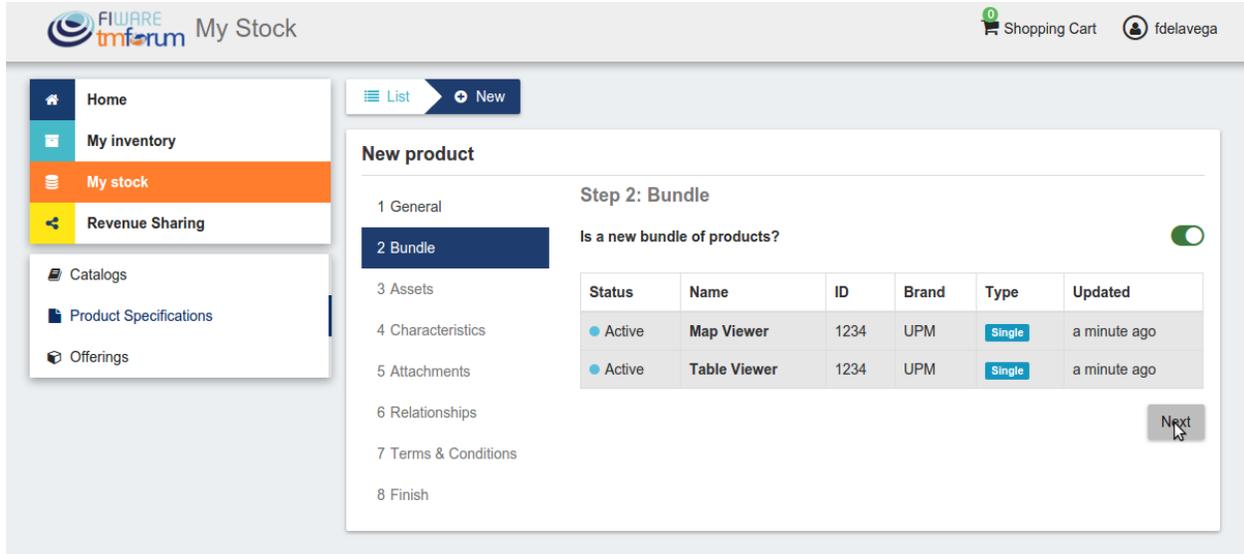
In the next step, you can choose whether your product specification is a bundle or not. Product bundles are logical containers that allow you to sell multiple products as if it were a single one. Once you have selected the right option click on *Next*

The screenshot shows the 'New product' form in Step 2: Bundle. The left sidebar and main content area are the same as in the previous screenshot. The 'Step 2: Bundle' form includes the following field:

- Is a new bundle of products?:** A toggle switch that is currently turned off.

A 'Next' button is located at the bottom right of the form.

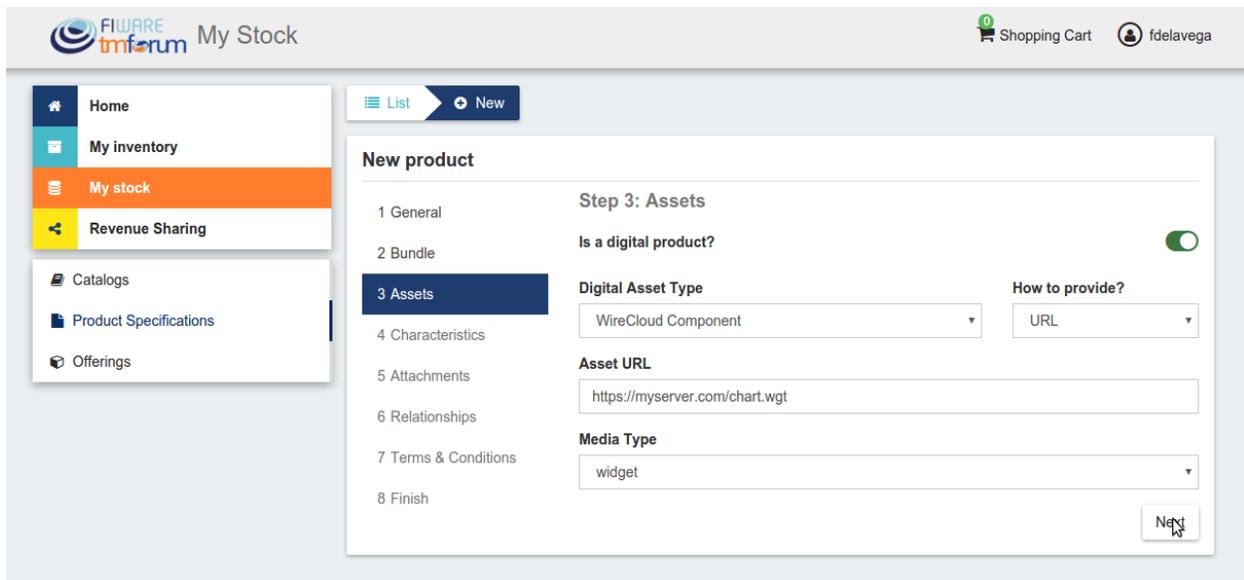
If you have decided to create a bundle, you will be required to choose 2 or more product specs to be included in the bundle.



In the next step you can choose if your product is a digital product. If this is the case, you will be required to provide the asset.

Note: If you are creating a product bundle, you will not be allowed to provide a digital asset since the offered ones will be the included in the bundled products

For providing the asset, you have to choose between the available asset types, choose how to provide the asset between the available options, provide the asset, and include its media type.



FIWARE **imforum** My Stock Shopping Cart fdelavega

Home
My inventory
My stock
Revenue Sharing
Catalogs
Product Specifications
Offerings

List New

New product

- General
- Bundle
- 3 Assets**
- Characteristics
- Attachments
- Relationships
- Terms & Conditions
- Finish

Step 3: Assets

Is a digital product?

Digital Asset Type: WireCloud Component How to provide?: FILE

Asset File: CoNWeT_and-filter_0.3.4.wgt

Media Type:

Next

The next step in the creation of a product is including its characteristics. For including a new characteristic click on *New Characteristic*

FIWARE **imforum** My Stock Shopping Cart fdelavega

Home
My inventory
My stock
Revenue Sharing
Catalogs
Product Specifications
Offerings

List New

New product

- General
- Bundle
- Assets
- 4 Characteristics**
- Attachments
- Relationships
- Terms & Conditions
- Finish

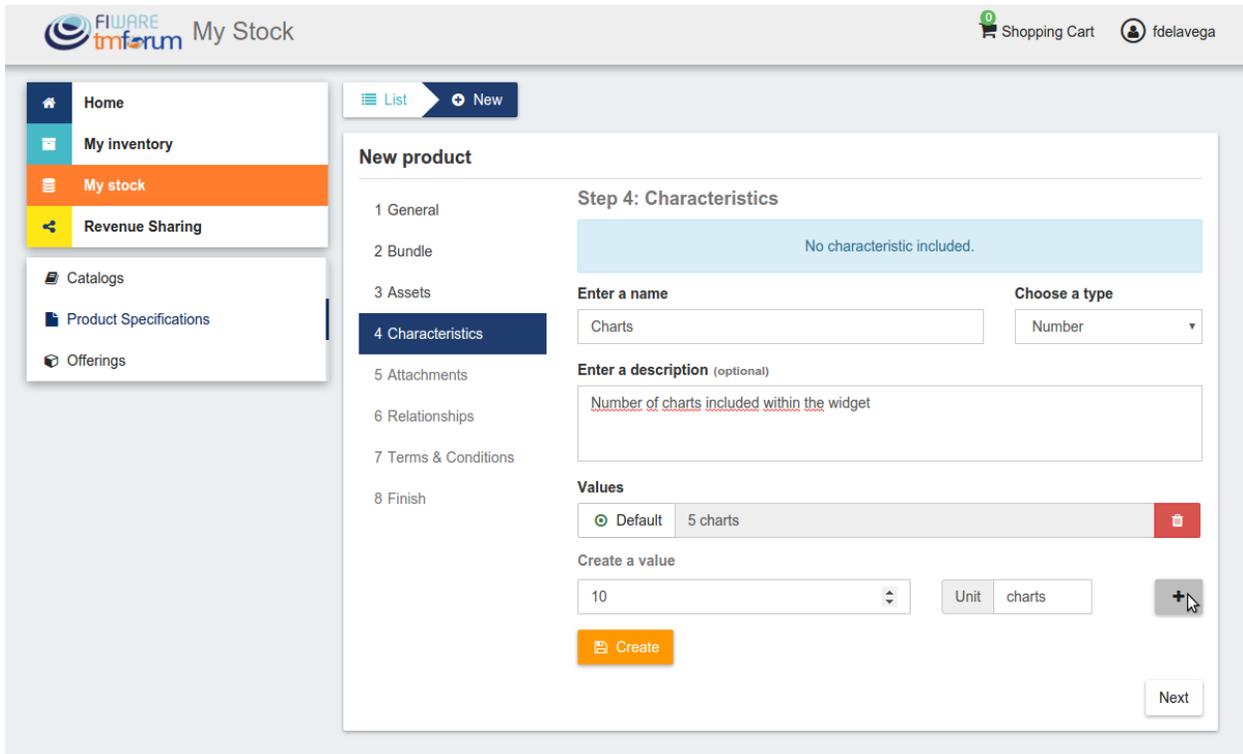
Step 4: Characteristics

No characteristic included.

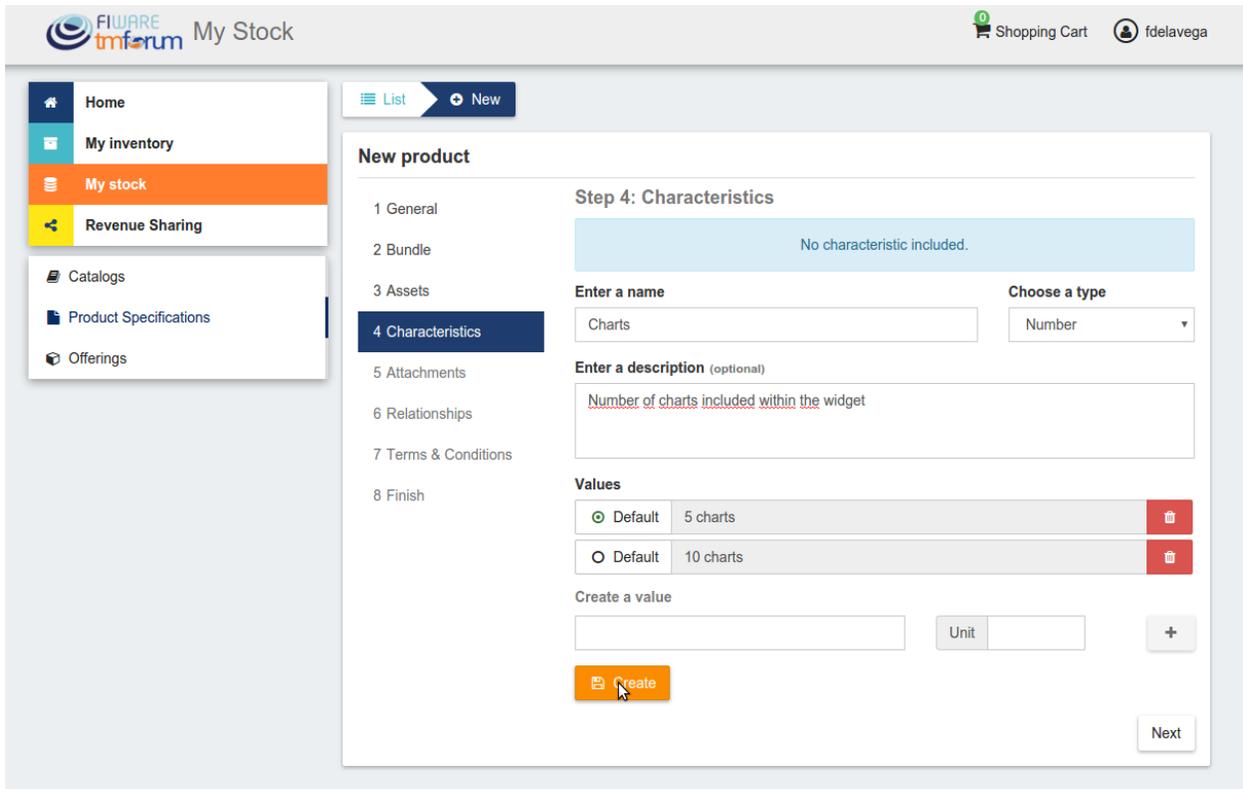
+ New Characteristic

Next

In the form, include the name, the type (string or number) and an optional description. Then create the values of the characteristic by filling the *Create a value* input and clicking on +.



Once you have included all the characteristic info, save it clicking on *Create*



Once you have included all the required characteristics click on *Next*

New product

Step 4: Characteristics

#	Name	Type	Values	Default	Delete
1	Charts	Number	5 charts, 10 charts	5 charts	

+ New Characteristic

Next

In the next step you can include a picture for your product spec. You have two options, providing an URL pointing to the picture or directly uploading it. Once provided click *Next*

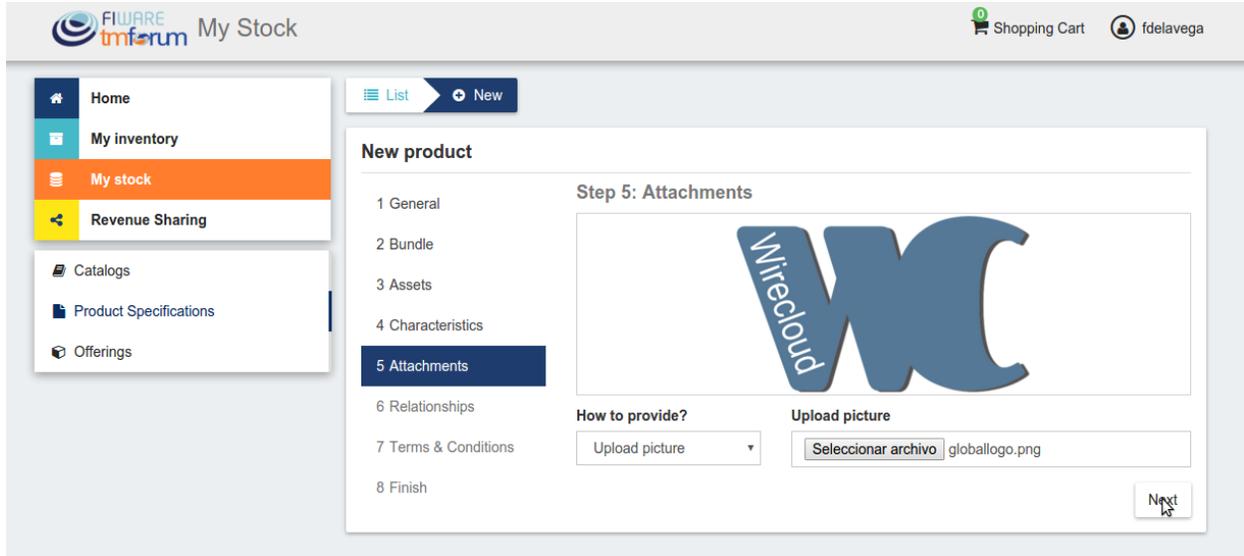
New product

Step 5: Attachments

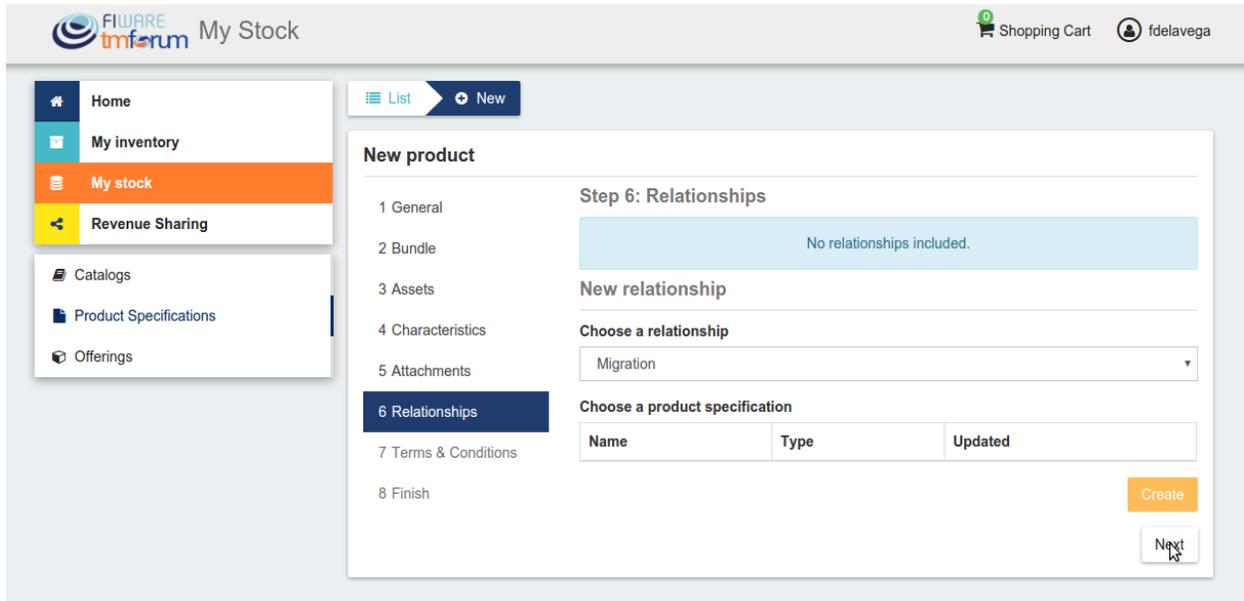
How to provide? Include picture URL

Include picture URL

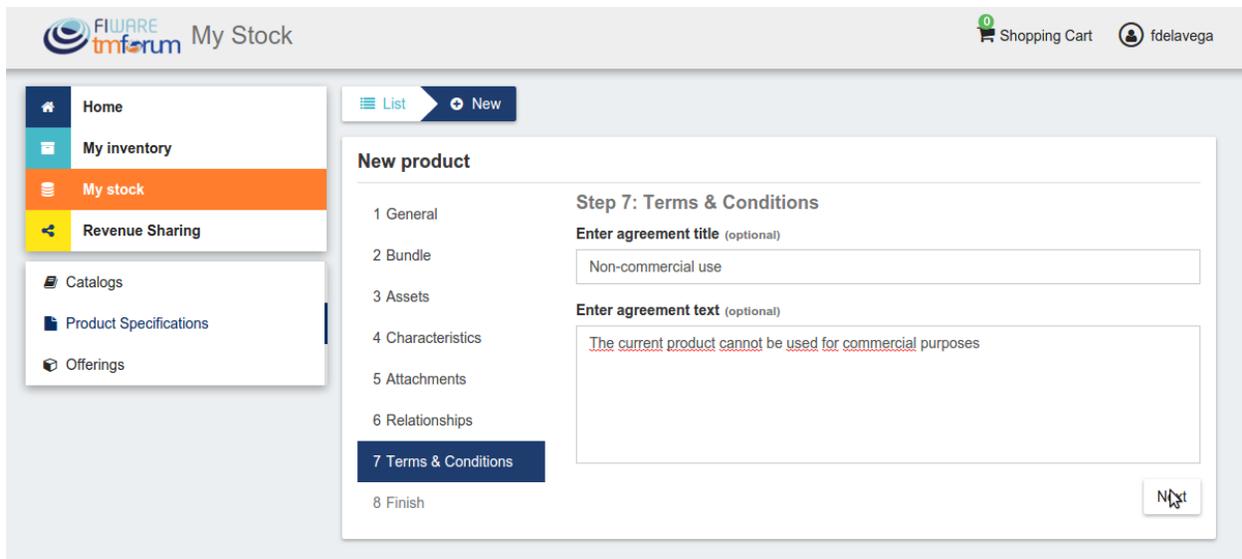
Next



Then, you can specify relationships of the product you are creating with other of your product specs.



In the last step, you can specify the terms and conditions that apply to your product and that must be accepted by those customers who want to acquire it. To do that, include a title and a text for your terms and click on *Next*. Note that the terms and conditions are not mandatory.



Once done click on *Next* and then on *Create*

New product

Step 8: Finish

1 General

2 Bundle

3 Assets

4 Characteristics

5 Attachments

6 Relationships

7 Terms & Conditions

8 Finish

Name: Basic Chart

Version: 0.1

Status: Active (Progress bar: Active, Launched, Retired, Obsolete)

Brand: UPM

ID Number: 1234

Description: A widget for showing basic charts

Characteristics

#	Name	Type	Values	Default
1	Charts	Number	5 charts, 10 charts	5 charts

Attachments

Picture URL: http://antares.ls.fi.upm.es:8004/charging/media/assets/fdelavega/globallogo.png

Agreement title: Non-commercial use

Enter agreement text: The current product cannot be used for commercial purposes

Create

Sellers can update their products. To do that click on the product specification to be updated.

Map Viewer

v0.1 an hour ago

No description provided.

Active

Basic Chart

v0.1 2 minutes ago

A basic charts widget

Active

Update the required values and click on *Update*. Note that for start selling an offering that includes the product specification you will be required to change its status to *Launched*

The screenshot shows the 'My Stock' interface with the 'Basic Chart' details form. The form includes the following fields and values:

- Name:** Basic Chart
- Version:** 0.1
- Status:** Launched (indicated by a green dot on a progress bar between Active and Retired)
- Brand:** UPM
- ID Number:** 1234
- Description (optional):** A basic charts widget

An 'Update' button is visible at the bottom right of the form.

Manage Product Offerings

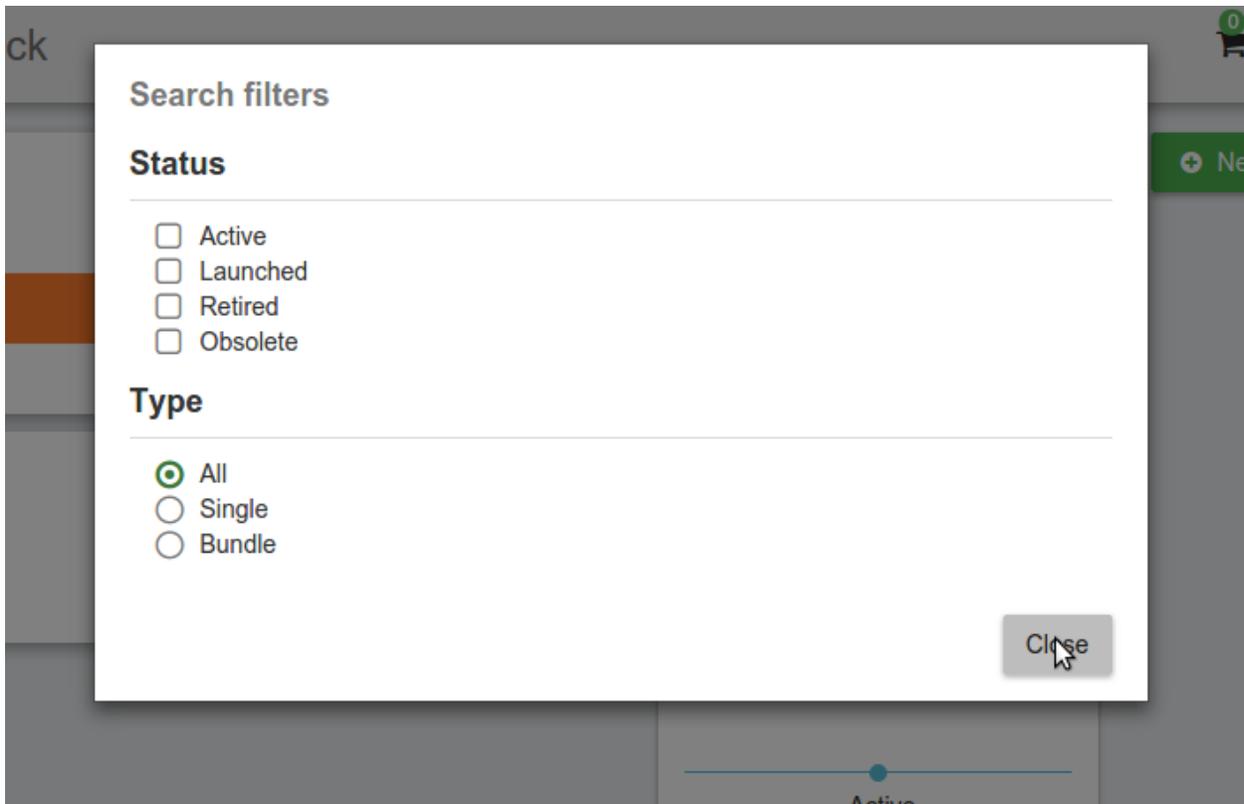
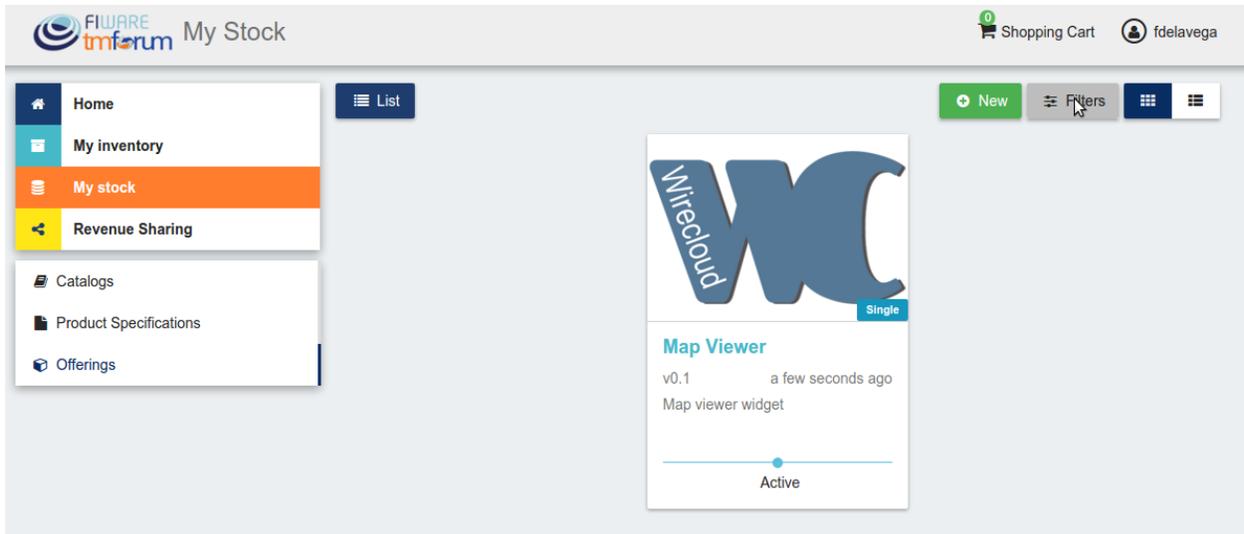
Product Offerings are the entities that contain the pricing models and revenue sharing info used to monetize a product specification. To list your product offerings, go to *My Stock* section and click on *Offerings*

The screenshot shows the 'My Stock' interface with the 'Offerings' list. The list contains the following entries:

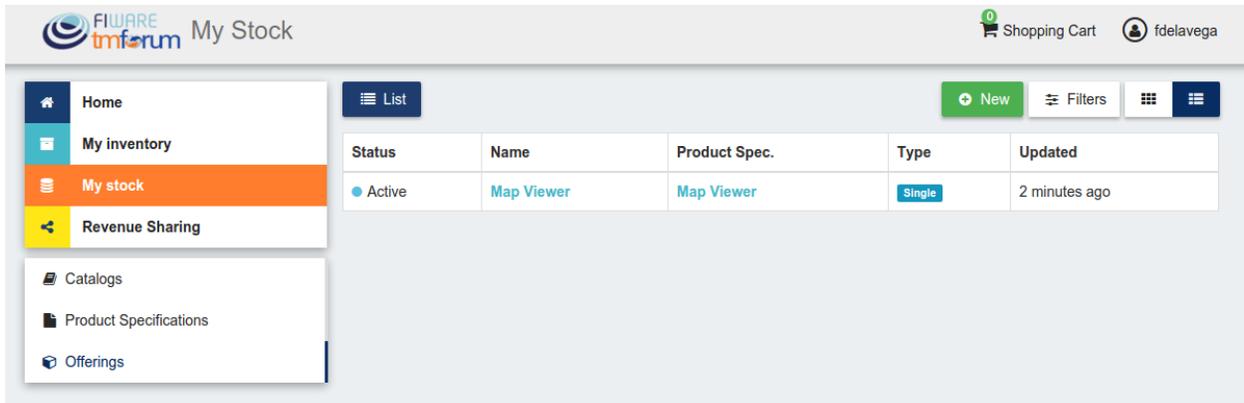
Status	Name	Rol	Updated
Launched	Widgets Catalog	Owner	17 hours ago
Active	Services Catalog	Owner	17 hours ago

The interface also shows a 'List' button, a 'New' button, and a 'Filters' button.

