# Business API Ecosystem Documentation

***Release latest***

**Oct 06, 2023**

# DOCUMENTATION

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.

In particular, 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.

The current documentation covers the Business API Ecosystem version 8.1.0, corresponding to FIWARE release 8. 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

CHAPTER

ONE

# INDEX

*Installation and Administration Guide*
: The guide for maintainers that explains how to install the BAE.

*Configuration Guide*
: The guide for administrations which explains the different configuration options

*User Guide*
: The guide for users that explains how to use it.

*Programmer Guide*
: The guide for programmers that explains how to develop plugins

*Plugins Guide*
: The guide for admins that cover the available plugins

## 1.1 Installation and Administration Guide

This guide covers the installation of the Business API Ecosystem (BAE) version 8.1.0. The recommended procedure for the installation of the Business API Ecosystem is using Docker and the Docker images available in Docker Hub.

### 1.1.1 Installation with Docker

The installation with Docker requires the following:

- Docker
- Docker Compose

As stated, the Business API Ecosystem in made up of a set of different components which work jointly in order to provide the functionality. In this regard the following images have been defined:

- fiware/biz-ecosystem-apis: This image includes all the TMForum APIs and can be found in Docker Hub
- fiware/biz-ecosystem-charging-backend: This image includes the Charging Backend component and can be found in Docker Hub
- fiware/biz-ecosystem-logic-proxy: This image includes the Logic Proxy component and can be found in Docker Hub
- fiware/biz-ecosystem-rss: This Image include the Revenue Sharing Component and can be found in Docker Hub

### Local BAE deployment

The easiest way to deploy the Business API Ecosystem with Docker is using *Docker Compose*. There have been created a docker compose file that allows to deploy the Business API Ecosystem locally. Such a file can be found here

The local BAE repository deploys all the BAE components as well as a Keyrock instance that can be used as IDM. By default, local BAE requires an external network called *bae* that will be used by the different BAE components for communications. That network can be created with the following command:

```
docker network create bae
```

As an alternative, local BAE can be configured using the bridge driver, meaning that the different components will be assigned a port in the host machine and a local IP to access them. The bridge mode can be enabled uncommeting driver and IP configuration:

```
networks:
    bae:
        name: bae
        external: false
        driver: bridge
        ipam:
        config:
          - subnet: 10.2.0.0/16
```

The BAE can be launched with:

```
docker compose up -d
```

And terminated with:

```
docker compose down
```

The local BAE repository includes database initializations that will create a Marketplace application within Keyrock and having some pre-configured settings. In this regard, the local BAE will be ready to use in a local environment without further configuration.

As soon as the Logic Proxy component of the BAE is healthy, the marketplace page can be accessed in the 8004 of the host machine. The login can be done through the pre-configured Keyrock IDP using the initial test credentials:

```
Username: admin@test.com
Password: admin
```

New users can be created directly in the Keyrock instance available in the port 8080 of the host machine.

The configuration of the BAE can be updated using environment variables by updating the .env file or the environment files included in envs/ directory. For details on the different configuration options please refer to the

*Configuration Guide*

### Data storage

The different images used as part of the Business API Ecosystem provide several volumes. Following it is described the different options available in each image.

The **biz-ecosystem-logic-proxy** image defines 2 volumes. In particular:

- */business-ecosystem-logic-proxy/themes*: This volume includes the different themes that can be used to customize the portal
- */business-ecosystem-logic-proxy/static*: This volume includes the static files ready to be rendered including the selected theme and js files

Additionally, the **biz-ecosystem-logic-proxy** image defines two environment variables intended to optimize the production deployment of the BAE Logic proxy:

- *NODE_ENV*: Specifies whether the system is in *development* or in *production* (default: development)
- *COLLECT*: Specifies if the container should execute the collect static command to generate static files or use the existing on start up (default: True)

On the other hand, the **biz-ecosystem-charging-backend** image defines 4 volumes. In particular:

- */business-ecosystem-charging-backend/src/media/bills*: This directory contains the PDF invoices generated by the Business Ecosystem Charging Backend
- */business-ecosystem-charging-backend/src/media/assets*: This directory contains the different digital assets uploaded by sellers to the Business Ecosystem Charging Backend
- */business-ecosystem-charging-backend/src/plugins*: This directory is used for providing asset plugins (see section *Installing Asset Plugins*)
- */business-ecosystem-charging-backend/src/wstore/asset_manager/resource_plugins/plugins*: This directory includes the code of the plugins already installed