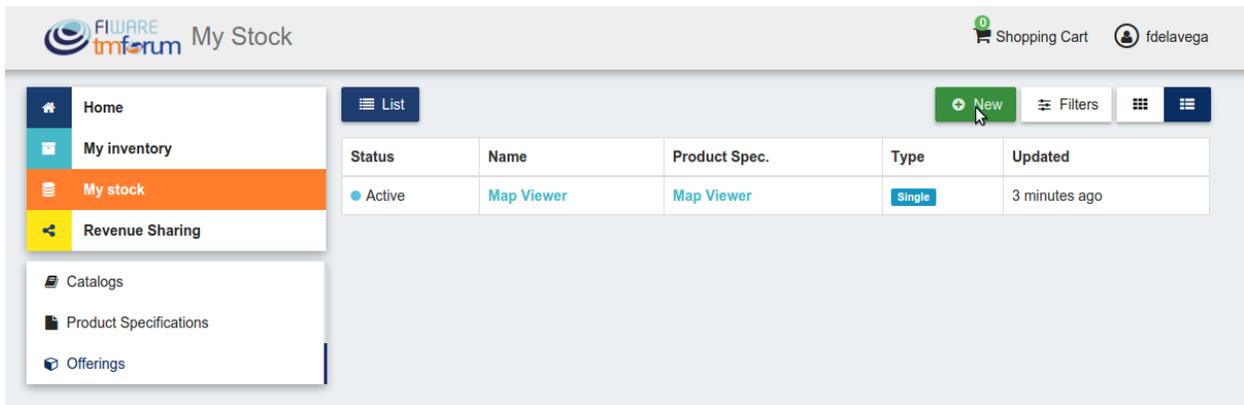
The existing product offerings can be filtered by status or by if they are a bundle or not. To filter offerings click on *Filters* choose the appropriate ones and click on *Close*



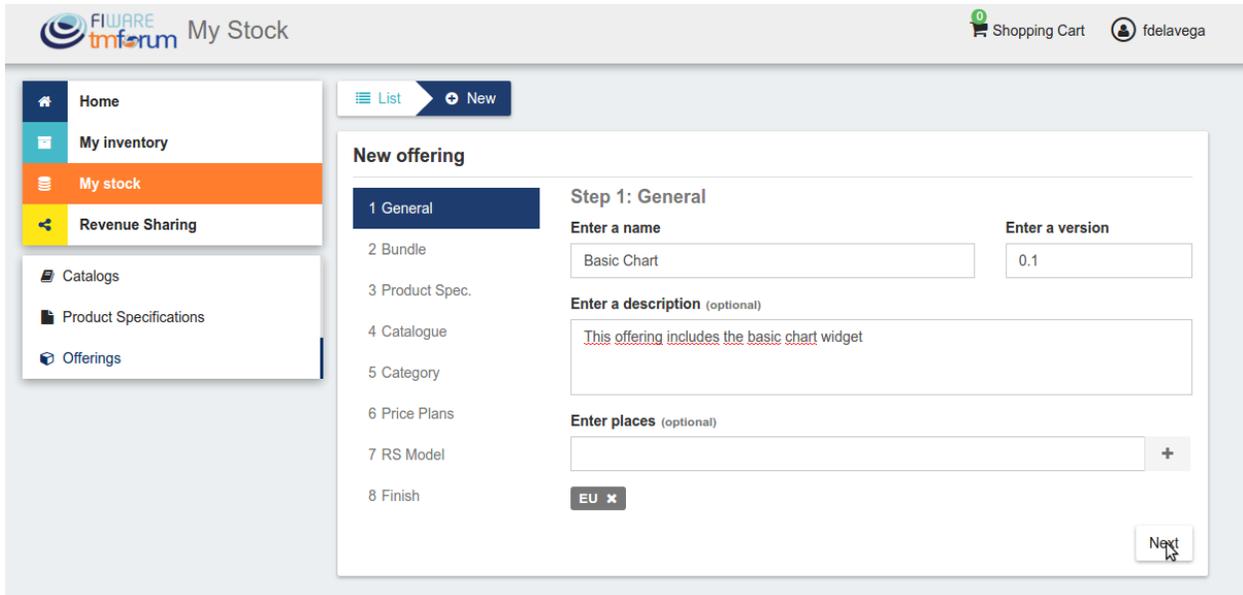
Additionally, it is possible to switch between the grid view and the tabular view by clicking on the specific button.



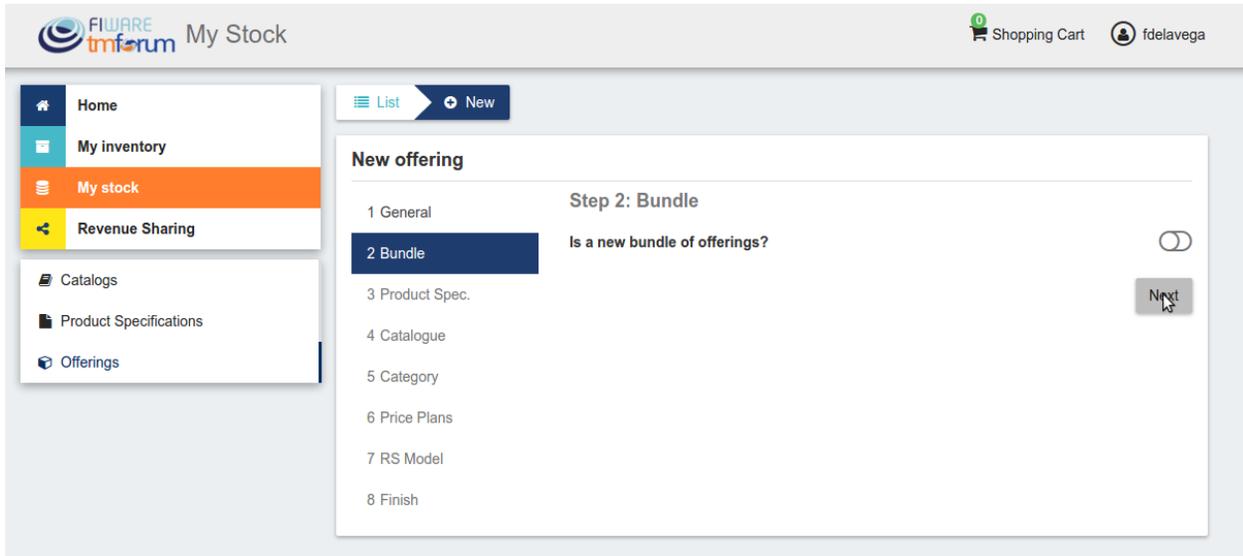
To create a new offering click on *New*



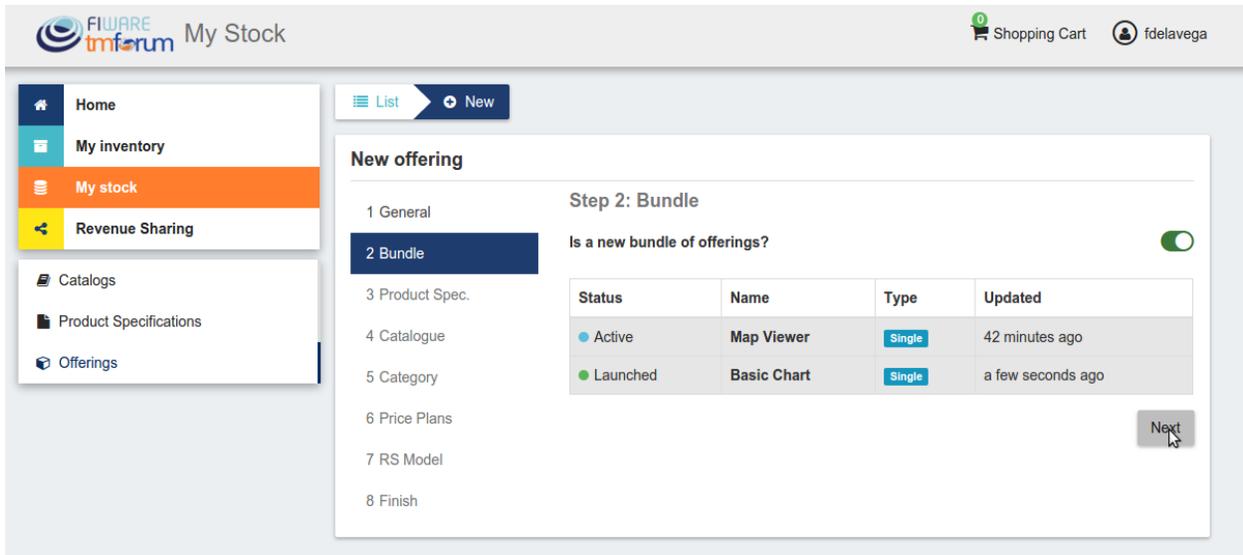
In the displayed form, include the basic info of the offering. Including, its name, version, an optional description, and an optional set of places where the offering is available. Once the information has been provided click on *Next*



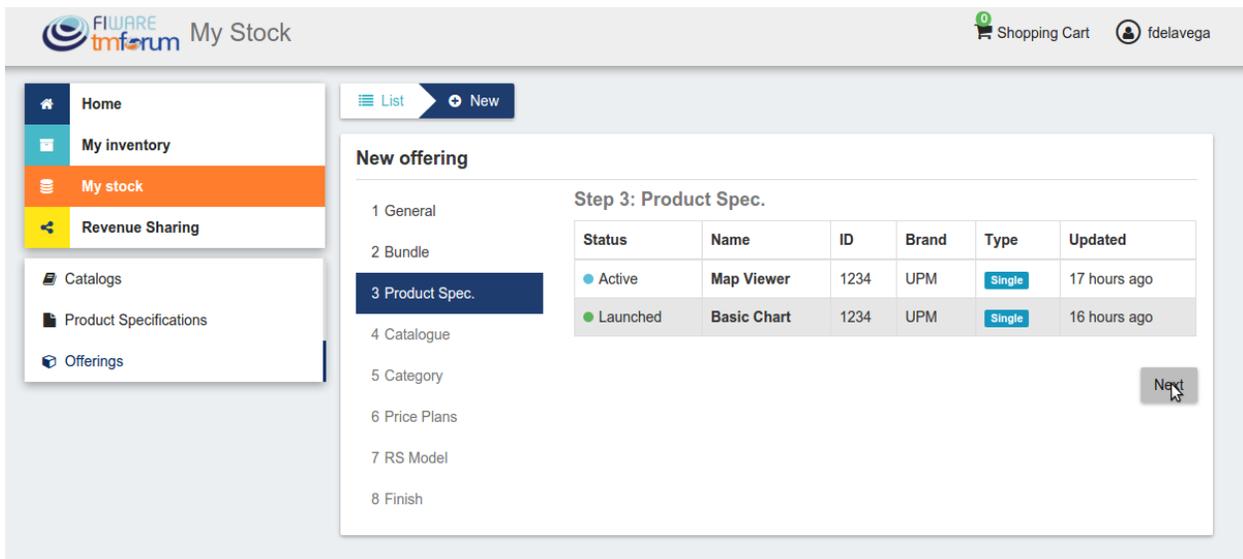
In the next step, you can choose whether your offering is a bundle or not. In this case, offering bundles are logical containers that allow you to provide new pricing models when a set of offerings are acquired together. Once selected click on *Next*



If you want to create a bundle you will be required to include at least two bundled offerings.

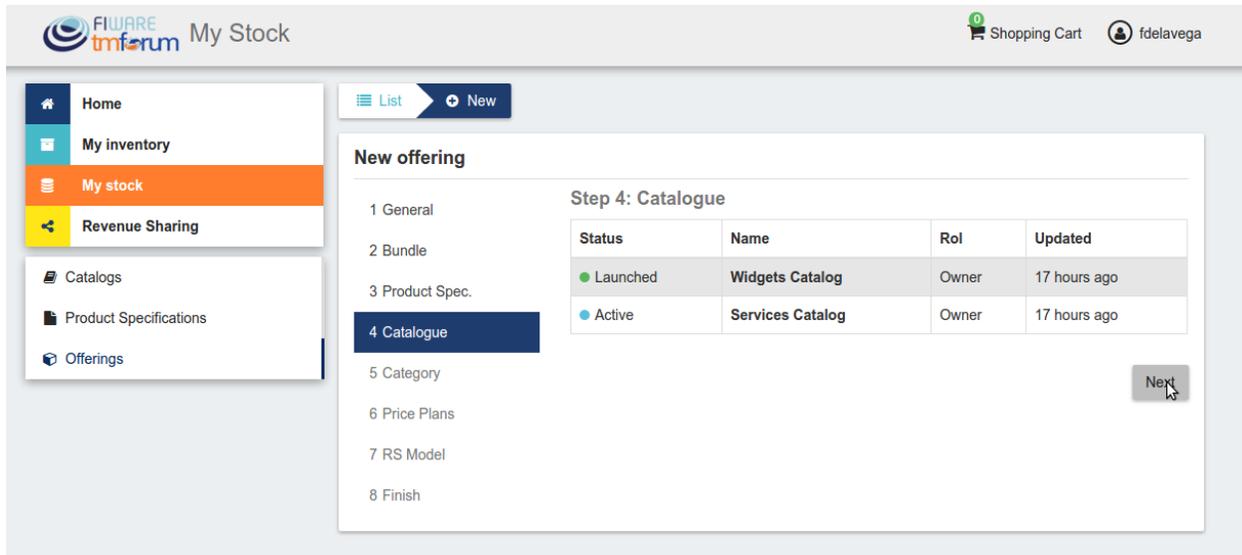


In the next step you have to select the product specification that is going to be monetized in the current offering. Once selected click on *Next*.

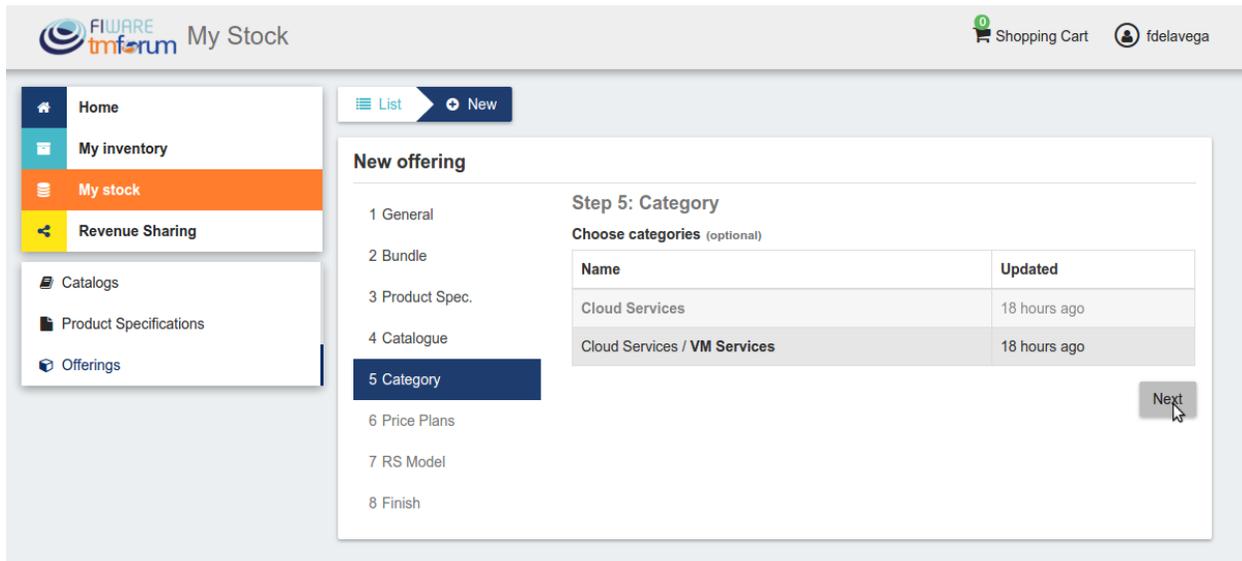


Note: If you are creating an offering bundle, you will not be allowed to include a product specification

Then, you have to select the catalog where you want to publish you offering and click on *Next*

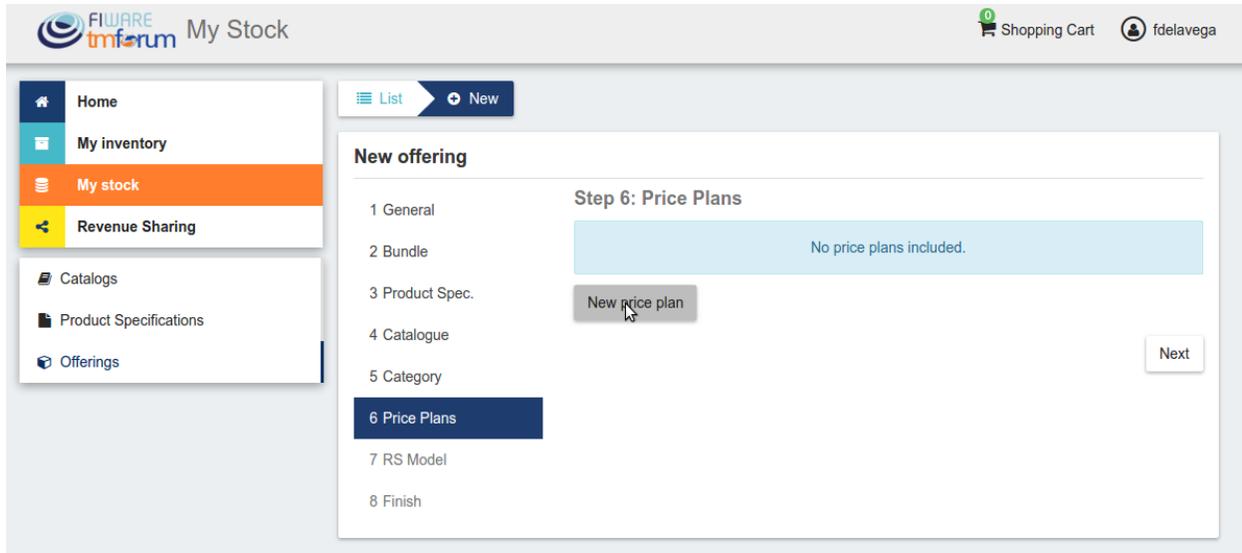


In the next step, you can optionally choose categories for you offering. Once done, click on *Next*



The next step is the more important for the offering. In the displayed form you can create different price plans for you offering, which will be selectable by customers when acquiring the offering. If you do not include any price plan the offering in considered free.

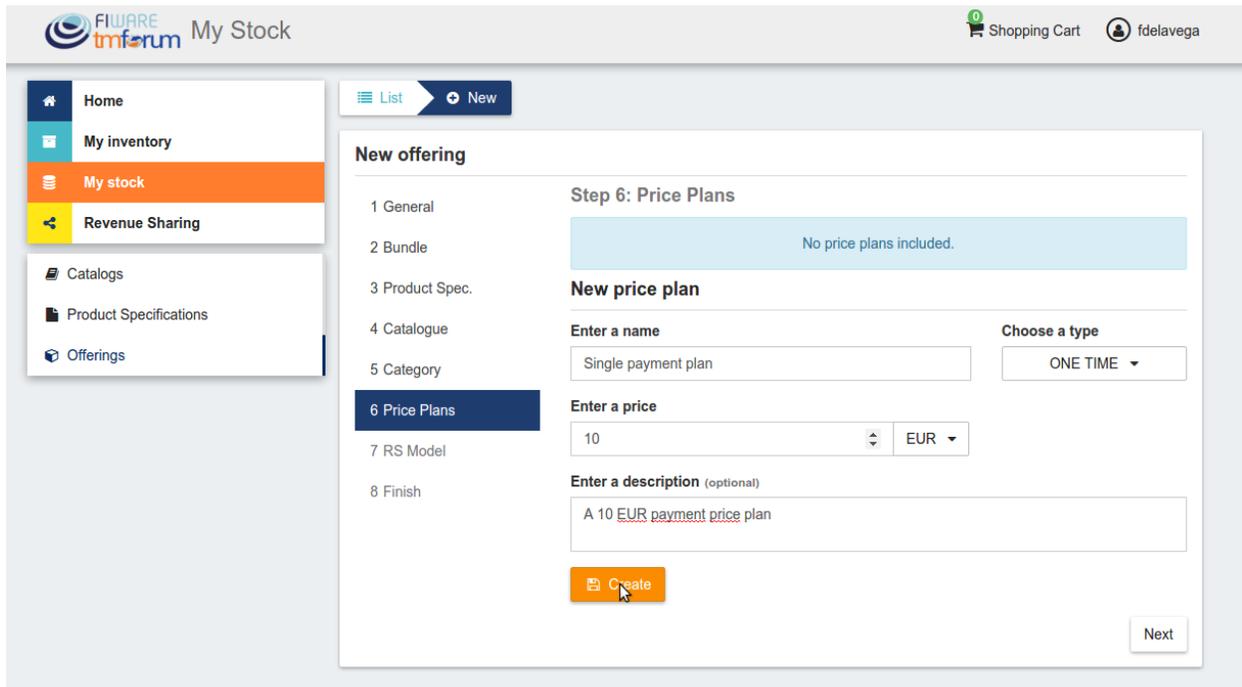
To include a new price plan the first step is clicking on *New Price Plan*



For creating the price plan, you have to provide a name, and an optional description. Then, you have to choose the type of price plan between the provided options.

The available types are: *one time* for payments that are made once when purchasing the offering, *recurring* for charges that are made periodically (e.g a monthly payment), and *usage* for charges that are calculated applying the pricing model to the actual usage made of the acquired service.

If you choose *one time*, you have to provide the price and the currency.



If you choose *recurring*, you have to provide the price, the currency, and the period between charges.

The screenshot shows the 'My Stock' application interface. On the left is a navigation menu with options: Home, My inventory, My stock (highlighted), Revenue Sharing, Catalogs, Product Specifications, and Offerings. The main content area is titled 'New offering' and shows a progress list from 1 to 8, with '6 Price Plans' selected. The 'Step 6: Price Plans' section contains a table with one entry:

Name	Description	Price	Actions
Single payment plan	A 10 EUR payment price plan	10 EUR	 

Below the table, the 'New price plan' form includes fields for:

- Enter a name: Subscription plan
- Choose a type: RECURRING
- Enter a price: 1 EUR
- Choose a charge period: MONTHLY
- Enter a description (optional): A monthly payment of 1 EUR

 A 'Create' button and a 'Next' button are at the bottom.

If you choose usage, you have to provide the unit to be accounted, the currency, and the price per unit

This screenshot shows the 'My Stock' application with the 'New offering' form at Step 6: Price Plans. The progress list on the left is the same as in the previous screenshot. The 'Step 6: Price Plans' table now contains two entries:

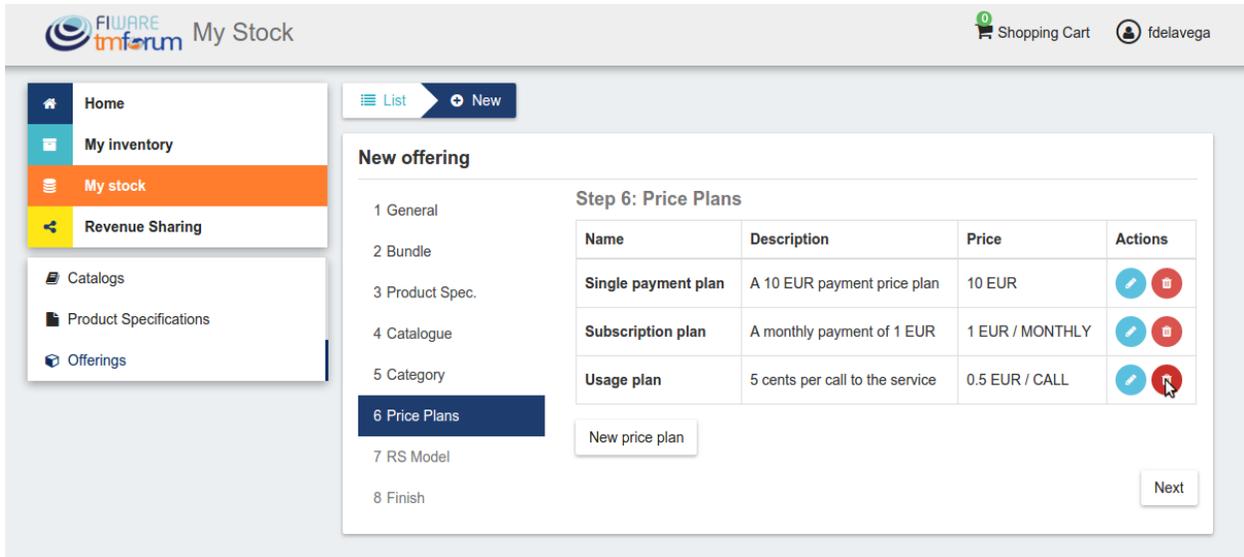
Name	Description	Price	Actions
Single payment plan	A 10 EUR payment price plan	10 EUR	 
Subscription plan	A monthly payment of 1 EUR	1 EUR / MONTHLY	 

The 'New price plan' form below is updated to:

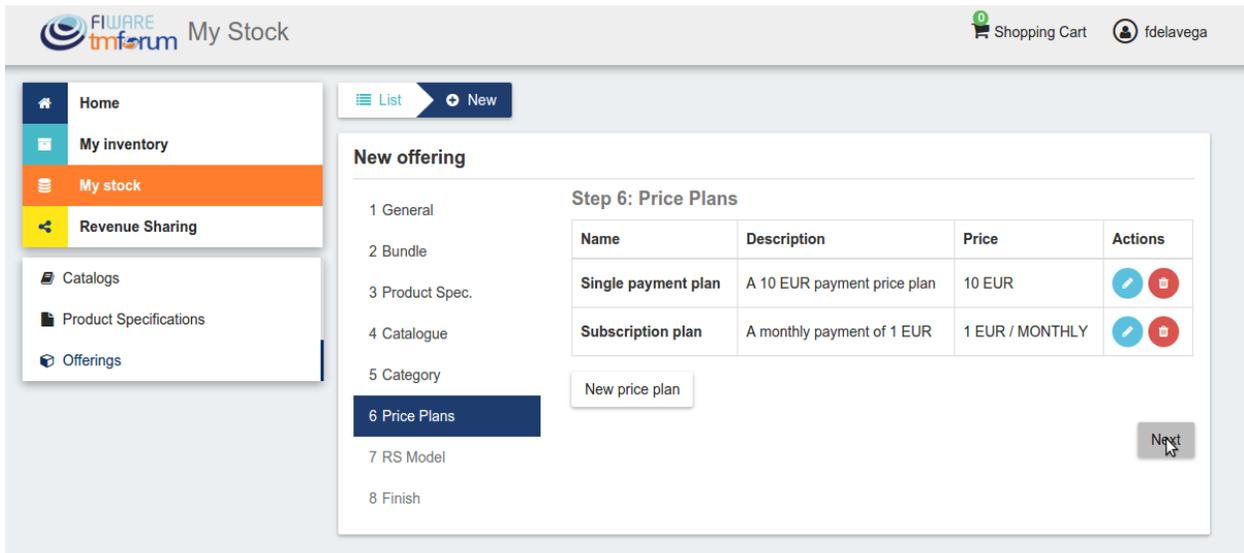
- Enter a name: Usage plan
- Choose a type: USAGE
- Enter a price: 0.5 EUR
- Enter a unit: call
- Enter a description (optional): 5 cents per call to the service

 The 'Create' and 'Next' buttons remain at the bottom.

You can update or remove plans by clicking on the corresponding action button.



Once you have created your pricing model click on *Next*



In the last step of the process, you have to choose the revenue sharing model to be applied to your offering between the available ones. Once done, click on *Next* and then on *Create*.

The screenshot shows the 'My Stock' dashboard with a sidebar menu containing: Home, My inventory, My stock (highlighted), Revenue Sharing, Catalogs, Product Specifications, and Offerings. The main content area is titled 'New offering' and shows a progress list from 1 General to 8 Finish, with '7 RS Model' selected. A table for 'Step 7: RS Model' is displayed:

Product Class	Platform Percentage	Provider Percentage	N° Stakeholders
defaultRevenue	30	70	0

A 'Next' button is visible at the bottom right of the table.

The screenshot shows the '8 Finish' step of the offering process. The sidebar menu is the same as in the previous screenshot. The main content area shows the '8 Finish' step selected in the progress list. Below the progress list, there are several sections:

- Places:** A button labeled 'EU'.
- Product Spec.:** A table with columns: Status, Name, Type, Updated.

Status	Name	Type	Updated
● Launched	Basic Chart	Single	16 hours ago
- Catalogue:** A table with columns: Status, Name, Rol, Updated.

Status	Name	Rol	Updated
● Launched	Widgets Catalog	Owner	17 hours ago
- Categories:** A table with columns: Name, Updated.

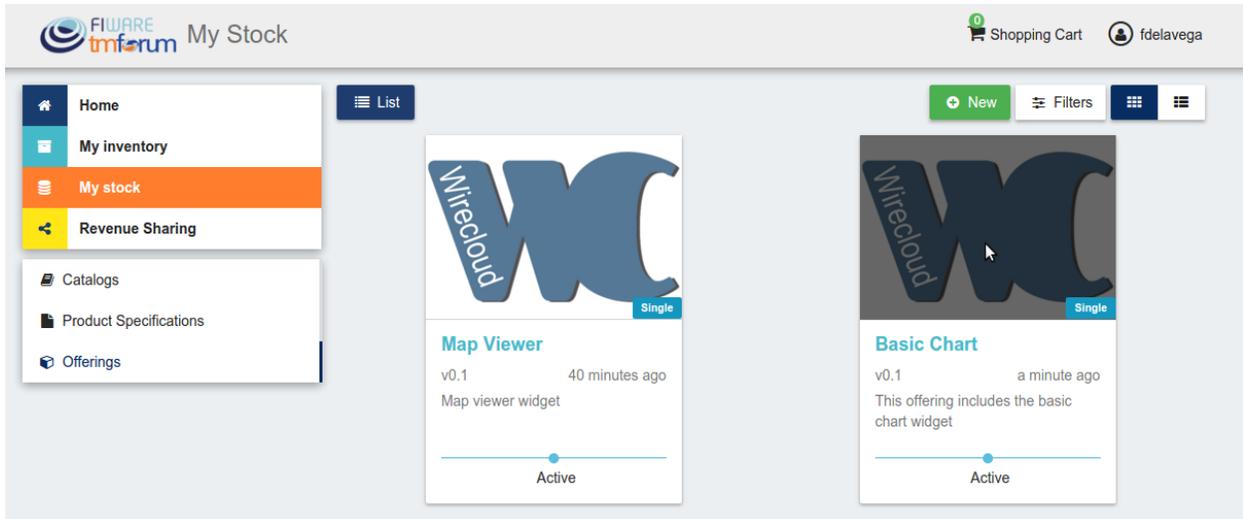
Name	Updated
Cloud Services / VM Services	18 hours ago
- Price plans:** A table with columns: #, Name, Description, Price.

#	Name	Description	Price
1	Single payment plan	A 10 EUR payment price plan	10 EUR
2	Subscription plan	A monthly payment of 1 EUR	1 EUR / MONTHLY
- Revenue Sharing Model:** A table with columns: Product Class, Platform Percentage, Provider Percentage, N° Stakeholders.

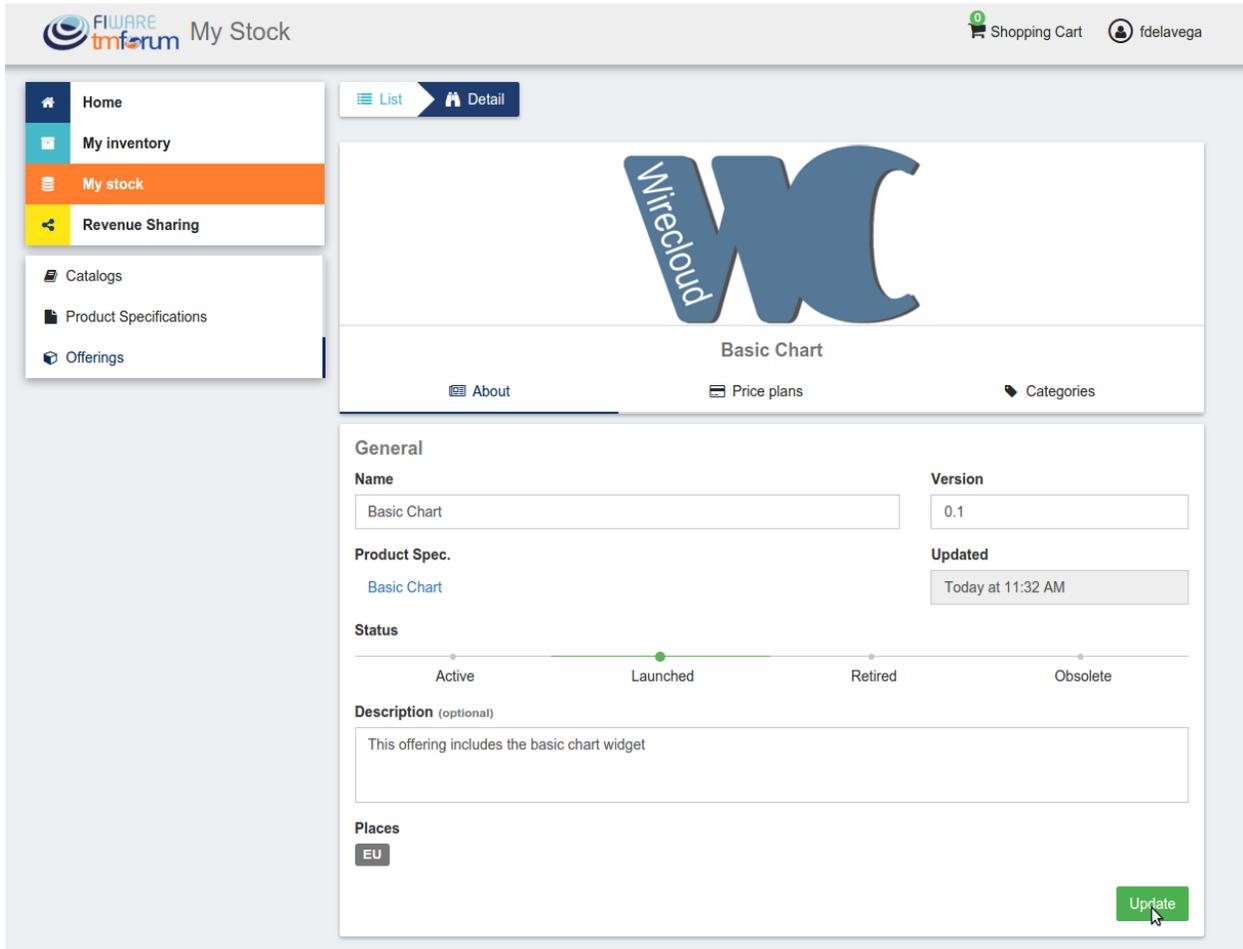
Product Class	Platform Percentage	Provider Percentage	N° Stakeholders
defaultRevenue	30	70	0

A 'Create' button is located at the bottom right of the main content area.

Sellers can also edit their offerings. To do that click on the offering to be updated.



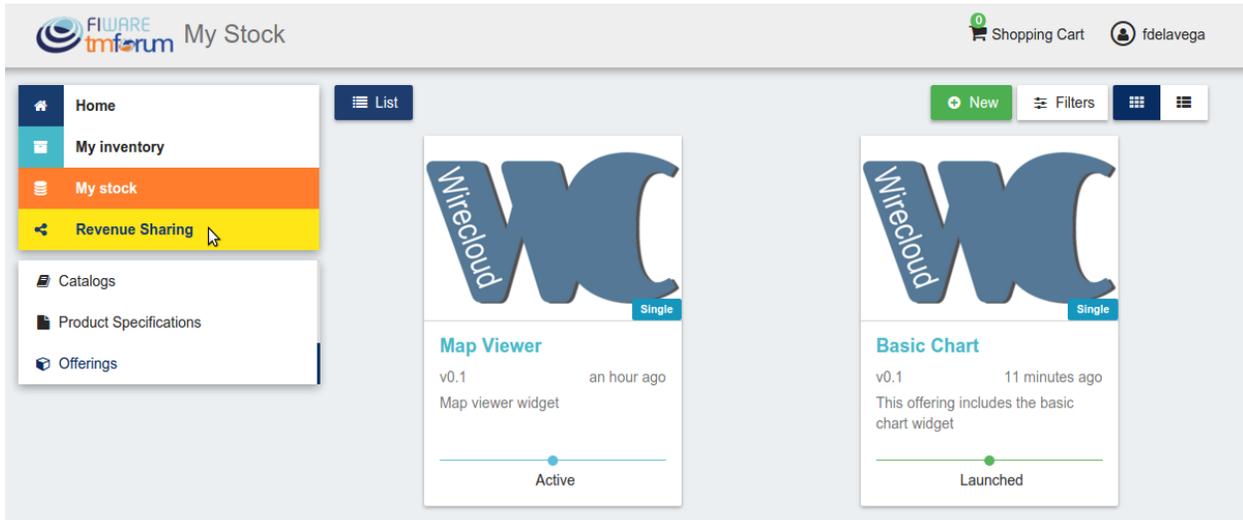
In the displayed form, change the fields you want to edit and click on *Update*. Note that for start selling you offering you have to update its status to *Launched*



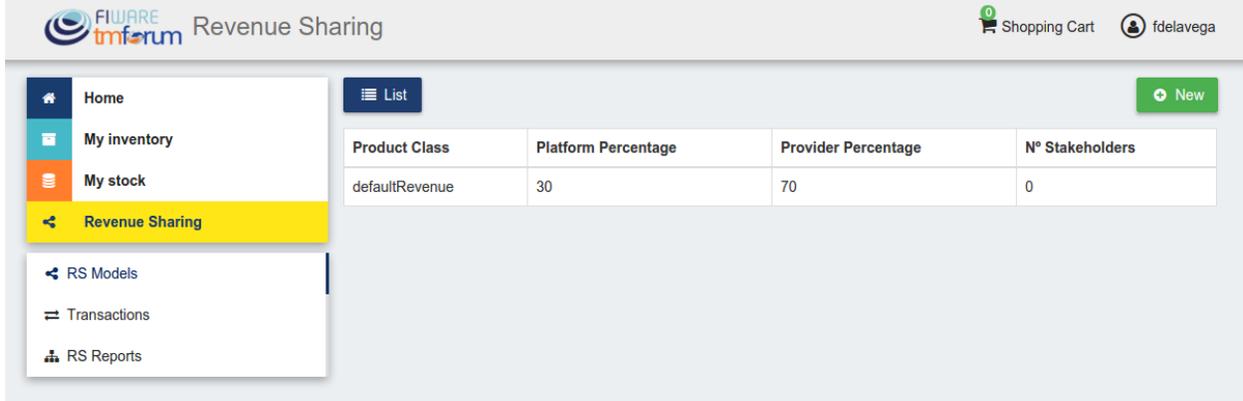
Manage Revenue Sharing Models

Revenue Sharing Models specify how the revenues generated by an offering or set of offerings must be distributed between the owner of the Business API Ecosystem instance, the provider of the offering, and the related stakeholders involved.

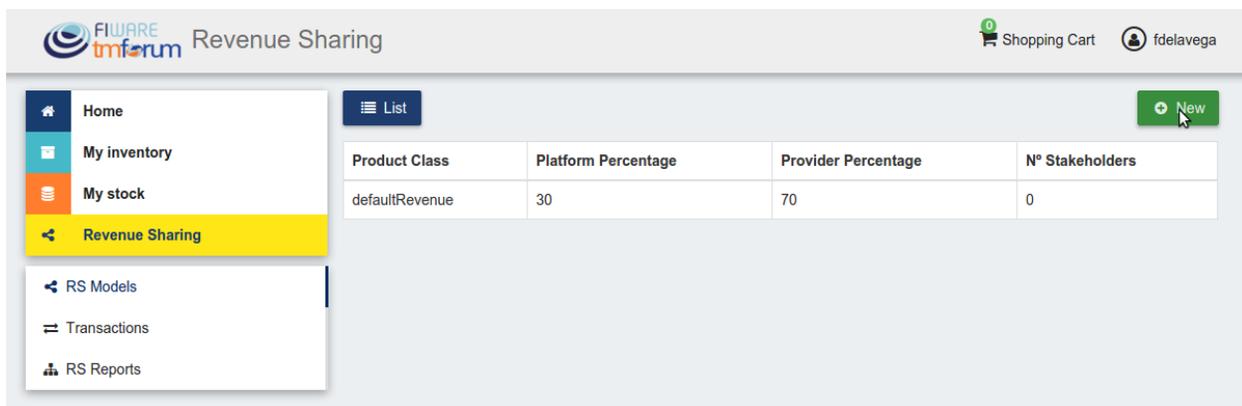
To manage RS models go to the *Revenue Sharing* section.



In this view, you can see the revenue sharing models you have available. By default it will appear the default RS model which establishes the revenue distribution between you and the Business API Ecosystem instance owner.



You can create a new RS model clicking on *New*

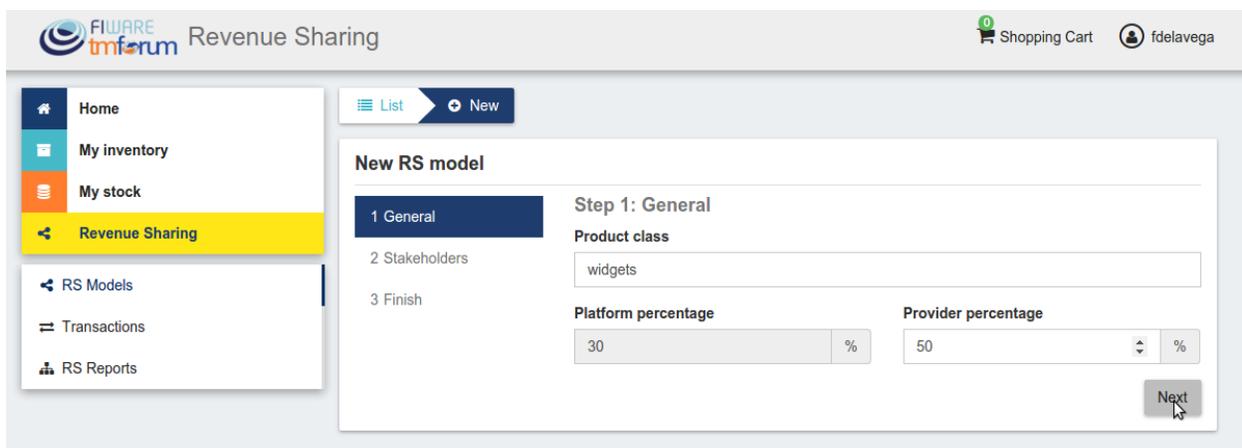


The screenshot shows the 'Revenue Sharing' application interface. On the left is a navigation menu with options: Home, My inventory, My stock, Revenue Sharing (highlighted), RS Models, Transactions, and RS Reports. The main content area displays a table with the following data:

Product Class	Platform Percentage	Provider Percentage	N° Stakeholders
defaultRevenue	30	70	0

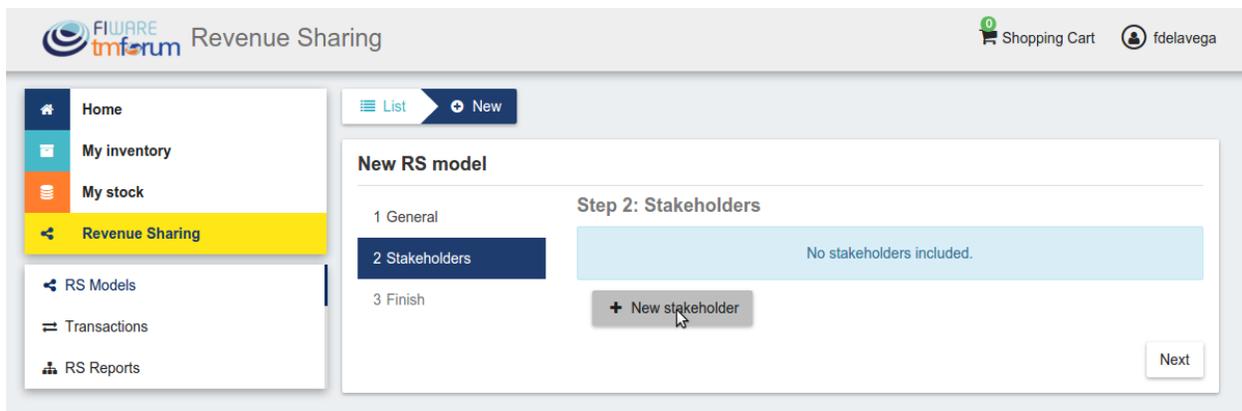
Buttons for 'List' and 'New' are visible at the top of the table area.

In the first step of the process you have to provide a product class, which identifies the RS model, and the percentage you want to receive. The platform percentage is fixed and cannot be modified. Once provided click on *Next*



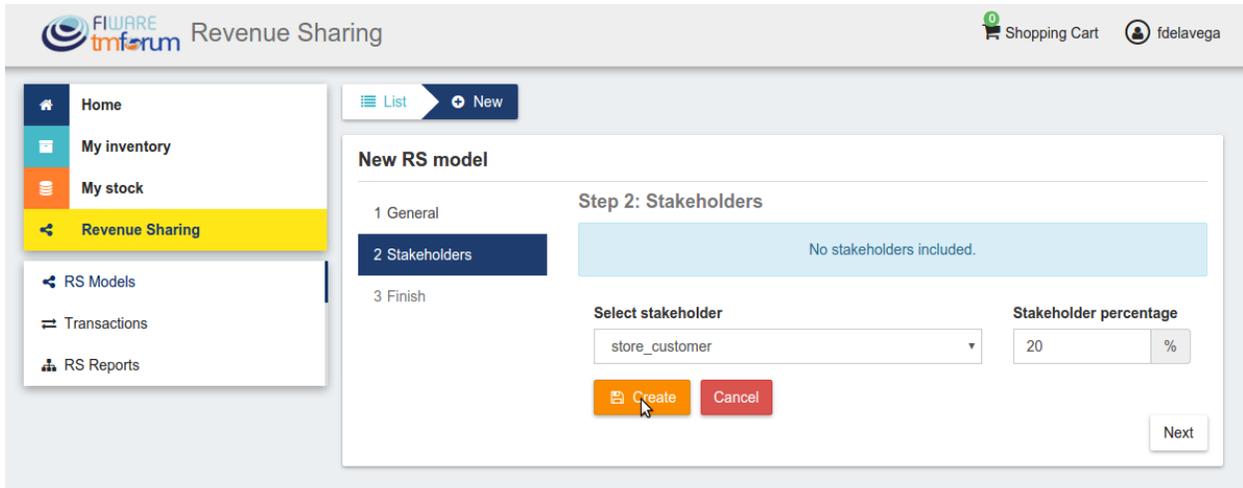
The screenshot shows the 'New RS model' form in the 'Step 1: General' stage. The left sidebar is the same as in the previous screenshot. The main form area has a progress indicator with three steps: 1 General (selected), 2 Stakeholders, and 3 Finish. The 'Product class' field contains 'widgets'. The 'Platform percentage' is set to 30% and is disabled. The 'Provider percentage' is set to 50% and is adjustable. A 'Next' button is located at the bottom right of the form.

In the next step, you can optionally add more stakeholders to the RS model. To do that click on *New Stakeholder*



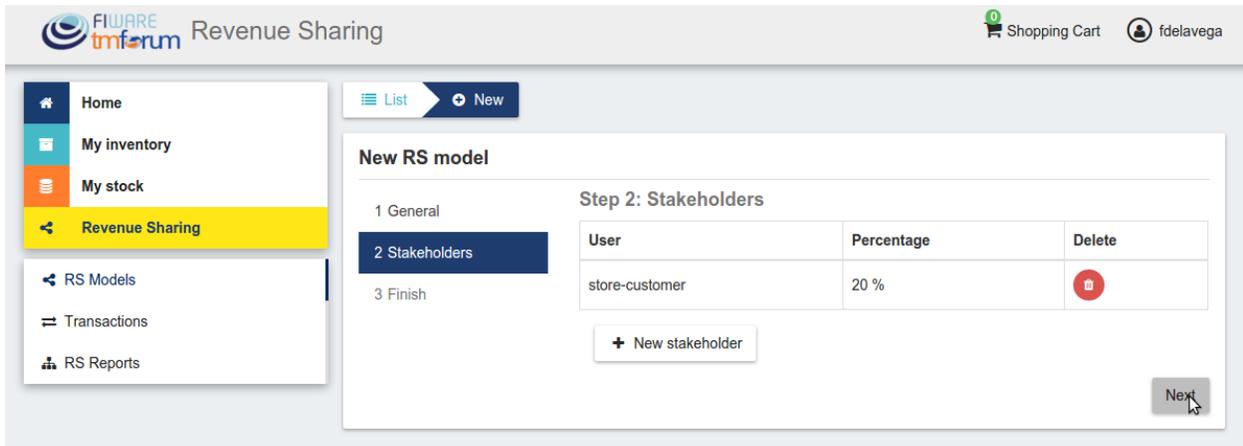
The screenshot shows the 'New RS model' form in the 'Step 2: Stakeholders' stage. The left sidebar is the same. The main form area has a progress indicator with three steps: 1 General, 2 Stakeholders (selected), and 3 Finish. A light blue box indicates 'No stakeholders included.' Below this box is a '+ New stakeholder' button. A 'Next' button is located at the bottom right of the form.

Then, select the Stakeholder between the available users, and provide its percentage. Finally, save it clicking on *Create*



Note: The total percentage (provider + platform + stakeholders) must be equal to 100

Finally, click on *Next* and then on *Create*



New RS model

1 General **Step 3: Finish**

2 Stakeholders General

3 Finish **Product class**

widgets

Platform percentage **Provider percentage**

30 % 50 %

Stakeholders

User	Percentage
store-customer	20 %

Total: 100 %

Create

Sellers can also update their RS model. To do that click on the RS model to be updated.

Revenue Sharing

Product Class	Platform Percentage	Provider Percentage	N° Stakeholders
defaultRevenue	30	70	0
widgets	30	50	1

Then, update the required fields (including the stakeholders if you want), and click on *Save Changes*

The screenshot shows the 'Revenue Sharing' configuration page for the 'widgets' product class. The configuration includes:

- Product class:** widgets
- Platform percentage:** 30%
- Provider percentage:** 50%

User	Percentage	Delete
store-customer	20 %	

Total: 100 %

[Save changes](#)

Manage Transactions

Sellers can manage the transactions related to their products in order to know how much money their products are generating, and to launch the revenue sharing process. To manage your seller transactions go to *Revenue Sharing* and click on *Transactions*

The screenshot shows the 'Transactions' page in the 'Revenue Sharing' section. The table displays the following data:

Product Class	Platform Percentage	Provider Percentage	N° Stakeholders
defaultRevenue	30	70	0
widgets	30	50	1

In the displayed view, you can see the transactions pending to be paid to you and your stakeholders. It is also possible to display the transactions in tabular way

FIWARE **tmforum** Revenue Sharing Shopping Cart fdelavega

defaultRevenue by fi-lab-user-example Tue, Sep 13th 2016, 13:22

Transaction Type Charge	Charged Amount 10 EUR
Product Offering 19 Basic Chart 0.1	Description One time payment: 10.00 EUR

defaultRevenue by fi-lab-user-example Tue, Sep 13th 2016, 14:02

Transaction Type Charge	Charged Amount 300 EUR
Product Offering 24 Nice Phone 0.1	Description One time payment: 300.00 EUR

FIWARE **tmforum** Revenue Sharing Shopping Cart fdelavega

Timestamp	Type	Product class	Customer	Product offering	Amount	Description
Tue, Sep 13th 2016, 13:22	Charge	defaultRevenue	fi-lab-user-example	19 Basic Chart 0.1	10 EUR	One time payment: 10.00 EUR
Tue, Sep 13th 2016, 14:02	Charge	defaultRevenue	fi-lab-user-example	24 Nice Phone 0.1	300 EUR	One time payment: 300.00 EUR

These transactions are aggregated and paid by the Business API Ecosystem periodically once a month. Nevertheless, if you need to be paid, you can force the revenue sharing calculus and payment of your pending transactions by manually generating a revenue sharing report.