### Installing Asset Plugins

As you may know, the Business API Ecosystem is able to sell different types of digital assets by loading asset plugins in its Charging Backend. In this context, it is possible to install asset plugins in the current Docker image as follows:

1) Copy the plugin file into the host directory of the volume */business-ecosystem-charging-backend/src/plugins*
2) Access the running container:

```
docker exec -i -t your-container bash
```

3) Go to the installation directory

```
cd /business-ecosystem-charging-backend/src
```

4) Load the plugin

```
python3 manage.py loadplugin ./plugins/pluginfile.zip
```

5) Restart the docker image

```
docker compose restart bae-charging
```

### 1.1.2 Manual 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 providing business capabilities. In this regard, this section contains the basic dependencies of the different components that made up the Business API Ecosystem.

#### TM Forum APIs and RSS requirements

- Java 8
- Glassfish 4.1
- MySQL 5.7

#### Charging Backend requirements

- Python 3.9
- MongoDB 4.4+
- wkhtmltopdf

#### Logic Proxy requirements

- NodeJS 16+ (Including NPM)
- Elasticsearch 7.5+

#### Installation

#### 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
```

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 --property user=<user>:password=<passwd>:URL=jdbc:mysql://<host>:
↪<port>/<database> <poolname>
```

**Note:** You have to provide you 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 v8.0.0
```

Then, the next step is coping, *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/DSRevenueSharing.war
```

### 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
```

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 dependecies use pip tool

```
$ pip3 install -r requirements.txt
```

If you are planning to run the tests or develop, you should install the development dependecies:

```
$ pip3 install -r dev-requirements.txt
```

### Installing the Logic Proxy

The Logic Proxy sources can be found in GitHub

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
```

Once the code has been downloaded, Node dependencies can be installed with NPM

```
$ npm install
```

### Final steps

### Media and Indexes

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/media*, *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 Elasticsearch indexes for efficiency and pagination. You can populate at any time the indexes directory using the *fill_indexes.js* script provided with the Logic Proxy.

```
$ node fill_indexes.js
```

#### 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 an running before starting it.

**Using Django runserver**

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

```
$ python3 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 Gunicorn**

The Charging Backend can be deployed in production using Gunicorn. To do that execute the following command

```
$ gunicorn wsgi:application --workers 1 --forwarded-allow-ips "*" --log-file - --bind 0.
↪0.0.0:8006 --log-level INFO
```

#### Running the Logic Proxy

The Logic Proxy can be started using Node as follows