To create a new report click on *New Report*

FIWARE **tmforum** Revenue Sharing Shopping Cart fdelavega

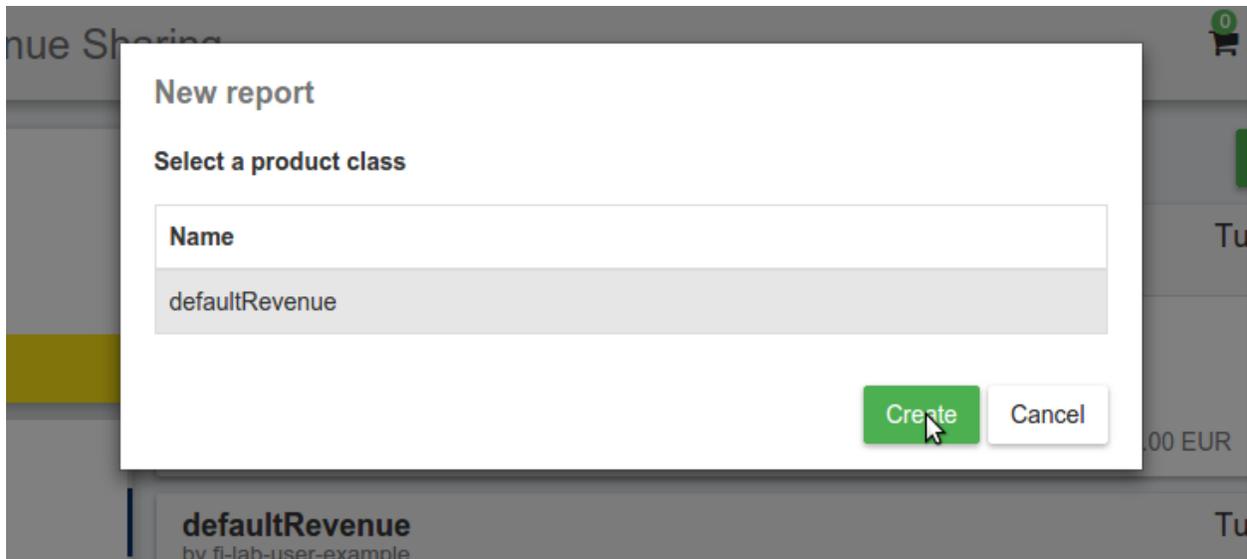
defaultRevenue by fi-lab-user-example Tue, Sep 13th 2016, 13:22

Transaction Type Charge	Charged Amount 10 EUR
Product Offering 19 Basic Chart 0.1	Description One time payment: 10.00 EUR

defaultRevenue by fi-lab-user-example Tue, Sep 13th 2016, 14:02

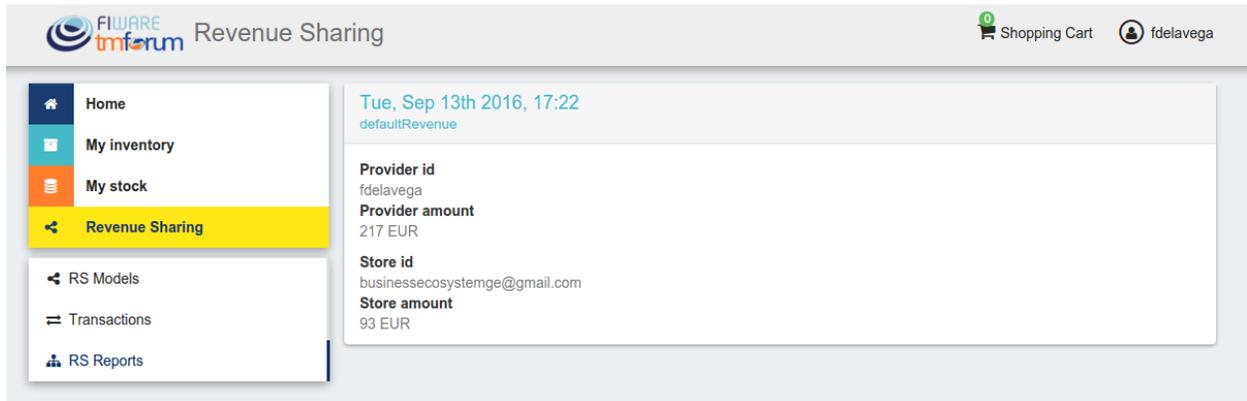
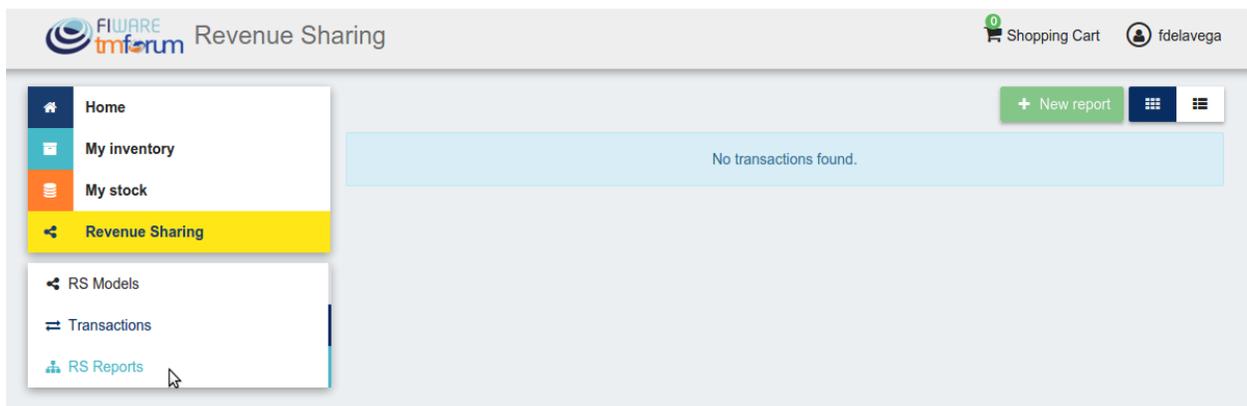
Transaction Type Charge	Charged Amount 300 EUR
Product Offering 24 Nice Phone 0.1	Description One time payment: 300.00 EUR

In the displayed modal, choose the product classes to be calculated and click on *Create*



This process will aggregate all the transactions with the selected product classes, calculate the amount to be paid to each stakeholder using the related revenue sharing model, generate a revenue sharing report, and pay the seller and the stakeholders using their PayPal account.

You can see the generated reports clicking on *RS Reports*



Note: Sellers would need to have a PayPal account associated to the email of their FIWARE IdM account in order to be paid for their products

Manage Received Orders

Sellers can manage the orders they have received in order to see the chosen characteristics, read customer notes, or process the order in case it has been acquired a physical product.

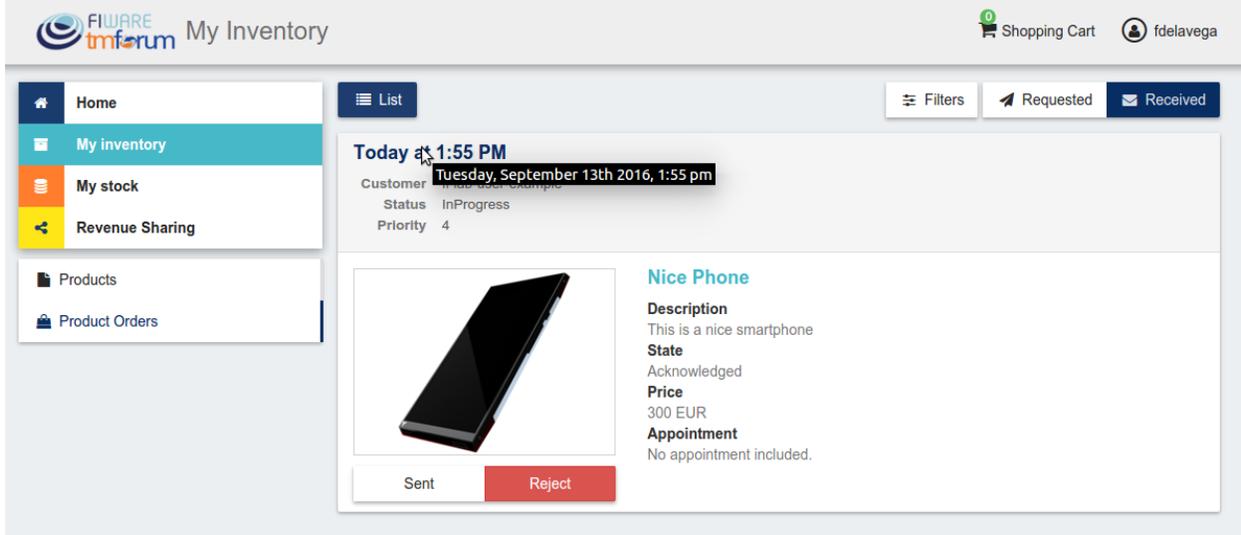
To view your received orders go to *My inventory* section, click on *Product orders*, and open the *Received* section.

The first screenshot shows the FIWARE Marketplace homepage. The left sidebar contains a navigation menu with 'Home', 'My inventory', 'My stock', and 'Revenue Sharing'. Below this is a 'Search by catalog' section with options for 'All', 'New Catalog', 'Widgets Catalog', and 'Services Catalog'. The main content area displays three product cards: 'Wirecloud Basic Chart' (v0.1, 3 hours ago, From 10 EUR), 'Nice Phone' (v0.1, 2 hours ago, From 300 EUR), and 'Usage' (v0.1, an hour ago, From 1 EUR / CALL). Each card has an 'Add to cart' button.

The second screenshot shows the 'My Inventory' section. The left sidebar now includes 'Products' and 'Product Orders'. The main content area displays a message: 'No products found.'

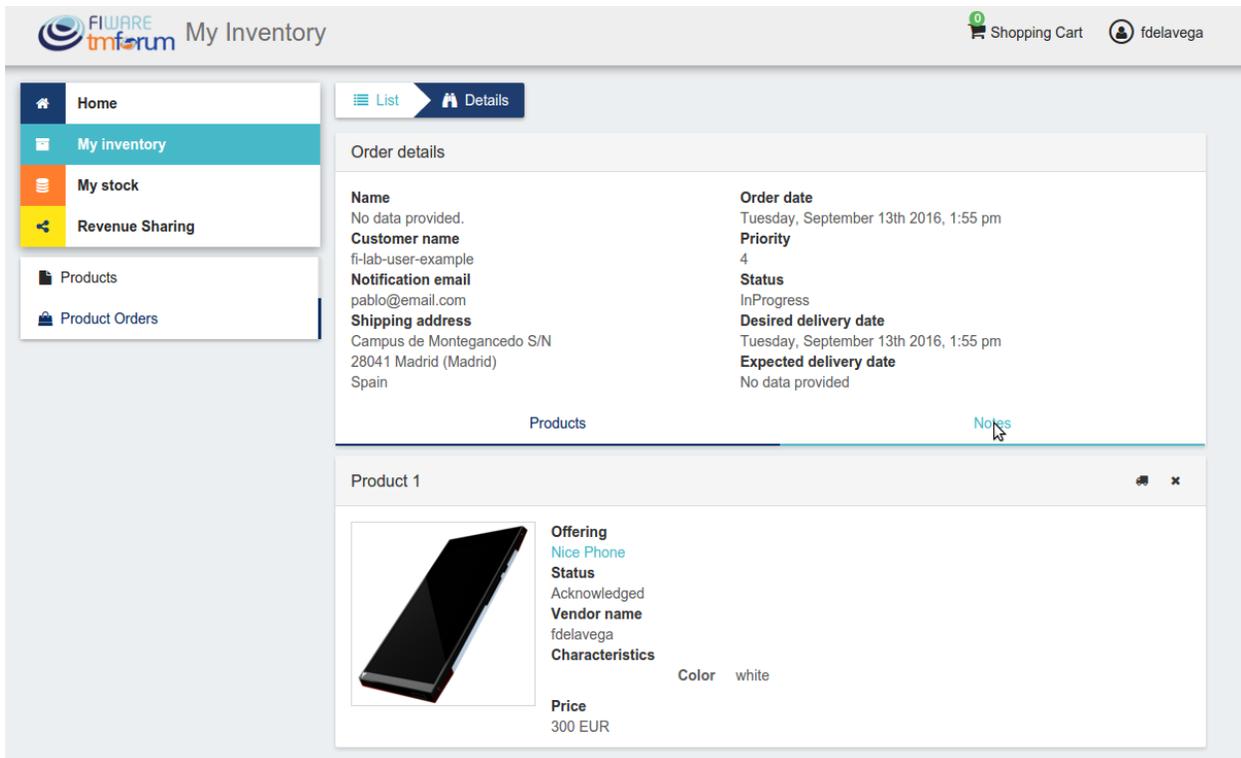
The third screenshot shows the 'My Inventory' section with the 'Product Orders' sub-section selected. The main content area displays a message: 'No product orders found.' The top right of this section includes tabs for 'Requested' and 'Received', with 'Received' being the active tab.

You can view the details of a received order clicking on the order date



In the displayed view you can review the details of the order and the details of your products acquired by the customer, including the chosen characteristics.

Additionally, you can view the customer notes clicking on the *Notes* tab



You can also give a reply to customer notes including it in the text area and clicking on the send button

Order details

Name No data provided.	Order date Tuesday, September 13th 2016, 1:55 pm
Customer name fi-lab-user-example	Priority 4
Notification email pablo@email.com	Status InProgress
Shipping address Campus de Montegancedo S/N 28041 Madrid (Madrid) Spain	Desired delivery date Tuesday, September 13th 2016, 1:55 pm
	Expected delivery date No data provided

Notes

Enter a note

There are't any remaining silver phone

fi-lab-user-example
Today at 2:16 PM I prefer the silver phone instead

If the acquired product is not digital, the order needs to be processed manually by the seller, in the sense that the seller will have to send the acquired product to the customer. To deal with this situation, the order details view allows sellers to manually change the status of the order.

To reject a received order you have to click in the *Reject* button located in the search or in the details view of the order.

Today at 1:55 PM

Customer: fi-lab-user-example
Status: InProgress
Priority: 4

Nice Phone

Description
This is a nice smartphone

State
Acknowledged

Price
300 EUR

Appointment
No appointment included.

Sent **Reject**

Order details

Name No data provided.	Order date Tuesday, September 13th 2016, 1:55 pm
Customer name fi-lab-user-example	Priority 4
Notification email pablo@email.com	Status InProgress
Shipping address Campus de Montegancedo S/N 28041 Madrid (Madrid) Spain	Desired delivery date Tuesday, September 13th 2016, 1:55 pm
	Expected delivery date No data provided

Product 1

	<p>Offering Nice Phone</p> <p>Status Acknowledged</p> <p>Vendor name fdelavega</p> <p>Characteristics Color white</p> <p>Price 300 EUR</p>
---	---

In case you accept the order and send the product to the customer, you have to put it as *inProgress* clicking on the *Sent* button

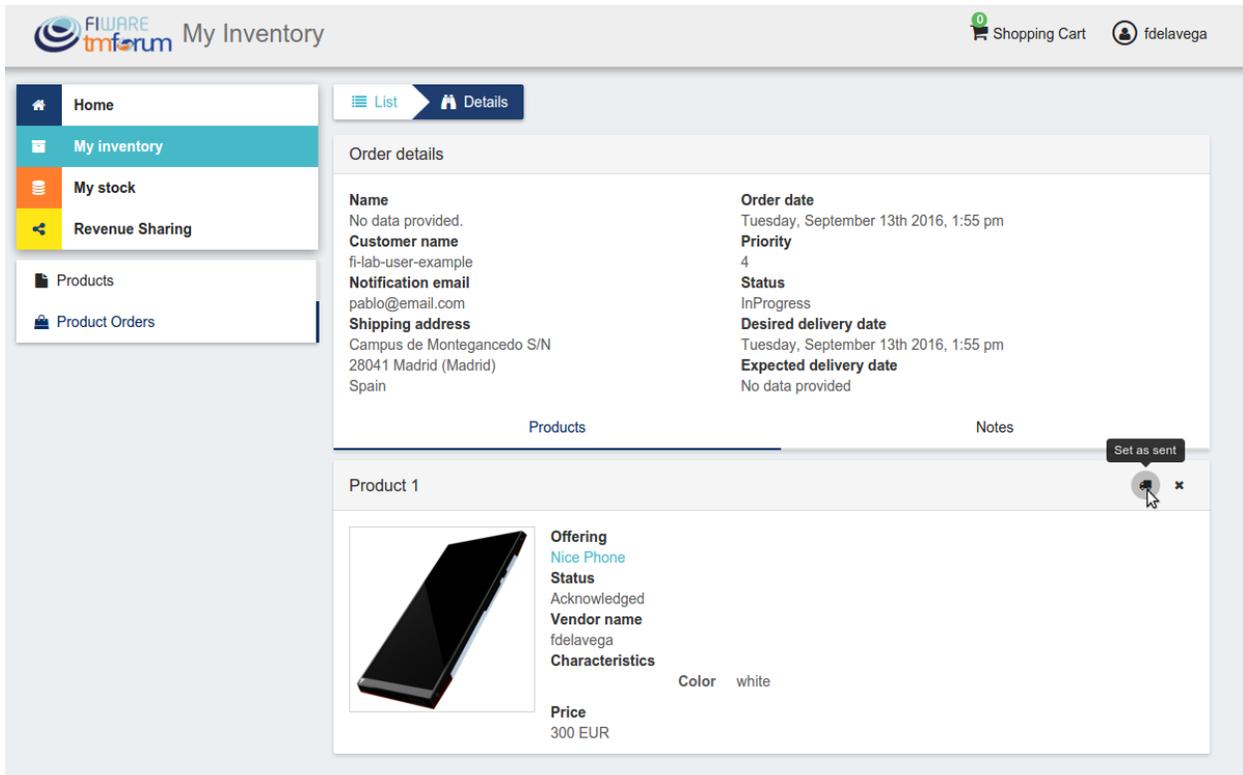
Today at 1:55 PM

Customer	fi-lab-user-example
Status	InProgress
Priority	4

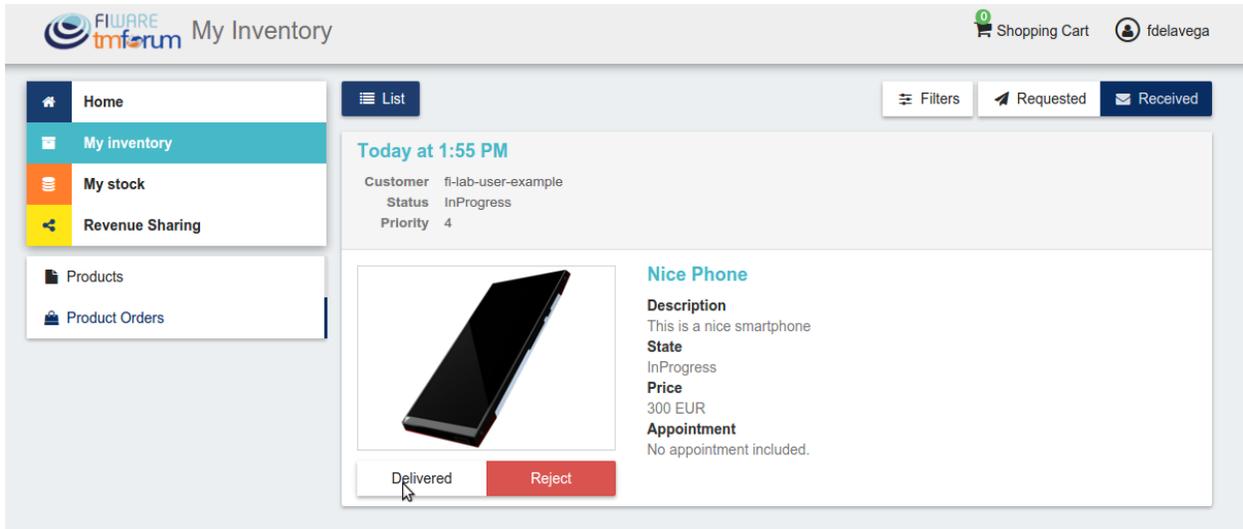
Nice Phone

	<p>Description This is a nice smartphone</p> <p>State Acknowledged</p> <p>Price 300 EUR</p> <p>Appointment No appointment included.</p>
---	---

Sent **Reject**



Finally, when the product arrives at its destination, you have to put it as *Completed* clicking on the *Delivered* button



My Inventory

- Home
- My inventory
- My stock
- Revenue Sharing
- Products
- Product Orders

Order details

Name No data provided.	Order date Tuesday, September 13th 2016, 1:55 pm
Customer name fi-lab-user-example	Priority 4
Notification email pablo@email.com	Status InProgress
Shipping address Campus de Montegancedo S/N 28041 Madrid (Madrid) Spain	Desired delivery date Tuesday, September 13th 2016, 1:55 pm
	Expected delivery date No data provided

Product 1

	<p>Offering Nice Phone</p> <p>Status InProgress</p> <p>Vendor name fdelavega</p> <p>Characteristics Color white</p> <p>Price 300 EUR</p>
---	---

Buttons: Products, Notes, Set as delivered

Customer

All of the users of the system have by default the *Customer* role. Customers are able to create orders for acquiring offerings.

List Available Offerings

All the available (*Launched*) offerings appear in the *Home* page of the Business API Ecosystem, so they can be seen by customers.

Marketplace

- Home
- My inventory
- My stock
- Revenue Sharing

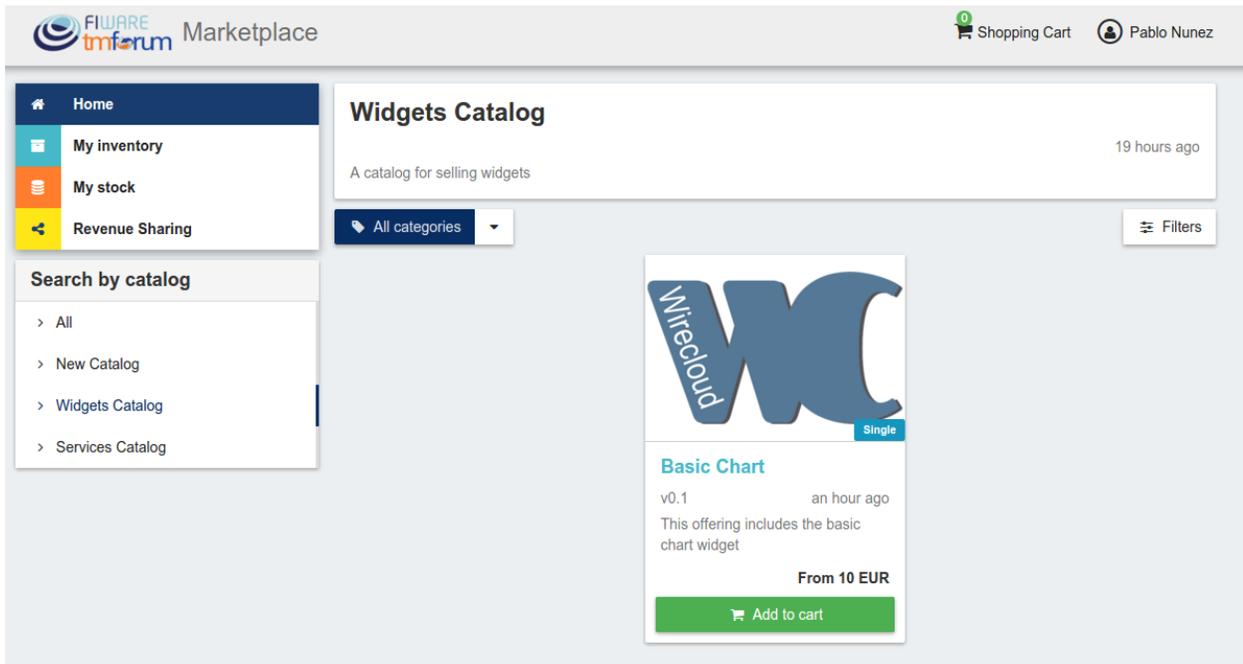
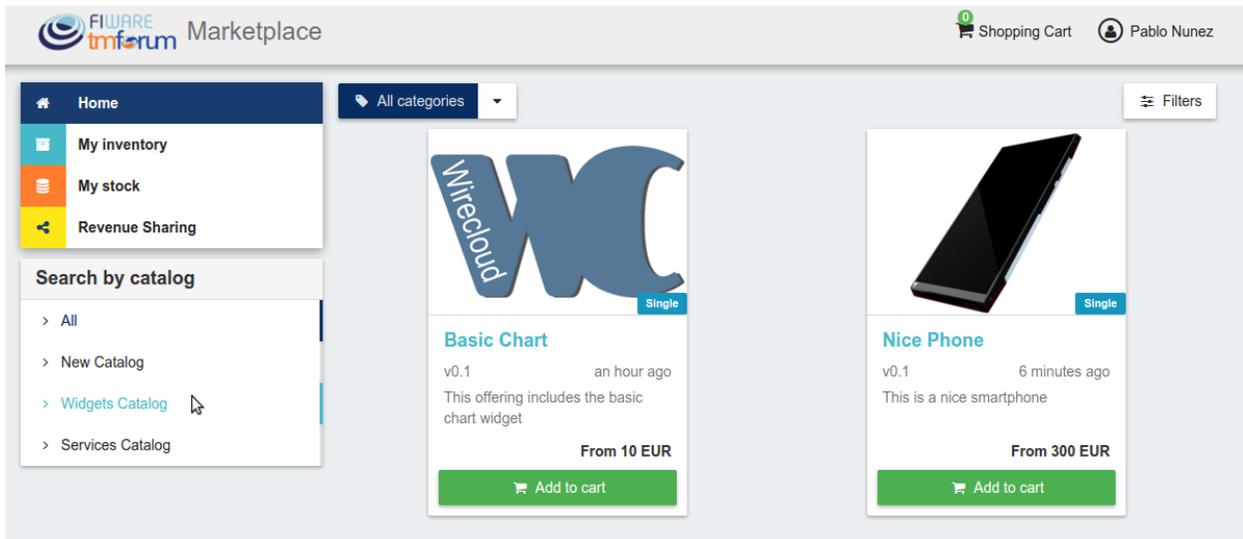
Search by catalog

- All
- New Catalog
- Widgets Catalog
- Services Catalog

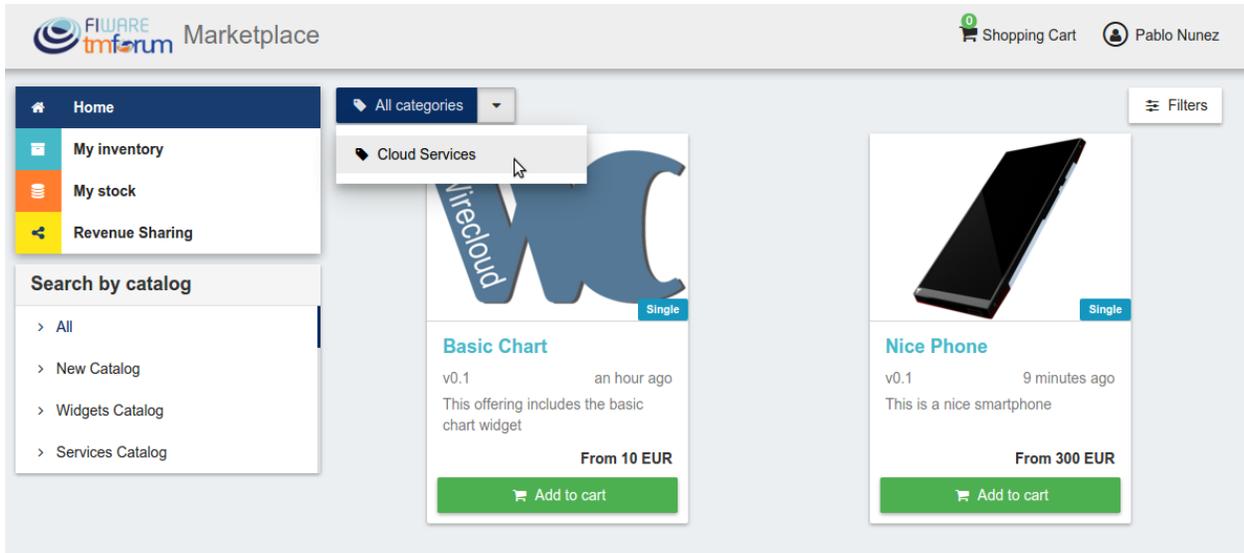
All categories | Filters

<p></p> <p>Basic Chart</p> <p>v0.1 an hour ago</p> <p>This offering includes the basic chart widget</p> <p>From 10 EUR</p> <p>Add to cart</p>	<p></p> <p>Nice Phone</p> <p>v0.1 5 minutes ago</p> <p>This is a nice smartphone</p> <p>From 300 EUR</p> <p>Add to cart</p>
--	--

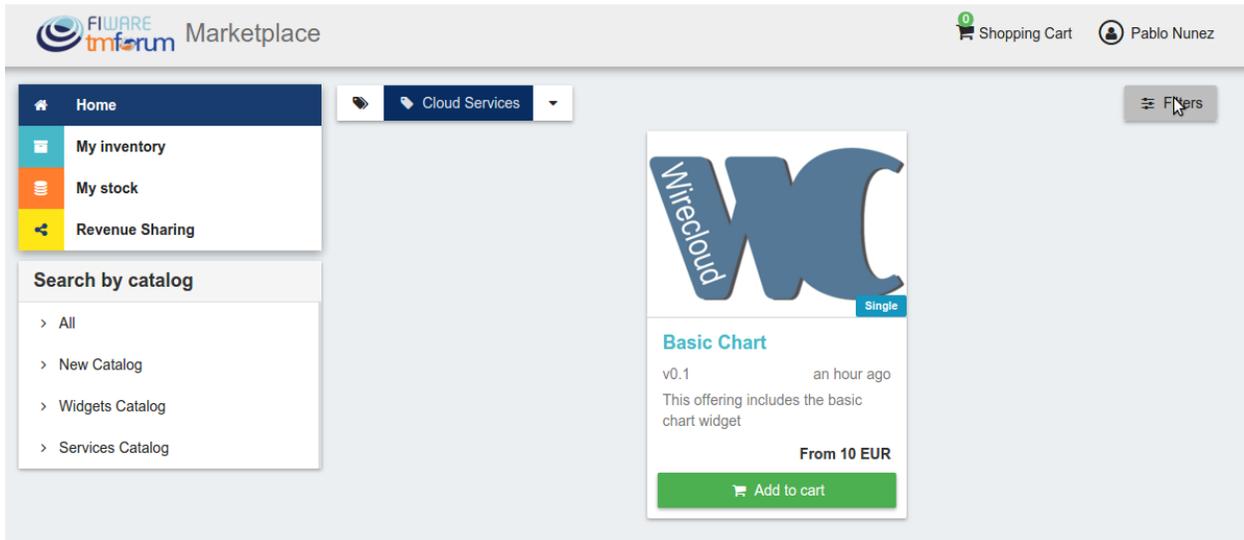
Additionally, customers can select a specific catalog of offerings by clicking on it.

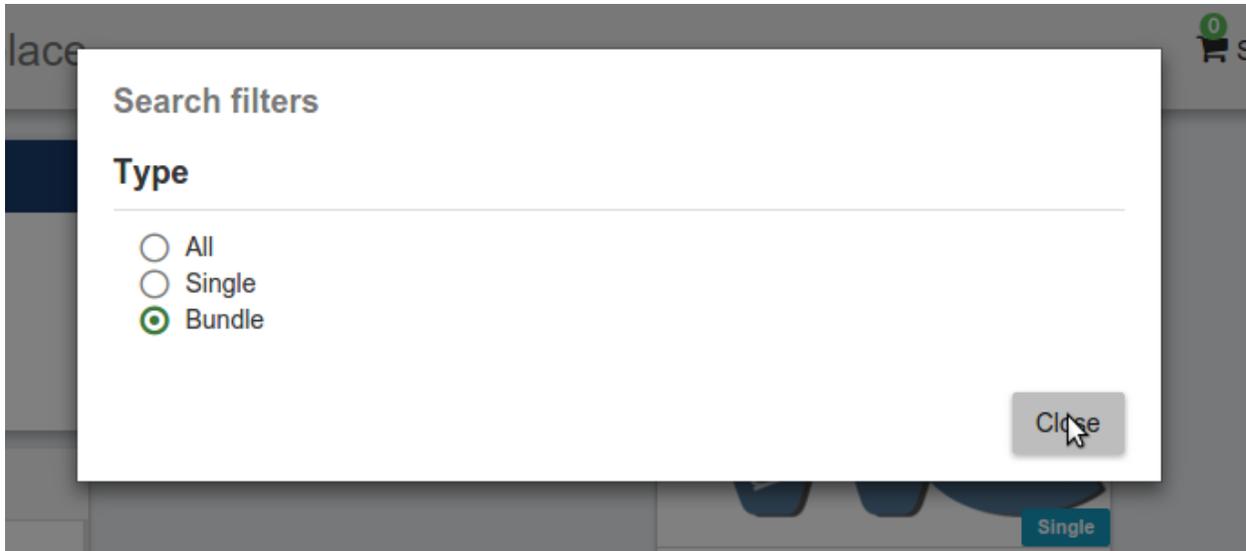


Moreover, customers can filter the shown offerings by category using the categories dropdown and choosing the wanted one.

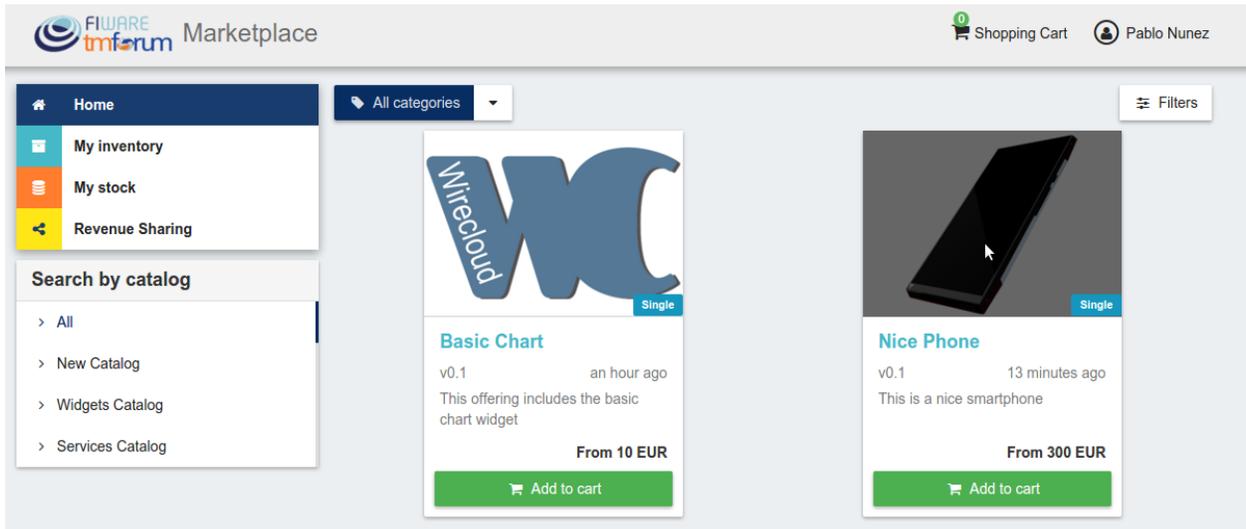


Finally, customers can also filter bundle or single offerings using the *Filters* modal.





Customers can open the details of an offering by clicking on it



In the displayed view, it is shown the general info about the offering and its included product, the characteristics of the product, the price plans of the offering, and the existing relationships.

FIWARE **tmforum** Marketplace Shopping Cart Pablo Nunez

[Back](#) [Details](#)



Nice Phone

From 300 EUR

[Add to cart](#)

[About](#) [Characteristics](#) [Price plans](#) [Relationships](#)

Offering information

Name
Nice Phone

Version
0.1

Description
This is a nice smartphone

Updated
Tuesday, September 13th 2016, 12:15 pm

Product information

Name
Nice Phone

Version
0.1

Description
This is a nice smartphone

Updated
Tuesday, September 13th 2016, 12:13 pm

Brand
UPM

ID Number

FIWARE **tmforum** Marketplace Shopping Cart Pablo Nunez

[Back](#) [Details](#)



Nice Phone

From 300 EUR

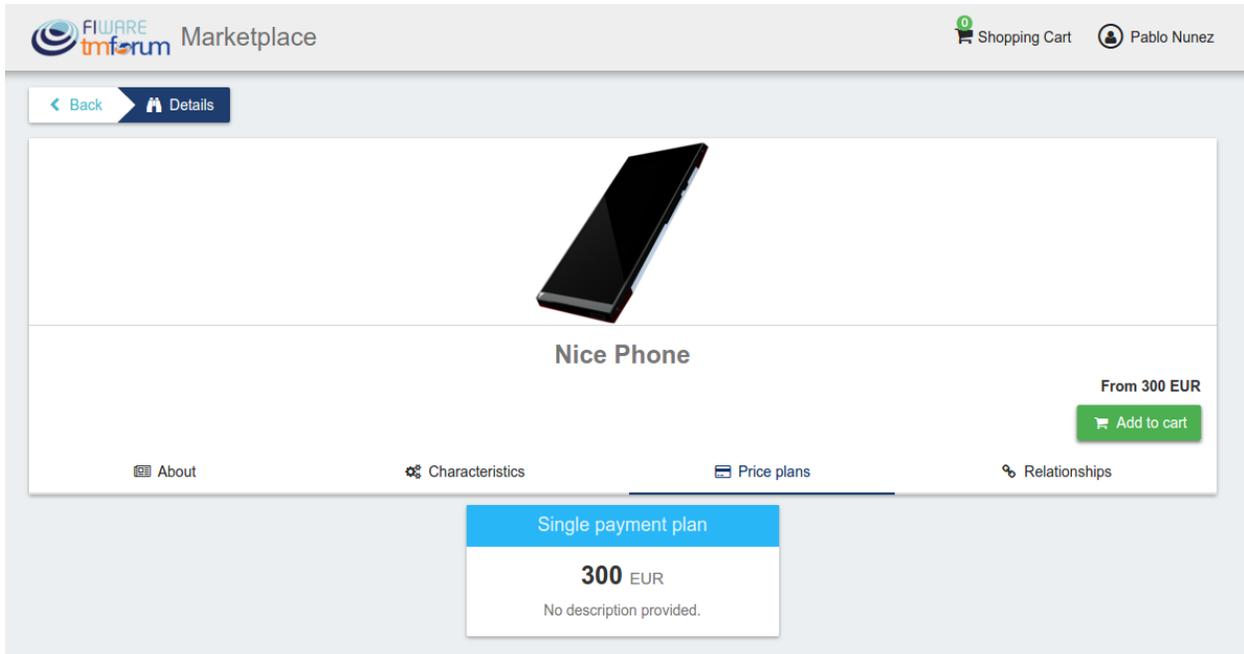
[Add to cart](#)

[About](#) [Characteristics](#) [Price plans](#) [Relationships](#)

Color

Color of the phone

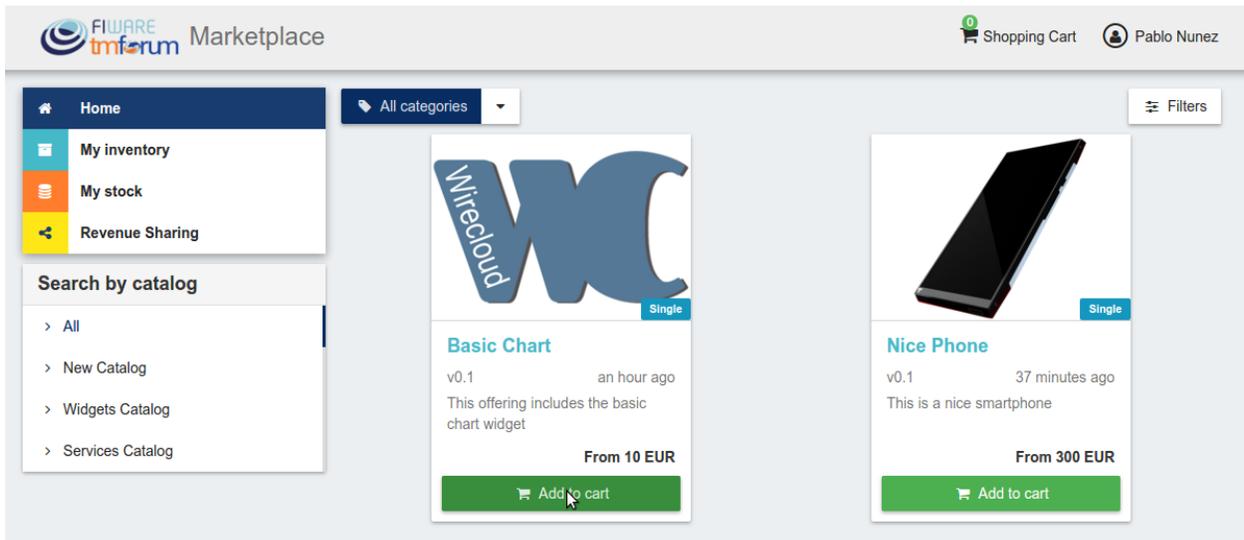
- white
- black
- silver

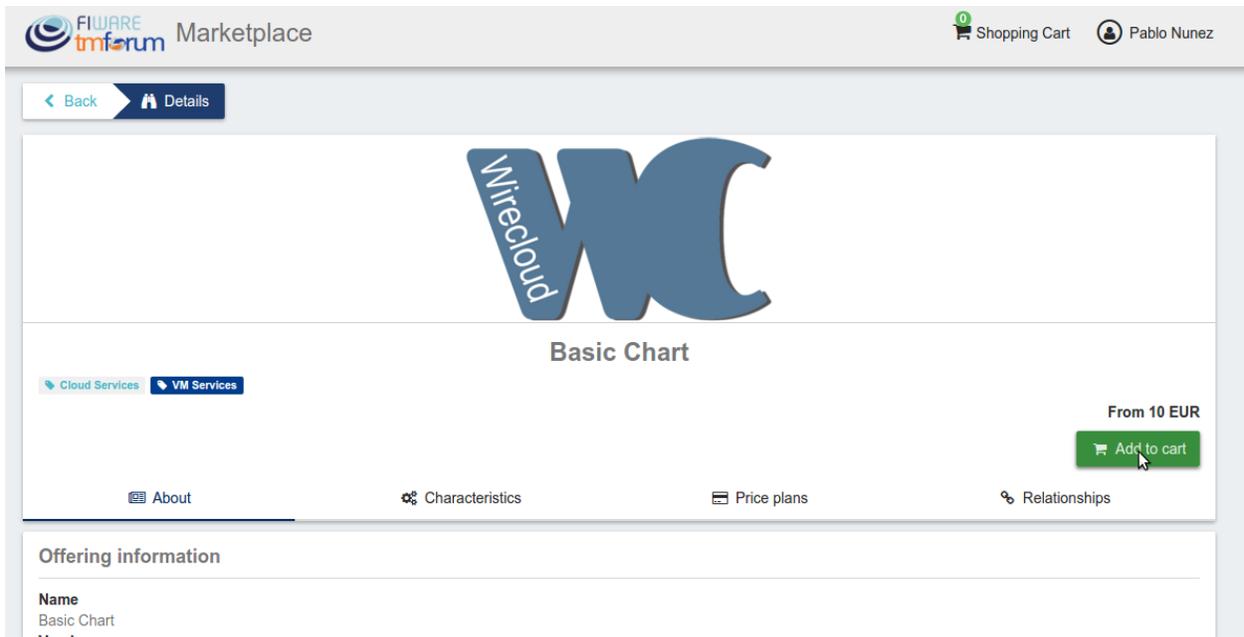


Create Order

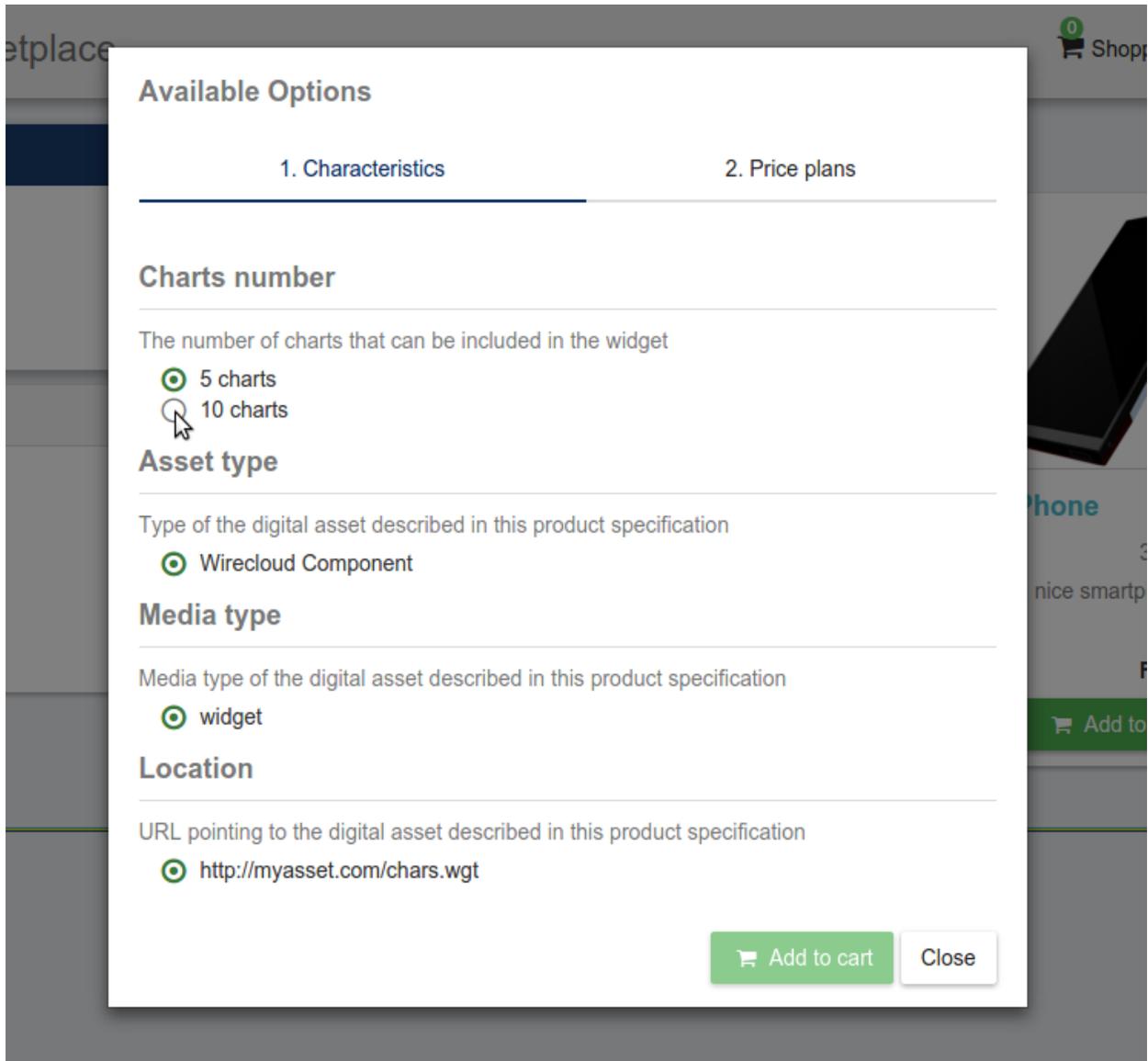
Customers can create orders for acquiring offerings. The different offerings to be included in an order are managed using the *Shopping Cart*.

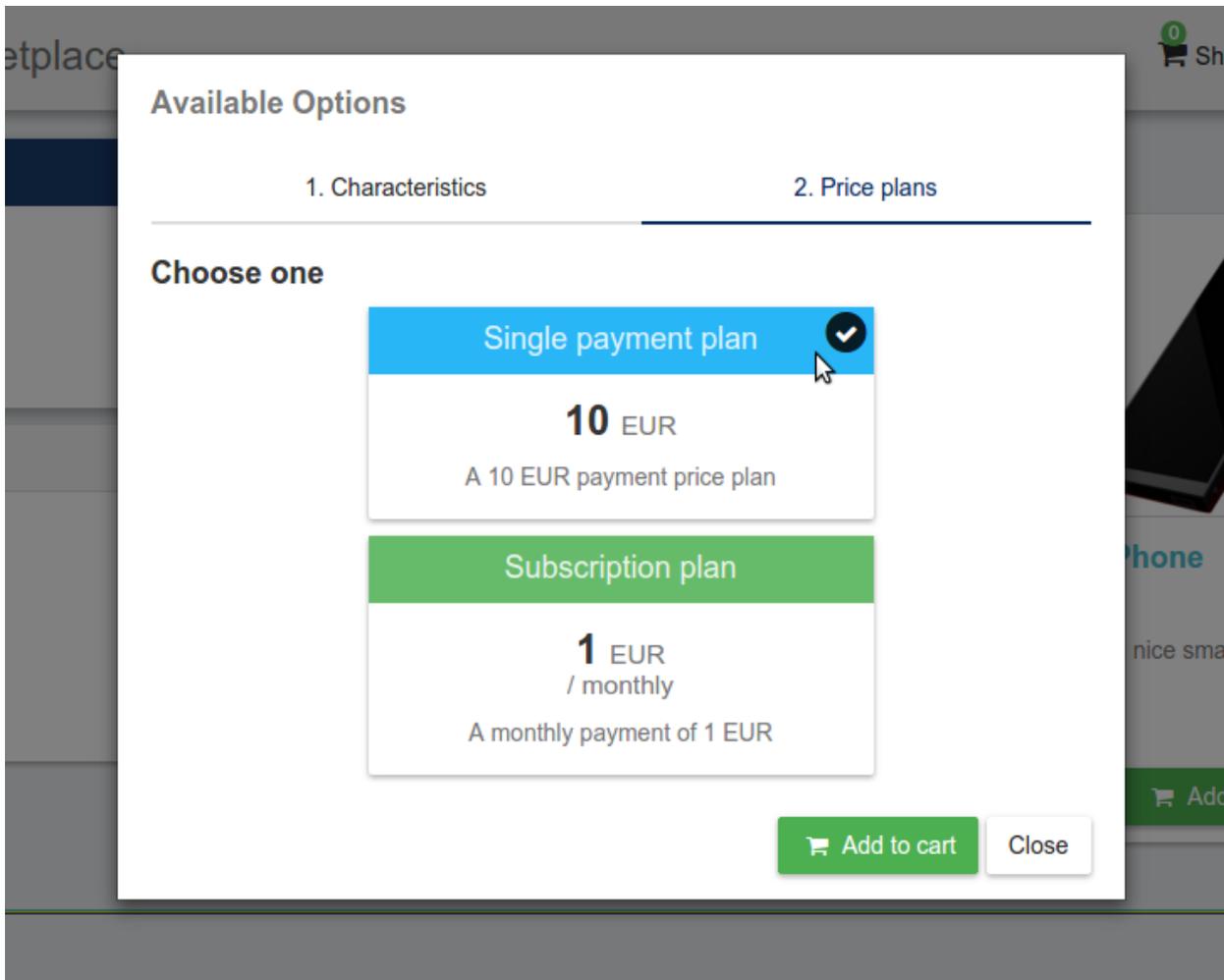
To include an offering in the shopping cart there are two possibilities. You can click on the *Add to Cart* button located in the offering panel when searching, or you can click on the *Add to Cart* button located in the offering details view.