```
$ node server.js
```

### 1.1.3 Sanity check Procedures

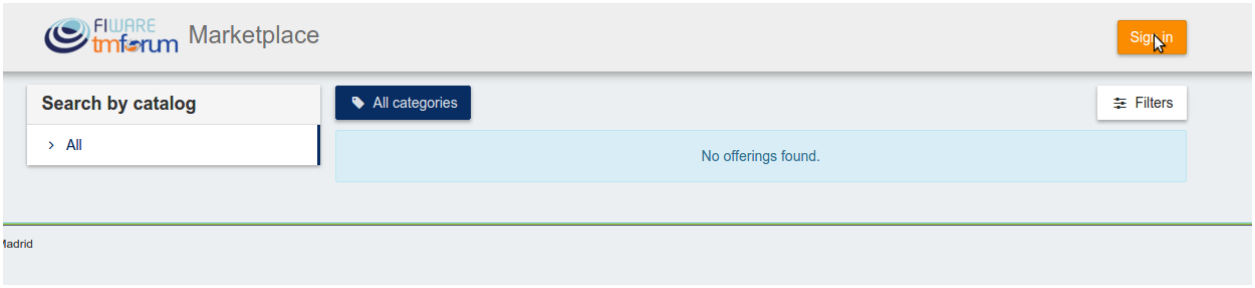
The Sanity Check Procedures are the steps that a System Administrator will take to verify that an installation is ready to be tested. This is therefore a preliminary set of tests to ensure that obvious or basic malfunctioning is fixed before proceeding to unit tests, integration tests and user validation.

### End to End Testing

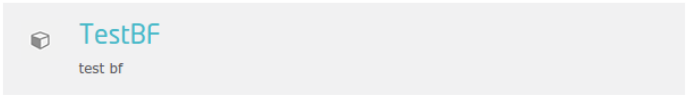
Please note that the following information is required before starting with the process: * The host and port where the Proxy is running * A valid IdM user with the *Seller* role

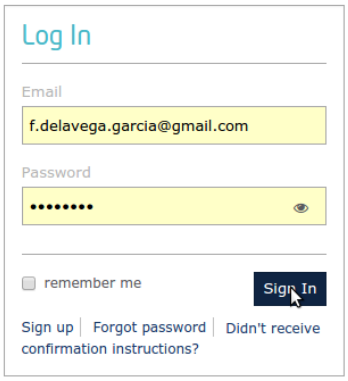
To Check if the Business API Ecosystem is running, follow the next steps:

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

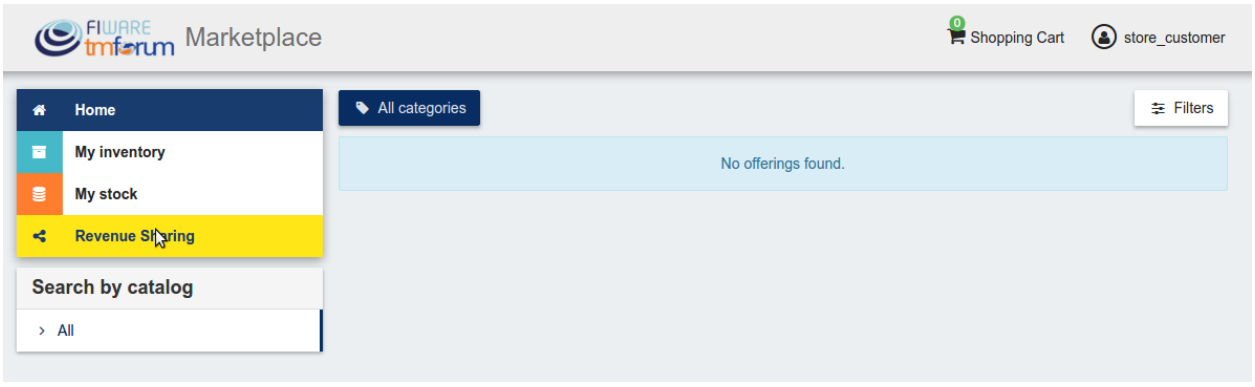



3. Provide your credentials in the IdM page

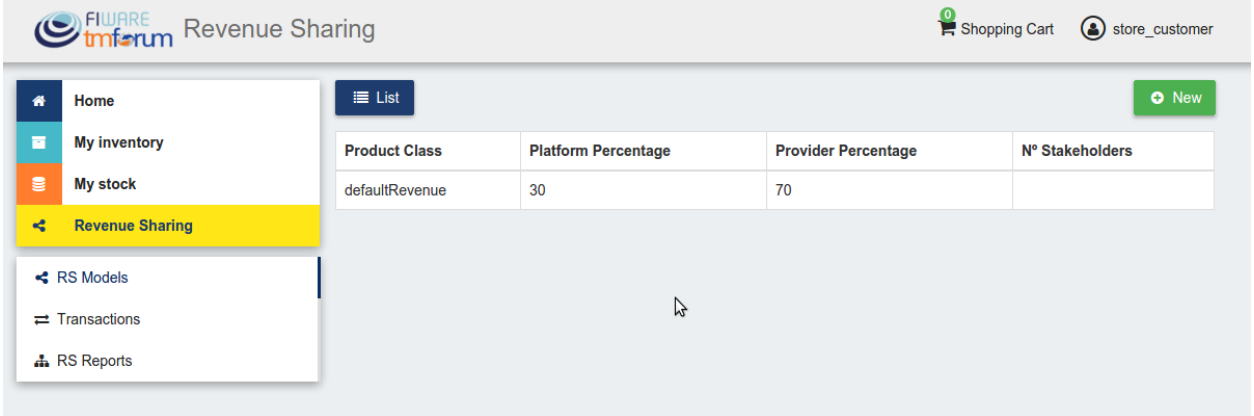


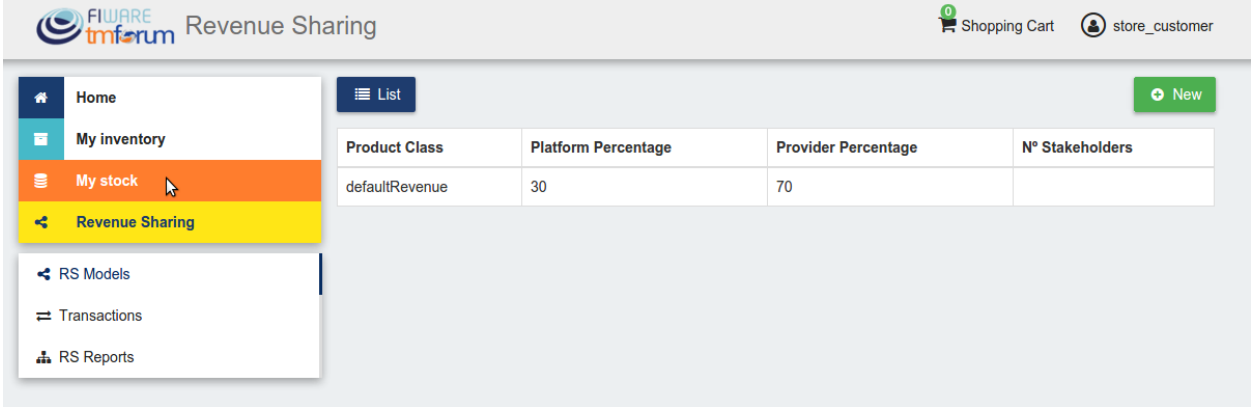

4. Go to the *Revenue Sharing* section



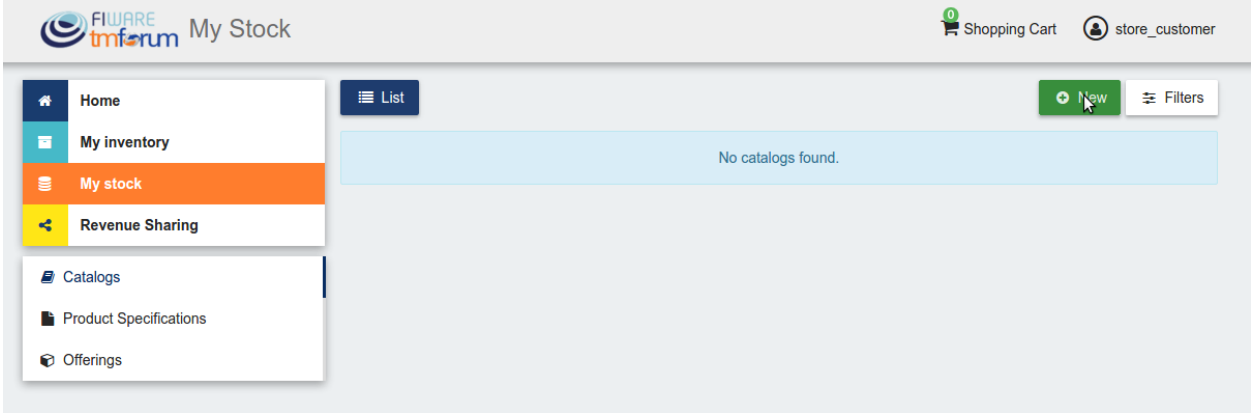