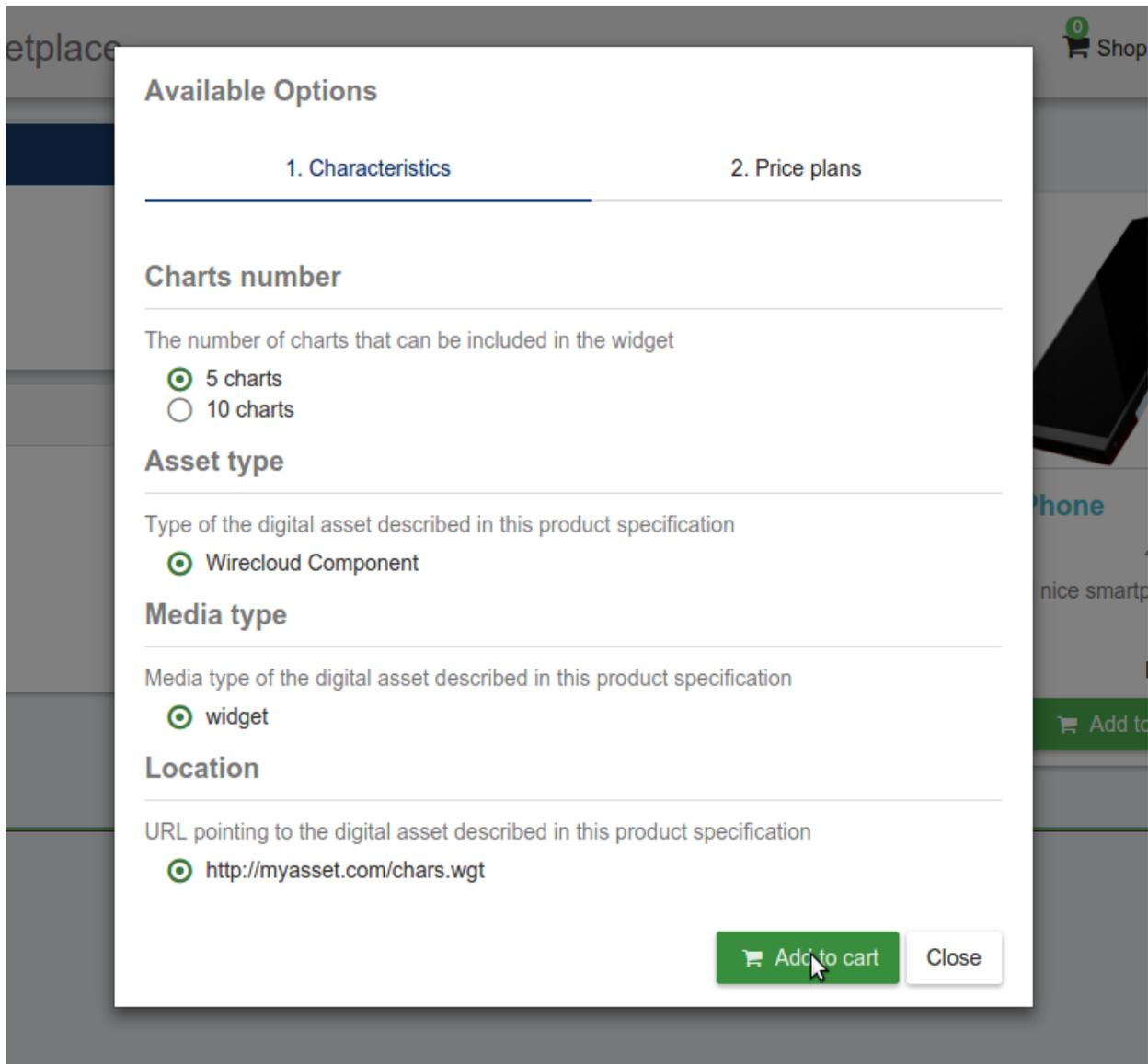


If the offering has configurable characteristics or multiple price plans, a modal will be displayed where you can select your preferred options

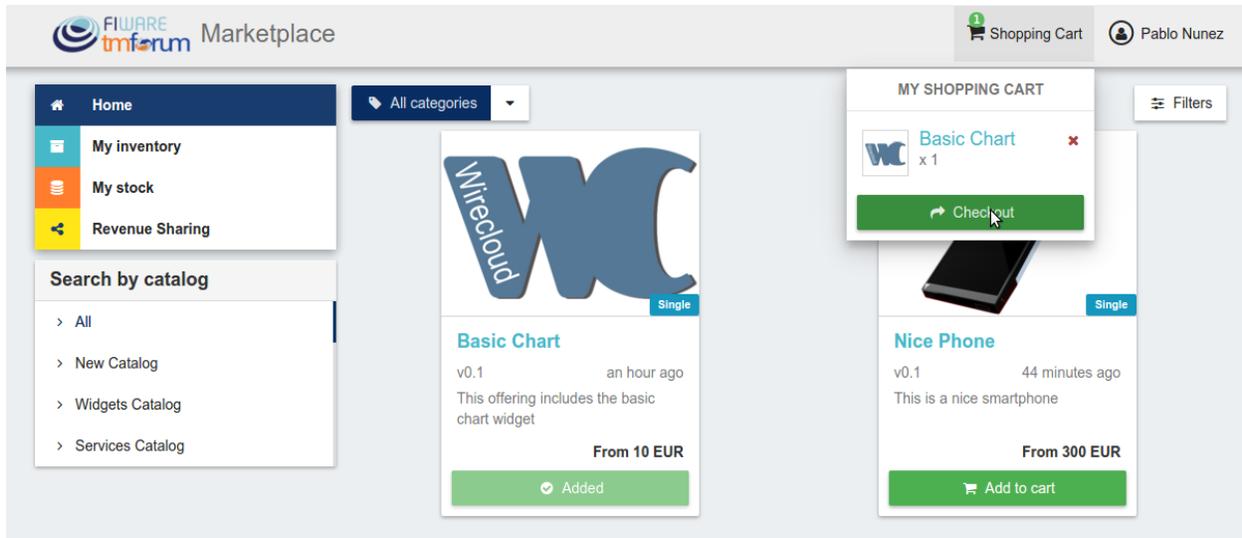




Once you have selected your preferences for the offering click on *Add to Cart*



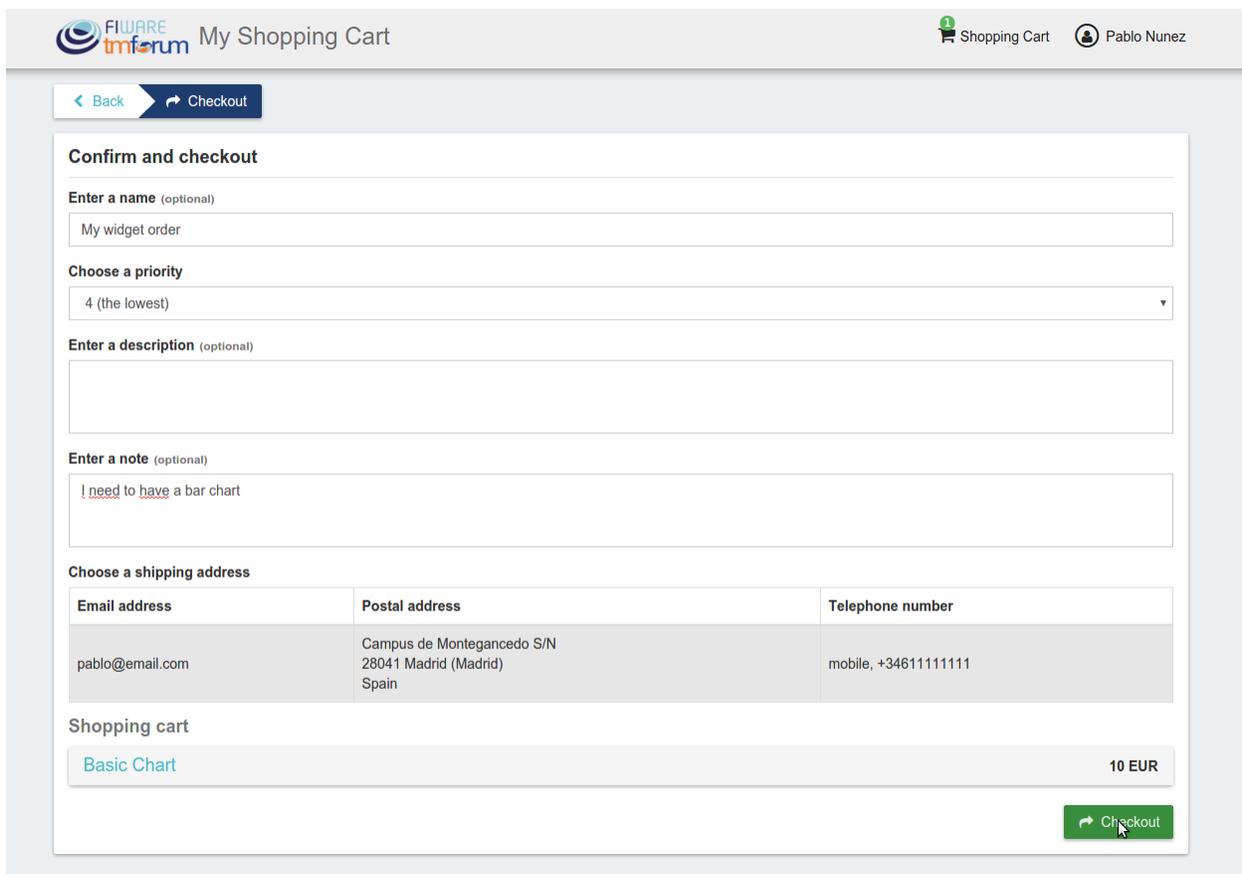
Once you have included all the offerings you want to acquire to the shopping cart, you can create the order clicking on *Shopping Cart*, and then on *Checkout*



In the displayed form, you can include an optional name, an optional description, or an optional note. Notes can include any additional information you want to provide to the sellers of the acquired offerings.

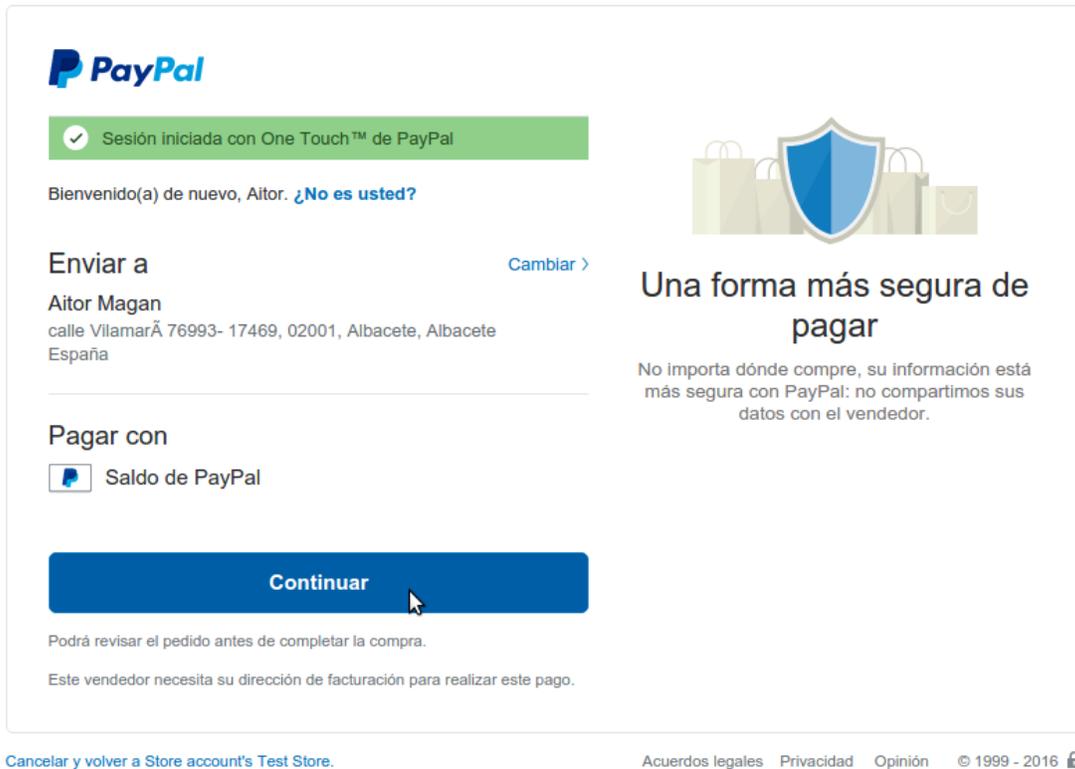
Then, you have to choose a priority for your order, and select one of your shipping addresses.

Once you have provided all the required information you can start the order creation clicking on *Checkout*



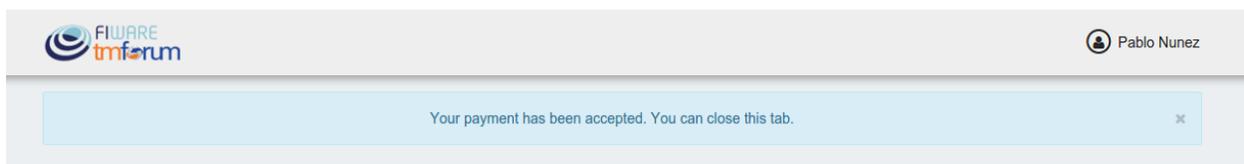
In the next step, you will be redirected to *PayPal* so you can pay for the offerings according to their pricing models

Store account's Test Store



The screenshot shows the PayPal checkout interface. At the top left is the PayPal logo. Below it, a green banner indicates 'Sesión iniciada con One Touch™ de PayPal'. A welcome message reads 'Bienvenido(a) de nuevo, Aitor. ¿No es usted?'. The recipient information is 'Enviar a Aitor Magan, calle VilamarÃ 76993- 17469, 02001, Albacete, Albacete España', with a 'Cambiar >' link. The payment method is 'Pagar con Saldo de PayPal'. A large blue 'Continuar' button is prominent. Below the button, it states 'Podrá revisar el pedido antes de completar la compra.' and 'Este vendedor necesita su dirección de facturación para realizar este pago.' On the right, there is a shield icon over shopping bags and the text 'Una forma más segura de pagar' followed by 'No importa dónde compre, su información está más segura con PayPal: no compartimos sus datos con el vendedor.' At the bottom, there are links for 'Cancelar y volver a Store account's Test Store.', 'Acuerdos legales', 'Privacidad', 'Opinión', and '© 1999 - 2016' with a lock icon.

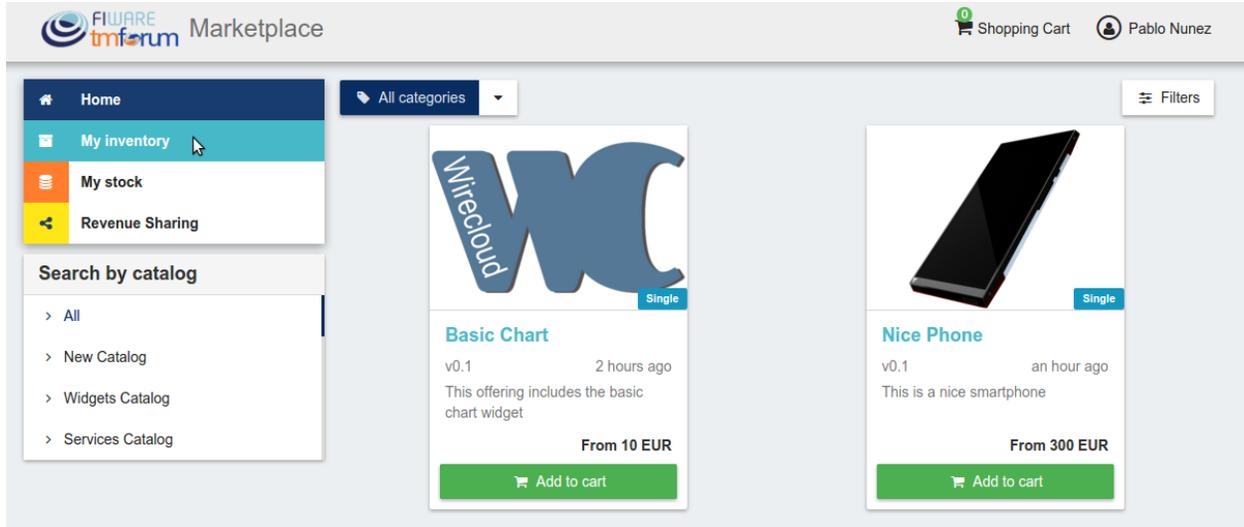
Finally, you will see a confirmation page



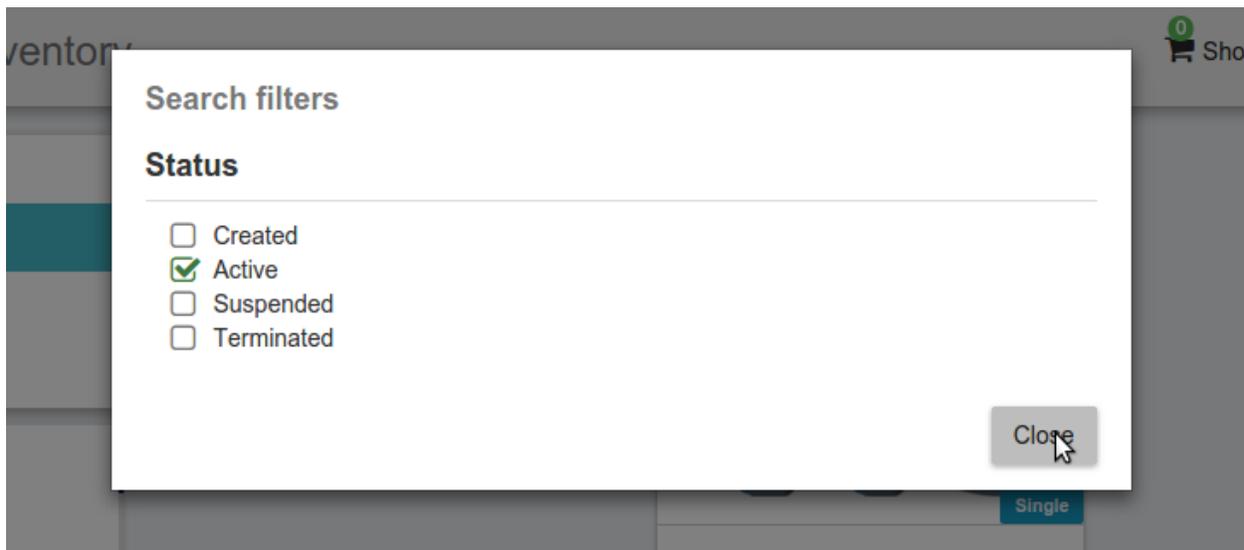
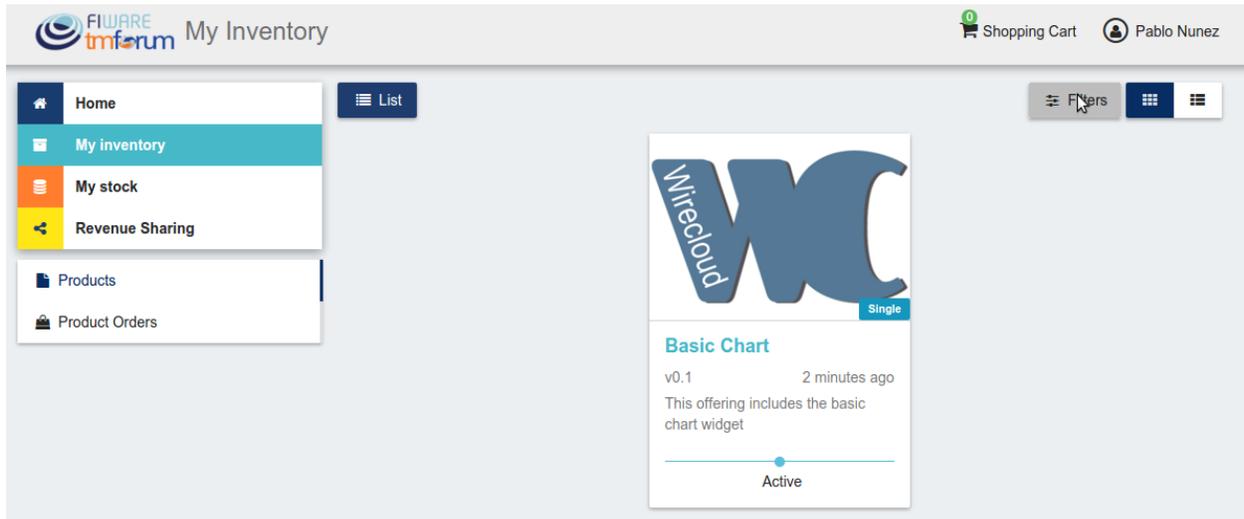
The screenshot shows a confirmation page with the 'FIJWARE tmforum' logo in the top left and a user profile 'Pablo Nunez' in the top right. A light blue notification box in the center contains the text 'Your payment has been accepted. You can close this tab.' with a close button (x) on the right.

Manage Acquired Products

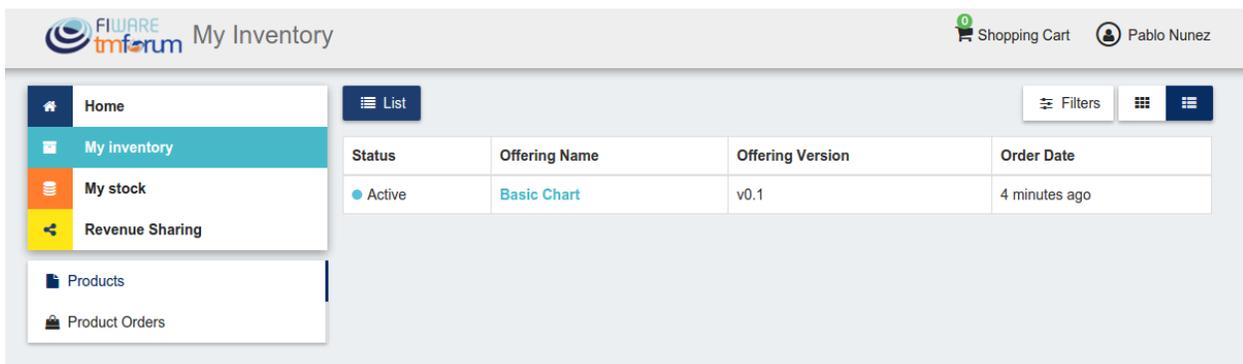
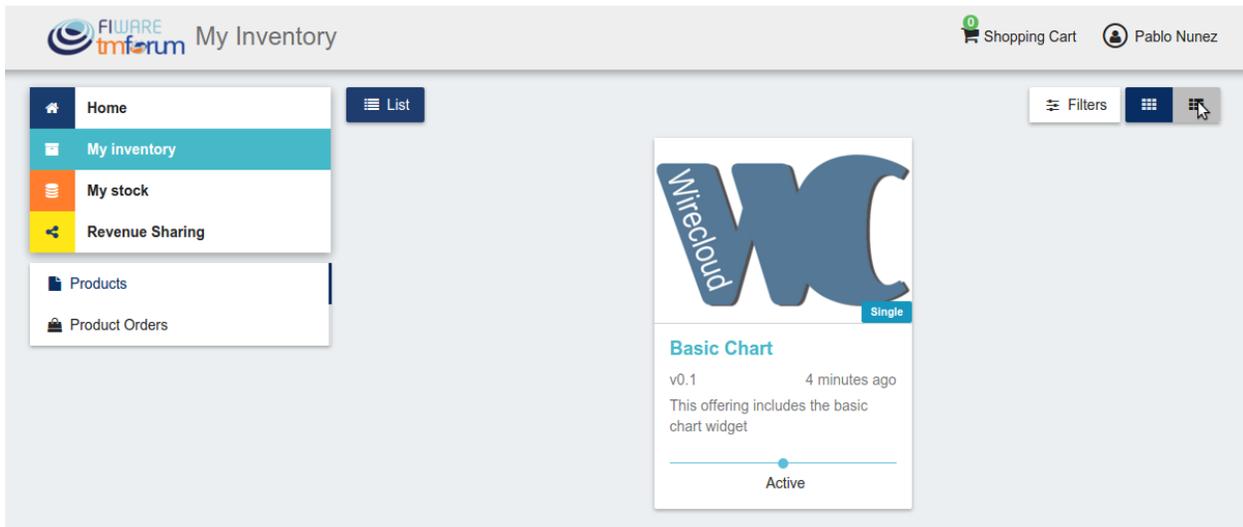
The products you have acquired are located in *My Inventory*, there you can list them, check their status, or download different assets.



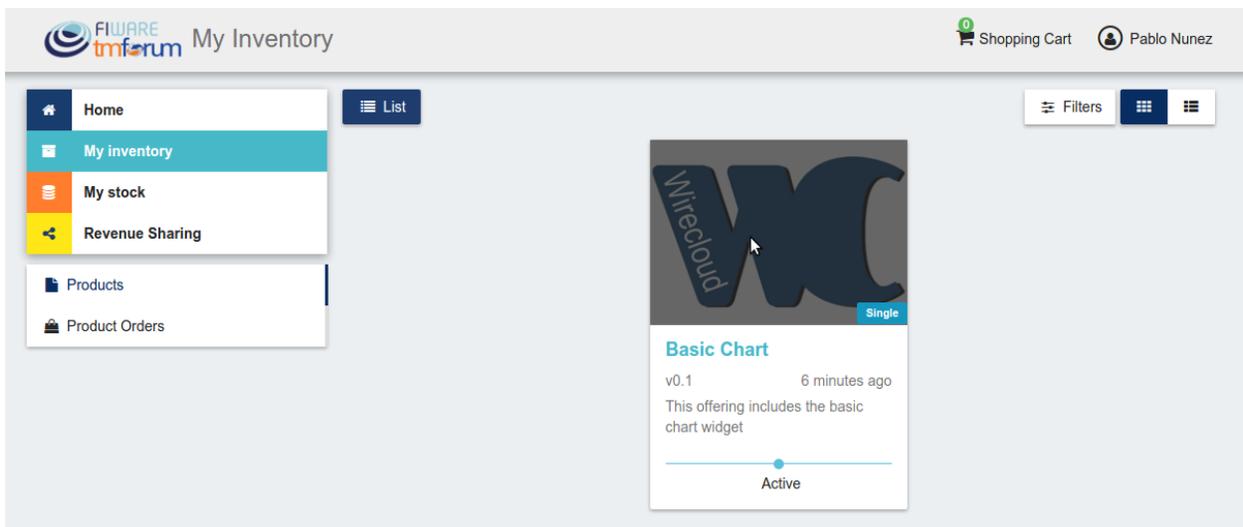
In this view, it is possible to filter you products by its status. To do that click on *Filters*, select the related statuses, and click on *Close*



It is also possible to switch between the grid and tabular views using the related buttons



You can manage a specific acquired product clicking on it



In the displayed view, you can see the general info of the acquired product, and the characteristics and pricing you have selected.

FIWARE inforum My Inventory Shopping Cart Pablo Nunez

Home My inventory My stock Revenue Sharing Products Product Orders

List Details

Wirecloud WC

Basic Chart Download

About Characteristics Price plans Charges

General

Description
No description provided.

Offering
Basic Chart

Start date
9 minutes ago

FIWARE inforum My Inventory Shopping Cart Pablo Nunez

Home My inventory My stock Revenue Sharing Products Product Orders

List Details

Wirecloud WC

Basic Chart Download

About Characteristics Price plans Charges

Charts number

The number of charts that can be included in the widget

- 5 charts
- 10 charts

Asset type

Type of the digital asset described in this product specification

- Wirecloud Component

Media type

Media type of the digital asset described in this product specification

- widget

The screenshot shows the 'My Inventory' interface for 'Wirecloud WC'. The main content area displays the 'Basic Chart' with two price plans:

Plan Type	Price	Description
Single payment plan	10 EUR	A 10 EUR payment price plan
Subscription plan	1 EUR / month	A monthly payment of 1 EUR

The 'Charges' tab is highlighted in the bottom navigation bar, indicating it is the active view.

Additionally, you can see your charges related to the product accessing to the *Charges* tab

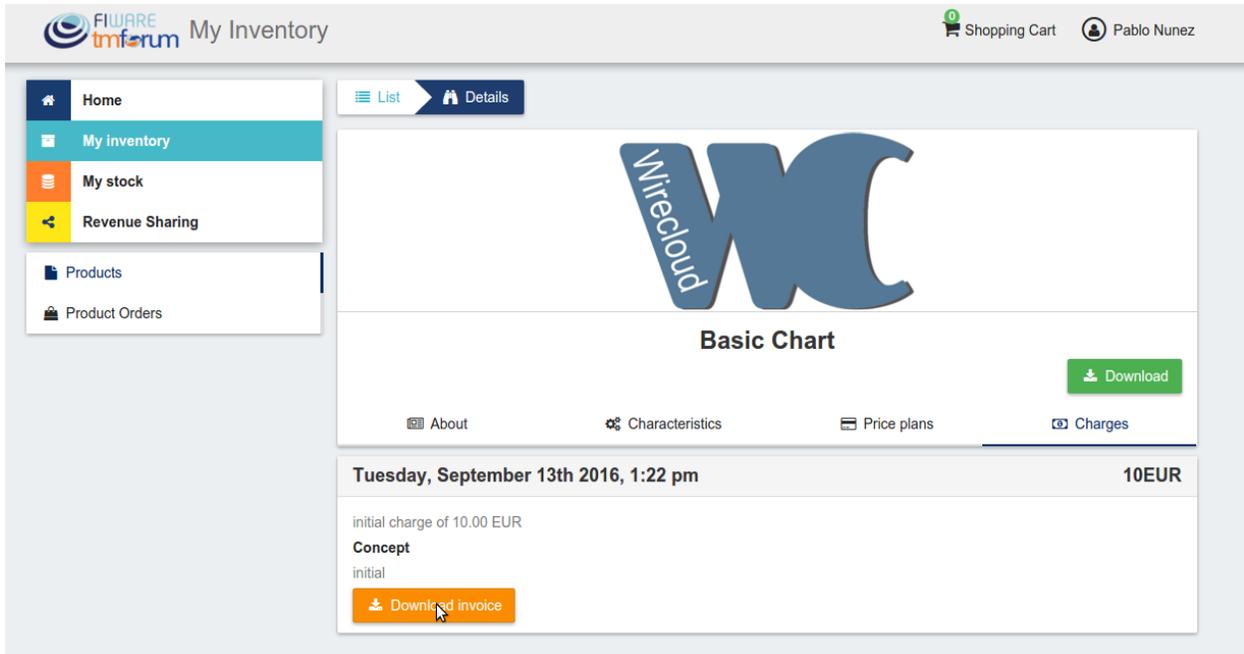
The screenshot shows the 'My Inventory' interface for 'Wirecloud WC' with the 'Charges' tab active. The 'Charges' section displays the following information:

General

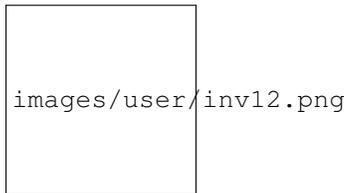
- Description:** No description provided.
- Offering:** Basic Chart
- Start date:** 14 minutes ago

The 'Download Invoice' button is visible in the top right corner of the 'Charges' section.

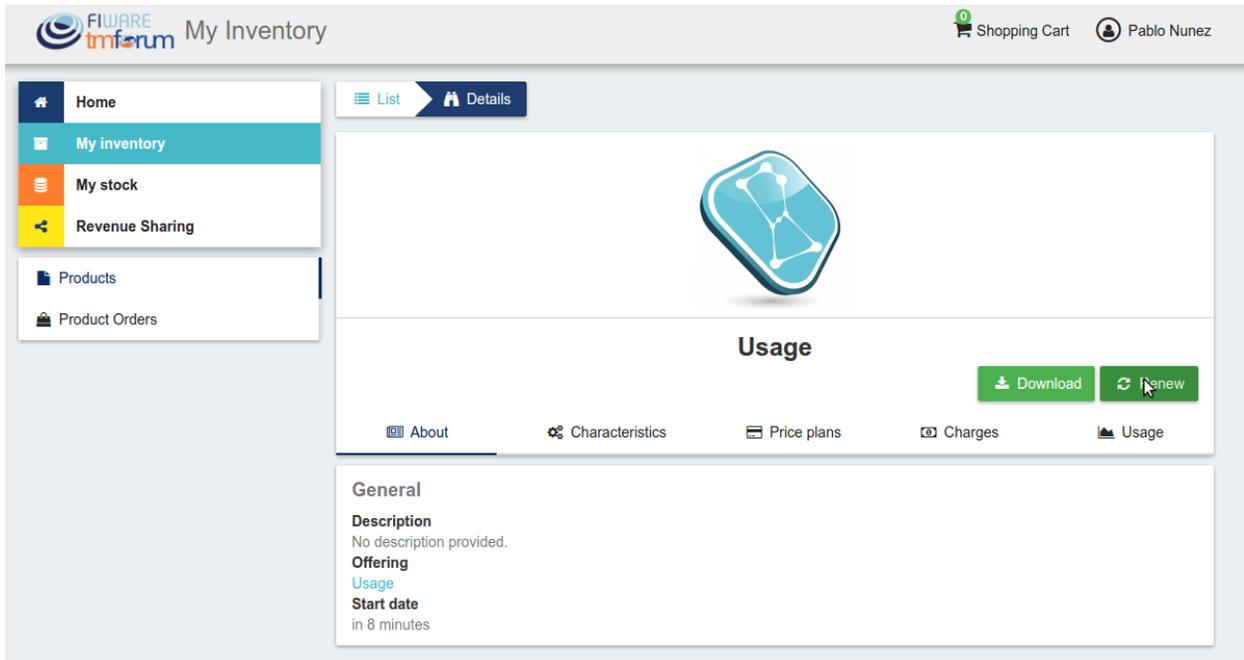
In this tab, you will find detailed information of the different charges and you will be able to download the related invoice clicking on *Download Invoice*



Moreover, this product view allows to download the related assets when the product is digital. To do that click on *Download*

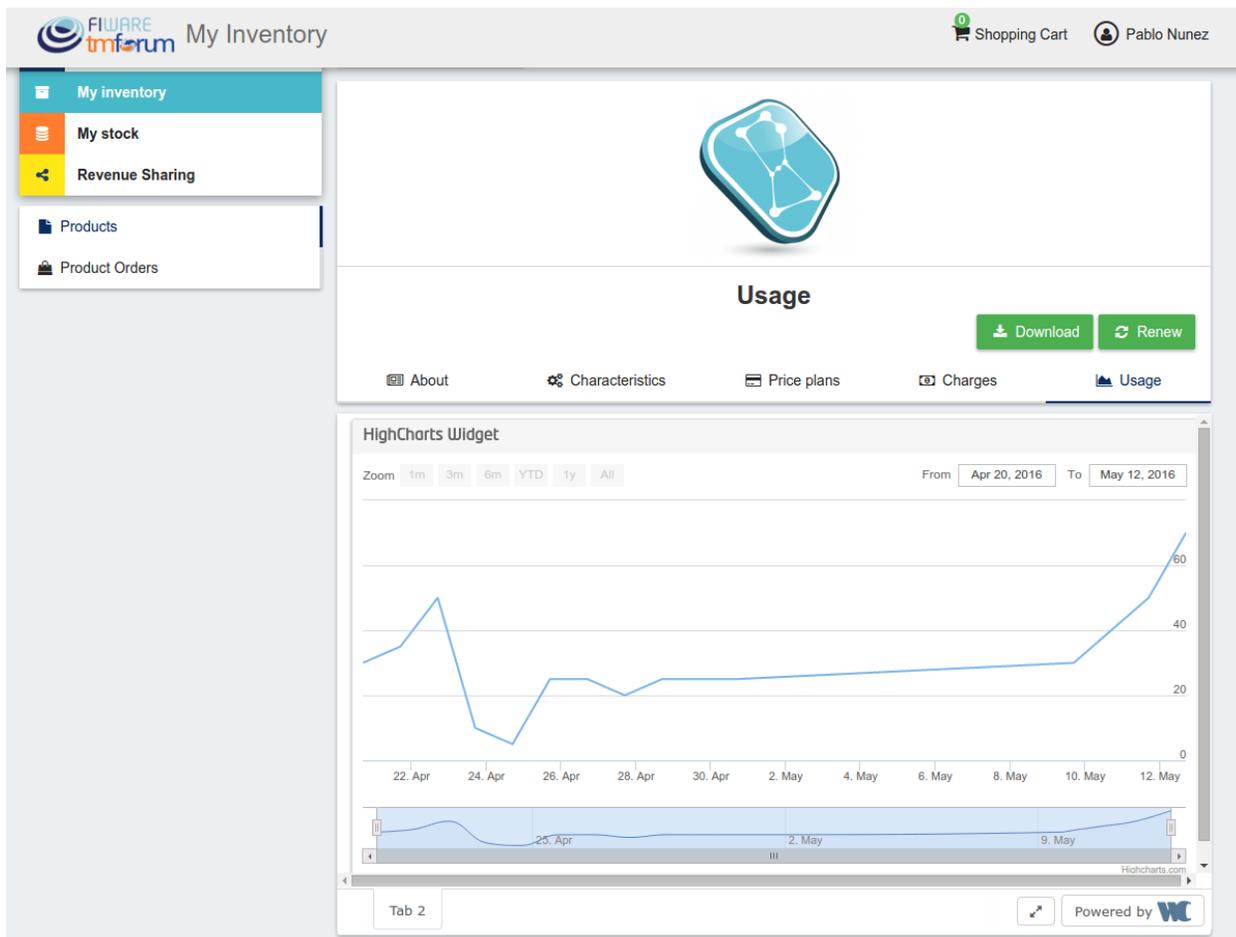


In case the chosen pricing model defines a recurring payment or a usage payment, you will be able to renew your product clicking on *Renew*. After clicking, you will be redirected to PayPal to pay the related amount.



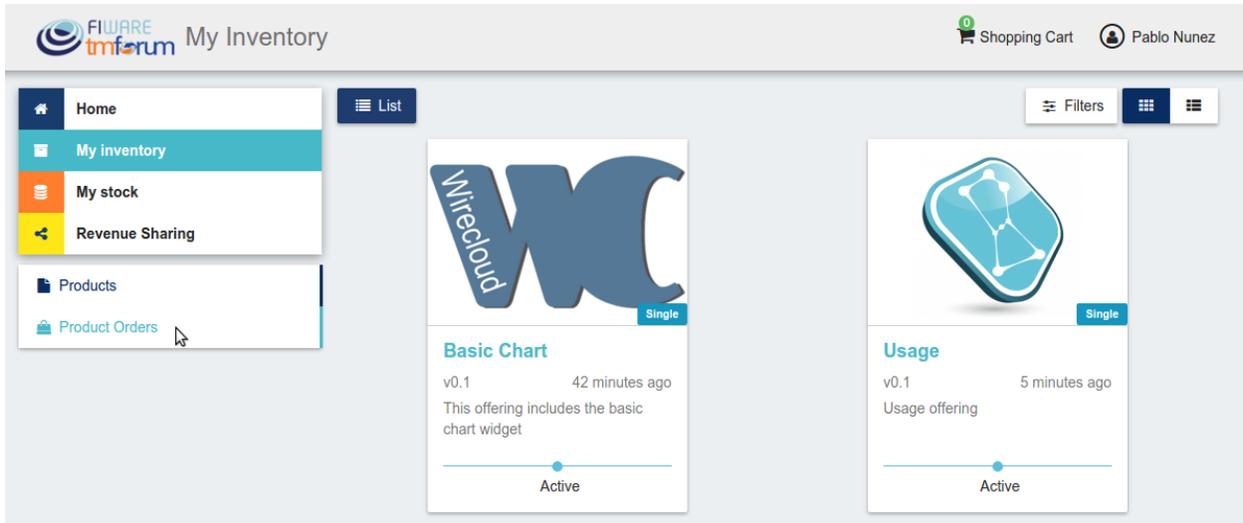
Note: If your product has expired and you do not renew it, it will be suspended, which means you will not have access to the acquired service until you pay

If the acquired product has a usage based price plan, you will be able to see your current consumption accessing the *Usage* tab

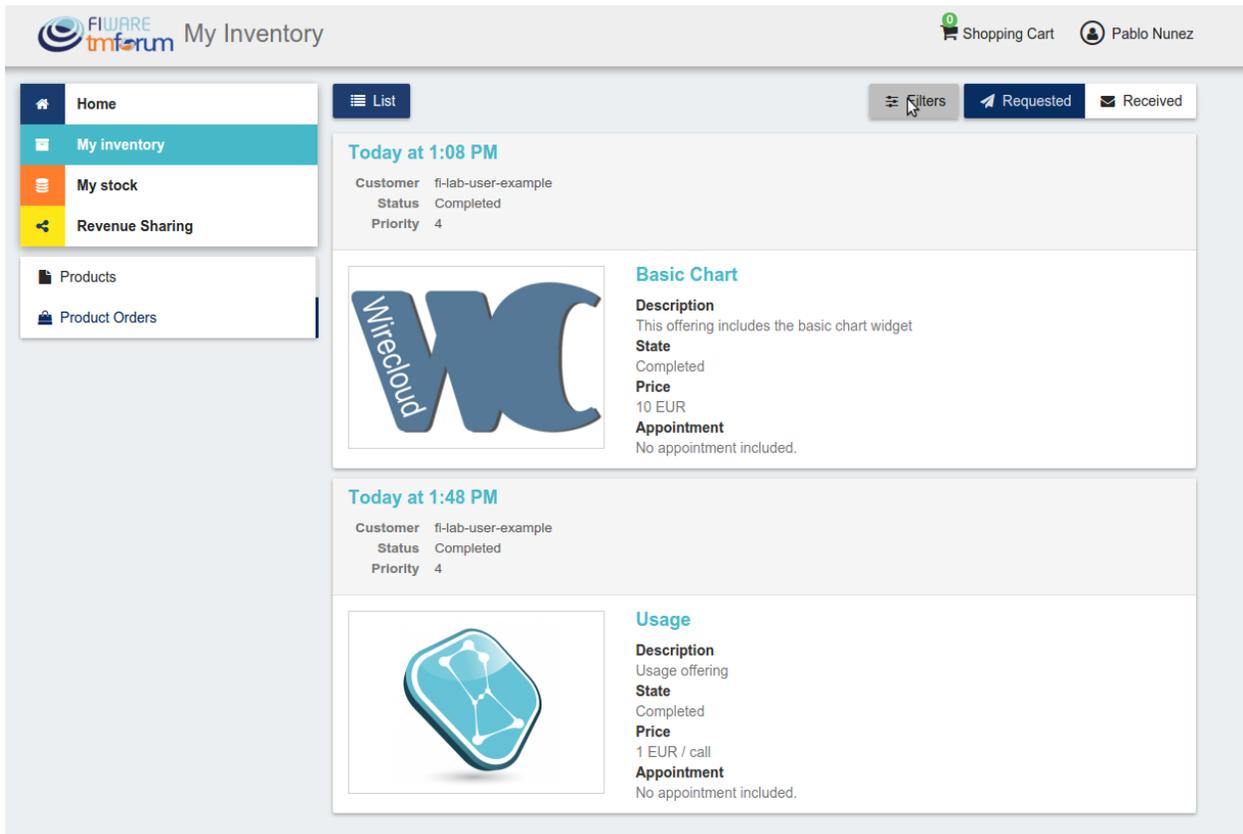


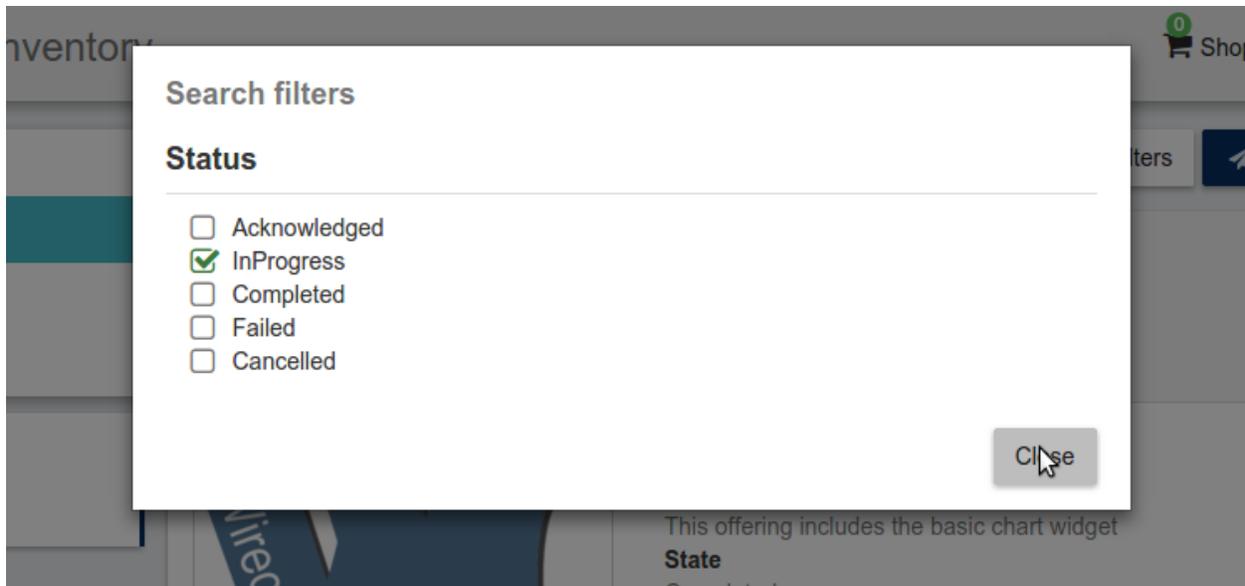
Manage Requested Orders

Customers can manage some aspects of the orders they have created. To see your requested orders, go to *My Inventory* and click on *Product Orders*

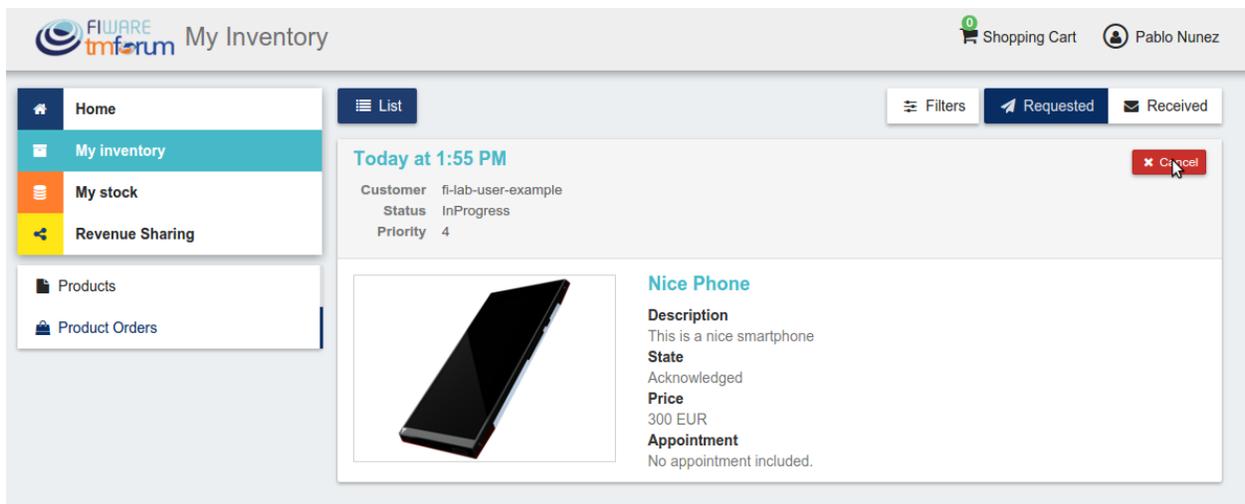


In the displayed view, you can see the orders you have created, which can be filtered by its status. To do that, click on *Filters*, select the wanted statuses, and click on *Close*





For those orders that include offerings of non digital products, you will be able to cancel them if the seller has not yet started the process. To do that, locate the order to be canceled and click on *Cancel*



Moreover, you can review the details of the order. To do that click on the date of the order.

The screenshot shows the 'My Inventory' application interface. On the left is a navigation menu with options: Home, My inventory (selected), My stock, Revenue Sharing, Products, and Product Orders. The main content area displays an order card titled 'Today at 1:55 PM' with a 'Cancel' button. The order details are: Customer (fi-lab-user-example), Status (InProgress), and Priority (4). Below the order card is a product card for 'Nice Phone' with a description, state (Acknowledged), price (300 EUR), and appointment (No appointment included).

In the displayed view, you can see all the details of the order, as well as the included products. In addition, you can leave a note for the seller in the *Notes* tab

This screenshot shows the 'Order details' view in the 'My Inventory' application. The navigation menu is the same as in the previous screenshot. The main content area is divided into two tabs: 'List' and 'Details' (selected). The 'Order details' section shows: Name (No data provided), Customer name (fi-lab-user-example), Notification email (pablo@email.com), Shipping address (Campus de Montegancedo S/N, 28041 Madrid (Madrid), Spain), Order date (Tuesday, September 13th 2016, 1:55 pm), Priority (4), Status (InProgress), Desired delivery date (Tuesday, September 13th 2016, 1:55 pm), and Expected delivery date (No data provided). Below this are two tabs: 'Products' and 'Notes'. The 'Products' tab is active, showing 'Product 1' with an image of the phone, offering name ('Nice Phone'), status (Acknowledged), vendor name (fdelavega), characteristics (Color: white), and price (300 EUR).

To leave a note, write it in the provided text area and click on the send button

The screenshot shows the 'My Inventory' application interface. On the left is a navigation menu with items: Home, My inventory (selected), My stock, Revenue Sharing, Products, and Product Orders. The main area is titled 'Order details' and contains the following information:

Name No data provided.	Order date Tuesday, September 13th 2016, 1:55 pm
Customer name fi-lab-user-example	Priority 4
Notification email pablo@email.com	Status InProgress
Shipping address Campus de Montegancedo S/N 28041 Madrid (Madrid) Spain	Desired delivery date Tuesday, September 13th 2016, 1:55 pm
	Expected delivery date No data provided

Below the order details is a 'Notes' section with a text input field containing the note: '! prefer the silver phone instead'.

1.2.3 Programmer Guide

The Business API Ecosystem allows to offer any kind of digital asset. In this regard, some kind of digital assets may require to perform specific actions and validations that require to know the format of the asset. To deal with this issue the Business API Ecosystem allows to register types of assets by creating plugins. This section explains how these plugins are created.

Additionally, the Business API Ecosystem exposes an API that can be used by developers in order to integrate the monetization features offered with their own solutions. The complete description of this API can be found in:

- [Apiary](#)
- [GitHub Pages](#)

Plugin Package

Business API Ecosystem plugins must be packaged in a zip. This file will contain all the sources of the plugin and a configuration file called *package.json* in the root of the zip. This configuration file allows to specify some aspects of the behaviour of the plugin and contains the following fields:

- **name:** Name given to the resource type. This is the field that will be shown to providers
- **author:** Author of the plugin.
- **formats:** List that specify the different allowed formats for providing an asset of the given type. This list can contain the values "URL" and "FILE".
- **module:** This field is used to specify the main class of the Plugin.
- **version:** Current version of the plugin.
- **media_types:** List of allowed media types that can be selected when providing an asset of the given type

Following you can find an example of a *package.json* file:

```
{
  "name": "Test Resource",
  "author": "fdelavega",
  "formats": ["FILE"],
  "module": "plugin.TestPlugin",
  "version": "1.0",
  "media_types": ["application/zip"]
}
```

The source code of the plugin must be written in Python and must contain a main class that must be a child class of the Plugin class defined in the Charging Backend of the Business API Ecosystem. Following you can find an example of a plugin main class.

```
from wstore.asset_manager.resource_plugins.plugin import Plugin

class TestPlugin(Plugin):
    def on_pre_product_spec_validation(self, provider, asset_t, media_type, url):
        pass

    def on_post_product_spec_validation(self, provider, asset):
        pass

    def on_pre_product_spec_attachment(self, asset, asset_t, product_spec):
        pass

    def on_post_product_spec_attachment(self, asset, asset_t, product_spec):
        pass

    def on_pre_product_offering_validation(self, asset, product_offering):
        pass

    def on_post_product_offering_validation(self, asset, product_offering):
        pass

    def on_product_acquisition(self, asset, contract, order):
        pass

    def on_product_suspension(self, asset, contract, order):
        pass
```

Implementing Event Handlers

It can be seen in the previous section that the main class of a plugin can implement some methods that are inherited from the Charging Backend Plugin class. This methods can be used to implement handlers of the different events of the life cycle of a product containing the asset. Concretely, the following events have been defined:

- **on_pre_product_spec_validation:** This method is executed when creating a new digital product containing an asset of the given type, before validating the product spec contents and saving the asset info in the database. This method can be used for validating the asset format or the seller permissions to sell the asset.
- **on_post_product_spec_validation:** This method is executed when creating a new digital product containing an asset of the given type, after validating the product spec and saving the asset info in the database. This method can be used if the plugin require to know some specific info of the asset model
- **on_pre_product_spec_attachment:** This method is executed when creating a new digital product containing an asset of the given type, after saving the product spec in the catalog API database but before attaching the

product spec id to the asset model. This method can be used if the plugin require to know the id in the catalog of the product spec

- **on_post_product_spec_attachment:** This method is executed when creating a new digital product containing an asset of the given type, after saving the product spec in the catalog API database and after attaching the product spec id to the asset model. This method can be used if the plugin require to know the id in the catalog of the product spec
- **on_pre_product_offering_validation:** This method is executed when creating a new product offering containing an asset of the given type, before validating its pricing model. This method can be used to make extra validations on the pricing model, for example check if the unit of an usage model is supported by the given asset
- **on_post_product_offering_validation:** This method is executed when creating a new product offering containing an asset of the given type, after validating its pricing model. This method can be used to make extra validations on the pricing model, for example check if the unit of an usage model is supported by the given asset
- **on_product_acquisition:** This method is called when a product containing an asset of the given type has been acquired. This method can be used to activate the service for the customer and give him access rights.
- **on_product_suspension:** This method is called when a product containing an asset of the given type has been suspended for a customer (e.g he has not paid). Tjis method can be used to suspend the service for the customer and remove his access rights

As can be seen in the Plugin example, the different handler methods receive some parameters with relevant information and objects. In particular:

on_pre_product_spec_validation

- **provider:** User object containing the user who is creating the product specification (The User object is described later)
- **asset_t:** String containing the asset type, it must be equal to the one defined in package.json
- **media_type:** String containing the media type of the asset included in the product being created
- **url:** String containing the url of the asset included in the product being created

on_post_product_spec_validation

- **provider:** User object containing the user who is creating the product specification (The User object is described later)
- **asset:** Asset object with the recently created asset (The Asset object is described later)

on_pre_product_spec_attachment

- **asset:** Asset object where the created product specification id is going to be attached
- **asset_t:** String containing the asset type, it must be equal to the one defined in package.json
- **product_spec:** JSON with the raw product specification information that is going to be used for the attachment. (The structure of this JSON object can be found in the Open Api documentation)

on_post_product_spec_attachment

- **asset:** Asset object where the created product specification id has been attached
- **asset_t:** String containing the asset type, it must be equal to the one defined in package.json
- **product_spec:** JSON with the raw product specification information that has been used for the attachment. (The structure of this JSON object can be found in the Open Api documentation)

on_pre_product_offering_validation

- **asset:** Asset object included in the offering being created
- **product_offering:** JSON with the raw product offering information that is going to be validated. (The structure of this JSON object can be found in the Open Api documentation)

on_post_product_offering_validation

- **asset:** Asset object included in the offering being created
- **product_offering:** JSON with the raw product offering information that has been validated. (The structure of this JSON object can be found in the Open Api documentation)

on_product_acquisition

- **asset:** Asset object that has been acquired
- **contract:** Contract object including the information of the acquired offering which contains the asset. (The Contract object is described later)
- **order:** Order object including the information of the order where the asset was acquired. (The Order object is described later)

on_product_acquisition

- **asset:** Asset object that has been suspended
- **contract:** Contract object including the information of the acquired offering which contains the asset
- **order:** Order object including the information of the order where the asset was acquired

Handler Objects

Following you can find the information regarding the different objects used in plugin handlers

- **User: Django model object with the following fields**
 - **username:** Username of the user
 - **email:** Email of the user
 - **complete_name:** Complete name of the user
- **Asset: Django model object with the following fields**
 - **product_id:** Id of the product specification which includes the asset

- **version**: Version of the product specification which includes the asset
- **provider**: User object of the user that created the asset
- **content_type**: media type of the asset
- **download_link**: URL of the asset if it is a service in an external server
- **resource_path**: Path to the asset file if it is uploaded in the server
- **resource_type**: Type of the asset as defined in the package.json file of the related plug-in
- **is_public**: If true the asset can be downloaded by any user without the need of acquiring it
- **meta_info**: JSON with any related information. This field is useful to include specific info from the plugin code

Additionally, it includes the following methods:

- **get_url**: Returns the URL where the asset can be accessed
- **get_uri**: Returns the url where the asset info can be accessed
- **Contract: Django model with the following fields**
 - **item_id**: Id of the order item which generated the current contract
 - **offering**: Offering object with the information of the offering acquired in the current contract (The offering object is described later)
 - **product_id**: Id of the inventory product created as a result if the acquisition of the specified offering
 - **pricing_model**: JSON with the pricing model that is used in the current contract for charging the customer who acquired the included offering
 - **last_charge**: Datetime object with the date and time of the last charge to the customer
 - **charges**: List of Charge objects containing the info of the different times the customer has been charged in the context of the current contract
 - **correlation_number**: Next expected correlation number for usage documents. This field is only used when the pricing model is usage
 - **last_usage**: Datetime object with the date and time of the last usage document received. This field is only used when the pricing model is usage
 - **revenue_class**: Product class of the involved offering for revenue sharing
 - **terminated**: Specified whether the contract has been terminated (the customer has no longer access to the acquired asset)
- **Offering: Django model with the following fields**
 - **off_id**: Id of the product offering
 - **name**: Name of the offering
 - **version**: Version of the offering
 - **description**: Description of the offering
 - **asset**: Asset offered in the offering
- **Charge Django model with the following fields**
 - **date**: Datetime object with the date and time of the charge
 - **cost**: Total amount charged

- **duty_free**: Amount charged without taxes
- **currency**: Currency of the charge
- **concept**: Concept of the charge (initial, renovation, usage)
- **invoice**: Path to the PDF file containing the invoice of the charge
- **Order: Django model with the following fields**
 - **order_id**: Id of the product order
 - **customer**: User object of the customer of the order
 - **date**: Datetime object with the date and time of the order creation
 - **tax_address**: JSON with the billing address used by the customer in the order
 - **contracts**: List of Contract objects, one for each offering acquired in the order

Additionally, it includes the following methods:

- **get_item_contract**: Returns a contract given an item_id
- **get_product_contract**: Returns a contract given a product_id

Managing Plugins

Once the plugin has been packaged in a zip file, the Charging Backend of the Business API Ecosystem offers some management command that can be used to manage the plugins.

When a new plugin is registered, The Business API Ecosystem automatically generates an id for the plugin that is used for managing it. To register a new plugin the following command is used:

```
python manage.py loadplugin TestPlugin.zip
```

It is also possible to list the existing plugins in order to retrieve the generated ids:

```
python manage.py listplugins
```

To remove a plugin it is needed to provide the plugin id. This can be done using the following command:

```
python manage.py removeplugin test-plugin
```

1.3 Plugins Guide

1.3.1 Introduction

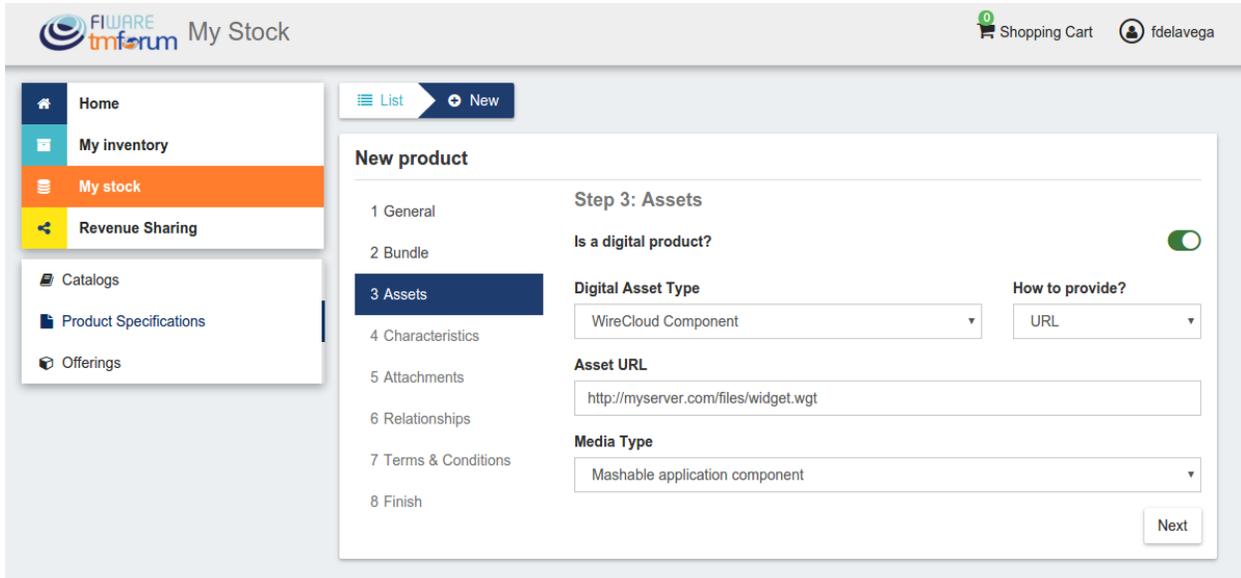
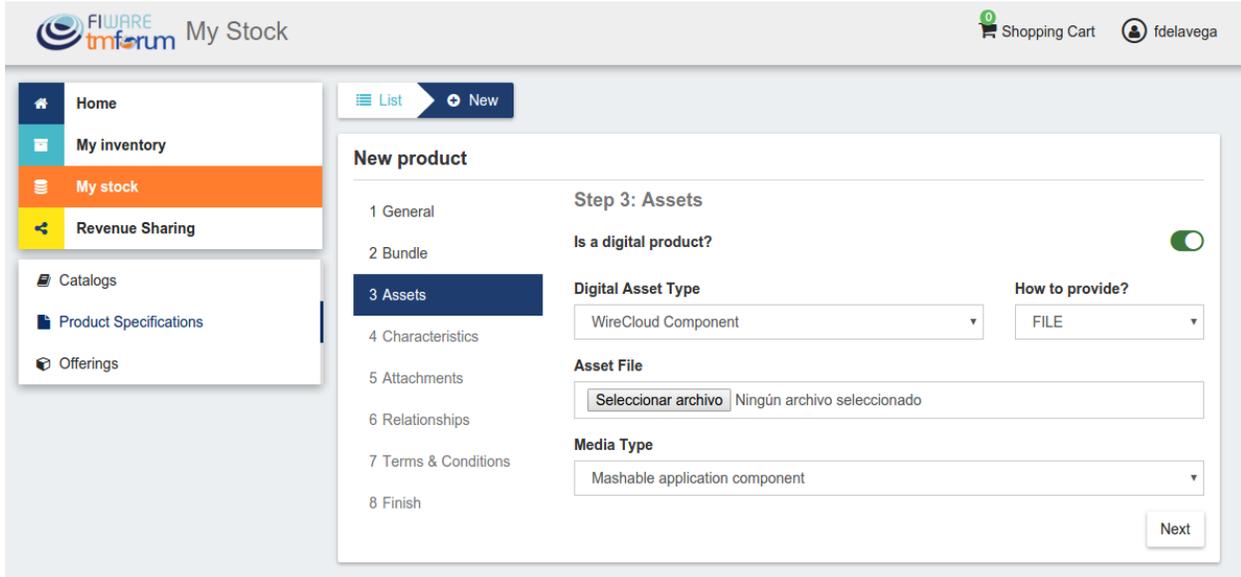
This plugins guide covers the available plugins (defining digital asset types) for the Business API Ecosystem v5.4.1. Any feedback on this document is highly welcomed, including bugs, typos or things you think should be included but aren't. Please send them to the "Contact Person" email that appears in the [Catalogue page for this GEi](#). Or create an issue at [GitHub Issues](#)

1.3.2 WireCloud Component

The *WireCloud Component* plugin is available in [GitHub](#). This plugin defines an asset type intended to manage and monetize the different WireCloud components (Widgets, Operators, and Mashups) in particular by enabling the creation of product specifications providing the WGT file of the specific component. (For more details on the WireCloud platform see its documentation in [ReadTheDocs](#))

The WireCloud component plugin allows to provide the WGT file in the two ways supported by the Business API Ecosystem, that is, uploading the WGT file when creating the product and providing a URL where the platform can download the file.

In addition, the plugin only allows the media type *Mashable application component*. Nevertheless, the plugin code uses the WGT metainfo to determine the type of the WireCloud component (Widget, Operator, or Mashup) and overrides the media type with the proper one understood by the WireCloud platform (*wirecloud/widget*, *wirecloud/operator* or *wirecloud/mashup*).



This plugin implements the following event handlers:

- **on_post_product_spec_validation:** In this handler the plugin validates the WGT file to ensure that it is a valid WireCloud Component
- **on_post_product_spec_attachment:** In this handler the plugin determines the media type of the WGT file and overrides the media type value in the specific product specification

1.3.3 CKAN Dataset

The *CKAN Dataset* plugin is available in [GitHub](#). This plugin defines an asset type intended to manage and monetize datasets offered in a CKAN instance. In particular, this plugin is able to validate the dataset, validate the rights of the seller creating a product specification to sell the provided dataset, and manage the access to the dataset of those customers who acquire it.

It is important to notice that by default CKAN does not provide a mechanism to publish protected datasets or an API for managing the access rights to the published datasets. In this regard, the CKAN instance to be monetized has to be extended with the following CKAN plugins:

- **ckanext-oauth2**: This extension allows to use an external OAuth2 Identity Manager for managing CKAN users. In particular, this extension must be used, in this context, to authenticate users using the same FIWARE IdM instance as the specific Business API Ecosystem instance, so both systems (CKAN and Business API Ecosystem) share their users.
- **ckanext-privatedatasets**: This extension allows to create protected datasets in CKAN which can only be accessed by a set of users selected by the dataset owner. Moreover, this extension exposes an API that can be used to add or remove authorized users from a dataset.

In addition, if the **ckanext-storepublisher** plugin is installed in CKAN, the *CKAN dataset* plugin must be installed in the Business API Ecosystem, since the aforementioned CKAN extension uses the *CKAN Dataset* asset type for creating product specifications.

The CKAN Dataset plugin only allows to provide the asset with a URL that must match the dataset URL in CKAN.

This plugin implements the following event handlers:

- **on_pre_product_spec_validation**: In this handler the plugin validates that the provided URL is a valid CKAN dataset and that the user creating the product specification is its owner.
- **on_product_acquisition**: In this handler the plugin uses the CKAN instance API in order to grant access to the user who has acquired a dataset.
- **on_product_suspension**: In this handler the plugin uses the CKAN instance API in order to revoke access to a dataset when a user has not paid or when the user cancels a subscription.

1.3.4 Accountable Service

The *Accountable Service* plugin is available in [GitHub](#). This plugin defines a generic asset type which is used jointly with the *Accounting Proxy* in order to offer services under a pay-per-use model. In particular, this plugin is able to validate services URLs, validate sellers permissions, generate API keys for the Accounting Proxy, validate offering pricing models, and manage customers access rights to the offered services.

Taking into account that this plugin is intended to work coordinately with an instance of the Accounting Proxy, all the assets to be registered using the *Accountable Service* type must be registered in the proxy as described in the Accounting Proxy section.

The *Accountable Service* plugin only allows to provide the assets with a URL that must match the service one.

This plugin implements the following event handlers:

- **on_post_product_spec_validation:** In this event handler the plugin validates that the provided URL belongs to a valid service registered in an instance of the Accounting Proxy, and that the user creating the product specification is its owner. In addition, this handler generates an API key for the Accounting Proxy to be used when it feeds the Business API Ecosystem with accounting information.
- **on_post_product_offering_validation:** In this event handler the plugin validates the pricing model of a product offering where the service is going to be sold. Specifically, it validates that all the price plans which can be selected by a customer are usage models and that the units (calls, seconds, mb, etc) are supported by the Accounting Proxy.
- **on_product_acquisition:** This event handler is used to grant access to a user who has acquired a service by sending a notification to the proxy, including also the unit to be accounted (price plan selected).
- **on_product_suspension:** This event handler is used to in order to revoke access to a service when a user has not paid or when the user cancels a subscription.

Accounting Proxy

The *Accounting Proxy* can be found in [GitHub](#). This software is a NodeJs server intended to manage services offered in the Business API Ecosystem. In particular, it is able to authenticate users, authorize or deny users to access to a particular service depending on the acquisition, the URL, or the HTTP method used, and account the usage made of the service so users can be charged on pay-per-use basis.

Having this software deployed allows service owners to protect their services and offer them in the Business API Ecosystem without the need of making any modification in the specific service.

Installation

This software is a pure NodeJS server, to install basic dependencies execute the following command:

```
$ npm install
```

Configuration All the Accounting Proxy configuration is saved in the `config.js` file in the root of the project.

In order to have the accounting proxy running it is needed to fill the following information:

- **`config.accounting_proxy`: Basic information of the accounting deployment.**
 - **`https`**: set this variable to undefined to start the service over HTTP.
 - * `enabled`: set this option to true to start the service over HTTPS and activate the certificate validation for some administration requests (see *Proxy API*).
 - * `certFile`: path to the server certificate in PEM format.
 - * `keyFile`: path to the private key of the server.
 - * `caFile`: path to the CA file.
 - `port`: port where the accounting proxy server is listening.

```
{
  https: {
    enabled: true,
    certFile: 'ssl/server1.pem',
    keyFile: 'ssl/server1.key',
    caFile: 'ssl/fake_ca.pem'
  },
  port: 9000
}
```

- **`config.database`: Database configuration used by the proxy.**
 - `type`: database type. Two possible options: `./db` (sqlite database) or `./db_Redis` (redis database).
 - `name`: database name. If the database type select is redis, then this field selects the database number (0 to 14; 15 is reserved for testing).
 - `redis_host`: redis database host.
 - `redis_port`: redis database port.

```
{
  type: './db',
  name: 'accountingDB.sqlite',
  redis_host: 'localhost',
  redis_port: 6379
}
```

- **`config.modules`: An array of supported accounting modules for accounting in different ways. Possible options are:**
 - `call`: the accounting is incremented in one unit each time the user send a request.
 - `megabyte`: counts the response amount of data (in megabytes).

- *millisecond*: counts the request duration (in milliseconds).

```
{
  accounting: [ 'call', 'megabyte', 'millisecond' ]
}
```

Other accounting modules can be implemented and included to the proxy (see *Accounting modules*).

- ***config.usageAPI***: the information of the usage management API where the usage specifications and the accounting information.
 - * *host*: Business API Ecosystem host. * *port*: Business API Ecosystem port. * *path*: path of the usage management API. * *schedule*: defines the daemon service schedule to notify the accounting information to the Business API Ecosystem. The format is similar to the cron tab format: “MINUTE HOUR DAY_OF_MONTH MONTH_OF_YEAR DAY_OF_WEEK YEAR (optional)”. By the default, the usage notifications will be sent every day at 00:00.

```
{
  host: 'localhost',
  port: 8080,
  path: '/DSUsageManagement/api/usageManagement/v2',
  schedule: '00 00 * * *'
}
```

- *config.api.administration_paths*: configuration of the administration paths. Default accounting paths are:

```
{
  api: {
    administration_paths: {
      keys: '/accounting_proxy/keys',
      units: '/accounting_proxy/units',
      newBuy: '/accounting_proxy/newBuy',
      checkURL: '/accounting_proxy/urls',
      deleteBuy: '/accounting_proxy/deleteBuy'
    }
  }
}
```

The Accounting Proxy can be used to proxy an Orion Context Broker, supporting the accounting of subscriptions. To do that, the following configuration params are used:

- ***config.resources***: configuration of the resources accounted by the proxy.
 - *contextBroker*: set this option to *true* if the resource accounted is an Orion Context Broker. Otherwise set this option to *false* (default value).
 - *notification_port*: port where the accounting proxy is listening to subscription notifications from the Orion Context Broker (port 9002 by default).

```
{
  contextBroker: true,
  notification_port: 9002
}
```

Administration

The Accounting Proxy is able to manage multiple services. In this regard, it has been provided a *cli* tool that can be used by admins in order to register, delete, and manage its services. The available commands are:

- `./cli addService [-c | -context-broker] <publicPath> <url> <appId> <httpMethod> [otherHttpMethods...]`: This command is used to register a new service in the Accounting Proxy. It receives the following parameters

- *publicPath*: Path where the service will be made available to external users. There are two valid patterns for the public path: (1) Providing a path with a single component (*/publicpath*) will make the Accounting Proxy accept requests to sub-paths of the specified one (i.e having a public path */publicpath* requests to */publicpath/more/path* are accepted). This pattern is typically used when you are offering the access to an API with multiple resources. (2) Providing a complete path (*/this/is/the/final/resource/path?color=Blue&shape=rectangular*) will make the Accounting Proxy to accept only requests to the exact registered path including query strings. This pattern is typically used when you are offering a single URL, like a Context Broker query.
- *url*: URL where your service is actually running and where requests to the proxy will be redirected. Note that in this case all the URL is provided (including the host) since the accounting proxy allows the management of services running in different servers.
- *appId*: ID of the service given by the FIWARE IdM. This id is used in order to ensure that the access tokens provided by users are valid for the accessed service
- *HTTP methods*: List of HTTP methods that are allowed to access to the registered service
- **Options:**
 - * *-c, -context-broker*: the service is an Orion Context broker service (*config.contextBroker* must be set to *true* in *config.js*).

Following you can find two examples in order to clarify the options available for registering a service:

```
$ ./cli addService /apacheapp http://localhost:5000/ 1111 GET PUT POST
```

In this case, there is a service running in the port 5000 which is made available though the */apacheapp* path, allowing only GET, PUT, and POST HTTP request. Supposing that the Accounting Proxy is running in the host *accounting.proxy.com* in the port 8000, the following requests will be accepted by it:

```
GET http://accounting.proxy.com:8000/apacheapp
GET http://accounting.proxy.com:8000/apacheapp/resource1/
POST http://accounting.proxy.com:8000/apacheapp/resource1/resource2
```

Note: The Accounting Proxy does not care about the API or the semantics of the monitored service, so it may accept a request to a URL which does not exist in the service, resulting in a usual 404 error given by the later

Additionally, a complete path can be provided, as in the following example:

```
$ ./cli addService /broker/v1/contextEntities/Room2/attributes/temperature http://localhost:1026/v1/
```

In this example, there is a Context Broker running in the port 1026 and a specific query is made available through the Accounting proxy, so only the following request is accepted:

```
GET http://accounting.proxy.com:8000/broker/v1/contextEntities/Room2/attributes/temperature
```

Note: For making the proxy transparent to final users is a good practice to use the same path in the external path and in the URL when providing a complete path. Nevertheless, this is not mandatory, so it is possible to create an alias for a query (i.e */room2/temperature* for the previous example)

- *./cli getService [-p <publicPath>]*: This command is used to retrieve the URL, the application ID and the type (Context Broker or not) of all registered services.

– **Options:**

- * *-p, -publicPath <path>*: only displays the information of the specified service.

- `./cli deleteService <publicPath>`: This command is used to delete the service associated with the public path.
- `./cli addAdmin <userId>`: This command is used to add a new administrator.
- `./cli deleteAdmin <userId>`: This command is used to delete the specified admin.
- `./cli bindAdmin <userId> <publicPath>`: This command is used to add the specified administrator to the service specified by the public path.
- `./cli unbindAdmin <userId> <publicPath>`: This command is used to delete the specified administrator for the specified service by its public path.
- `./cli getAdmins <publicPath>`: This command is used to display all the administrators for the specified service.

To display a brief description of the `cli` tool you can use : `./cli -h` or `./cli -help`. In addition, to get information for a specific command you can use: `./cli help [cmd]`.

Authentication and Authorization

The Accounting Proxy relies on the FIWARE IdM for authenticating users. To do that, the proxy expects that all the requests include a header `Authorization: Bearer access_token` or `X-Auth-Token: access_token` with a valid access token given by the IdM.

Moreover, if the authentication process has succeed, the Accounting Proxy validates the permissions of the user to access to specific service. To do that, it checks if the user has been registered as an admin of the service or if the user has acquired the service.

Is important to notice, that the Business API Ecosystem allows sellers to offer a service in different offerings with different pricing models. In this regard, having just the access token is not enough to determine the accounting unit (pricing model) that has to be used to account the usage of the service. It may happen, that a valid user has acquired the access to a service in two different offerings with two different models (i.e calls and seconds), so the proxy needs extra info to determine the unit to account (in this example calls or seconds). To deal with that problem, the Accounting Proxy generates an API Key which identifies the service, the user, and the accounting unit, so including it in a header `X-API-Key: api_key` when making requests, enables it to know what unit to account.

Note: The X-API-Key header is not intended to provide an extra level of security, but just to remove the possible incertitude around the request

Proxy API

The Accounting Proxy runs by default in the port 9000; nevertheless, this port can be configured as described in *Configuration* section. In this regard, the different services configured though the administration `cli` tool can be accessed directly in the root of the proxy using the public path defined for the service.

In addition, the Accounting Proxy has an administration API which can be accessed though the reserved path `/accounting_proxy`. Following, you can find the different services exposed in the administration API:

POST .../newBuy This service is used by the Business API Ecosystem to notify a new buy. If the accounting proxy has been started over HTTPS, these requests should be signed with the Business API Ecosystem key; otherwise, they will be rejected.

```
{
  "orderId": "...",
  "productId": "...",
  "customer": "...",
```

```
"productSpecification": {
  "url": "...",
  "unit": "...",
  "recordType": "..."
}
}
```

- *orderId*: order identifier.
- *productId*: product identifier.
- *customer*: customer id.
- *url*: base url of the service.
- *unit*: accounting unit (*megabyte*, *call*, etc).
- *recordType*: type of accounting.

POST .../deleteBuy This service is used by the Business API Ecosystem to notify a terminated buy. If the accounting proxy has been started over HTTPS, these requests should be signed with the Business API Ecosystem key; otherwise, they will be rejected.

```
{
  "orderId": "...",
  "productId": "...",
  "customer": "...",
  "productSpecification": {
    "url": "..."
  }
}
```

- *orderId*: order identifier.
- *productId*: product identifier.
- *customer*: customer id.
- *url*: base url of the service.

POST .../urls This service is used by the Business API Ecosystem to check if an URL is a valid registered service. This requests require the “authorization” header with a valid access token from the IdM and the user must be an administrator of the service. If the accounting proxy has been started over HTTPS, these requests should be signed with the Business API Ecosystem key cert; otherwise, they will be rejected.

```
{
  "url": "..."
}
```

GET .../keys Retrieve the user’s API_KEYS in a json. This request require the “authorization” header with a valid access token from the IdM.

```
[
  {
    "apiKey": "...",
    "productId": "...",
    "orderId": "...",
    "url": "..."
  }
]
```

```

    },
    {
      "apiKey": "...",
      "productId": "...",
      "orderId": "...",
      "url": "..."
    }
  ]

```

GET .../units Retrieve the supported accounting units by the accounting proxy in a JSON. This requests require the “authorization” header with a valid access token from the IdM.

```

{
  "units": ["..."]
}

```

Accounting modules

By default, the Accounting Proxy includes three different modules for accounting. Nevertheless, it is possible to extend the proxy with new modules by creating them in the *acc_modules* directory, those modules have to have the following structure:

```

/** Accounting module for unit: XXXXXX */

var count = function (countInfo, callback) {
  // Code to do the accounting goes here
  // .....

  return callback(error, amount);
}

var getSpecification = function () {
  return specification;
}

```

The function *count* receives two parameters: * *countInfo*: object containing both, the request made by the user and the response returned by the service

```

{
  request: { // Request object used by the proxy to make the request to the service.
    headers: {

    },
    body: {

    },
    ...
  },
  response: { // Response object received from the service.
    headers: {

    },
    body: {

    },
    elapsedTime: , // Response time
  }
}

```

```
    ...  
  }  
}
```

- **callback**: function, which is used to retrieve the accounting value or the error message. The callback expects 2 parameters
 - *error*: string with a description of the error if there is one. Otherwise, *null*.
 - *amount*: number with the amount to be added to the current accounting.

The function *getSpecification* should return a javascript object with the usage specification for the accounting unit according to the TMF635 usage management API ([TMF635 usage Management API](#)).

Finally, add the name of the developed accounting module to the *config.modules* array in the *config.js* file (the accounting module name is the name of the file, e.g. *megabyte* and *megabyte.js*) and restart the Accounting Proxy.