5. Ensure that the default RS Model has been created



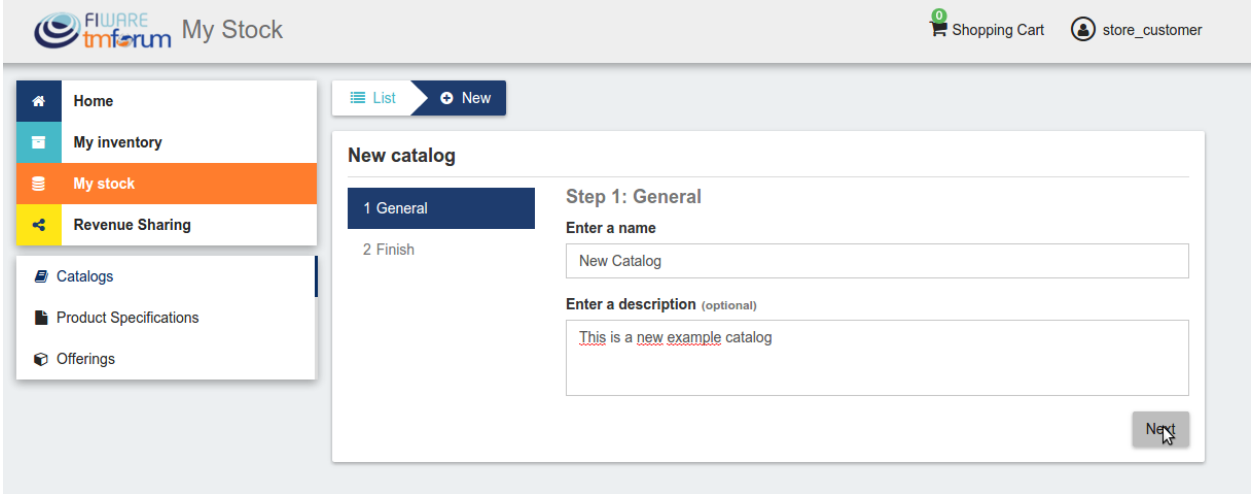

6. Go to *My Stock* section

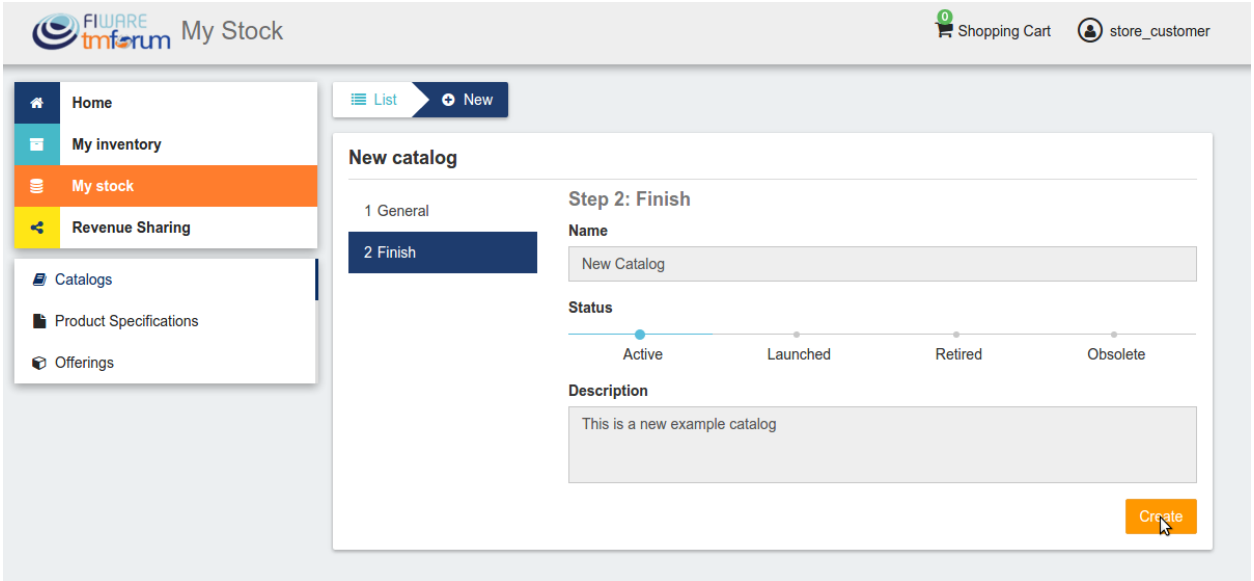


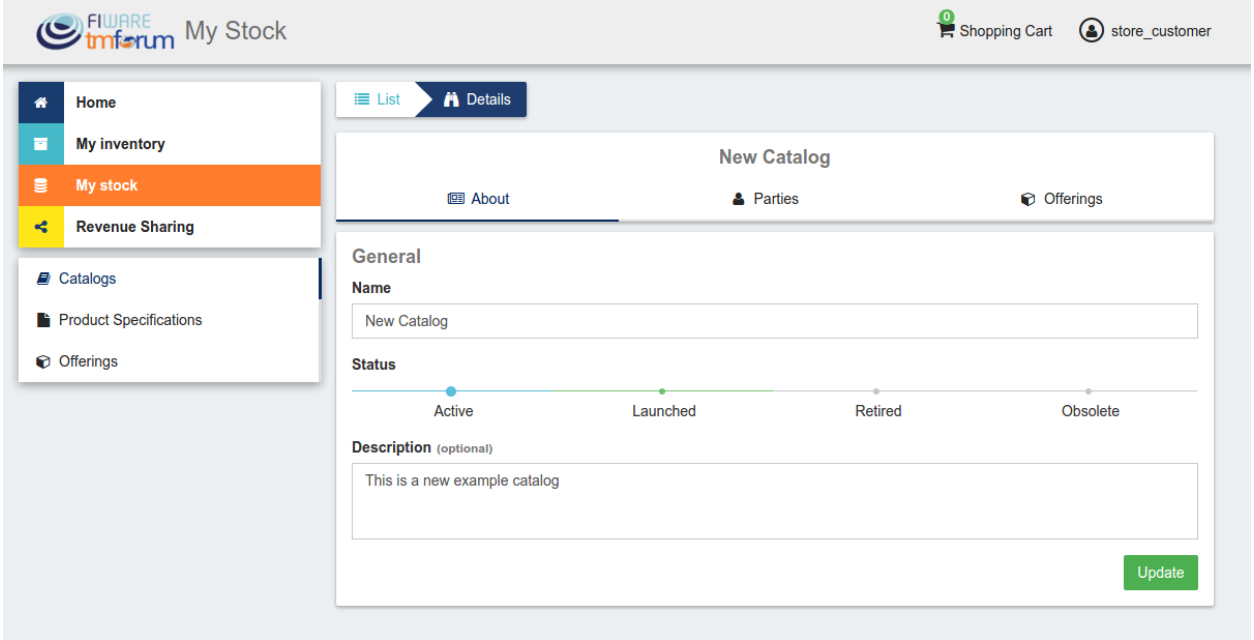

7. Click on *New* for creating a new catalog



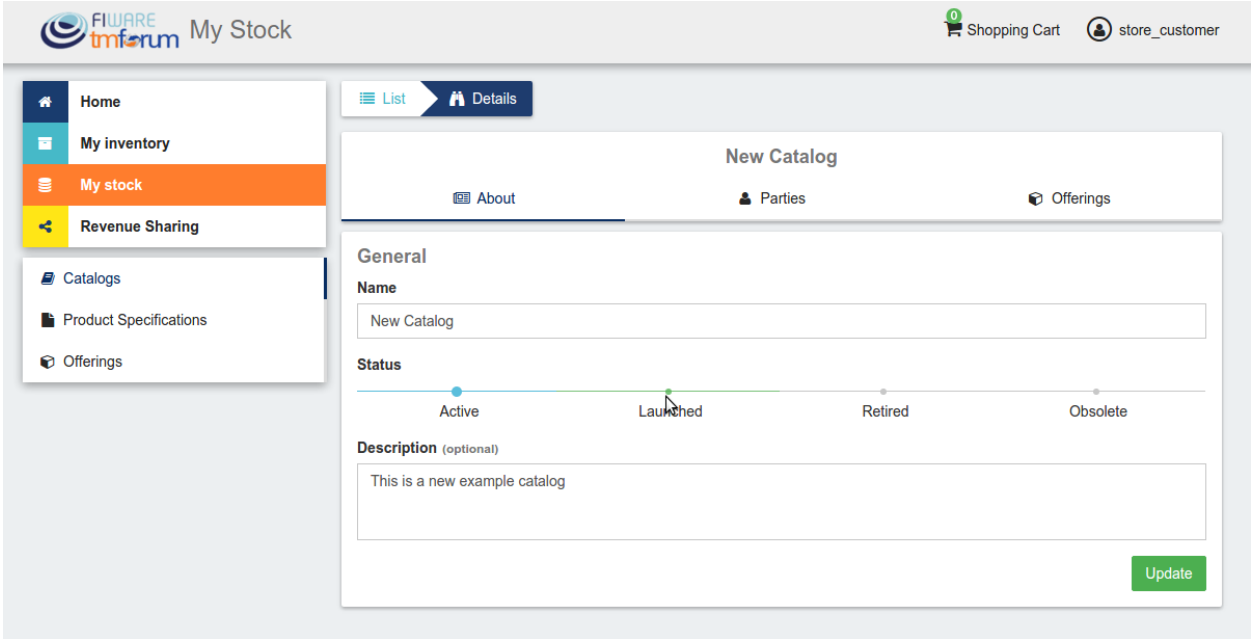

8. Provide a name and a description and click on *Next*. Then click on *Create*

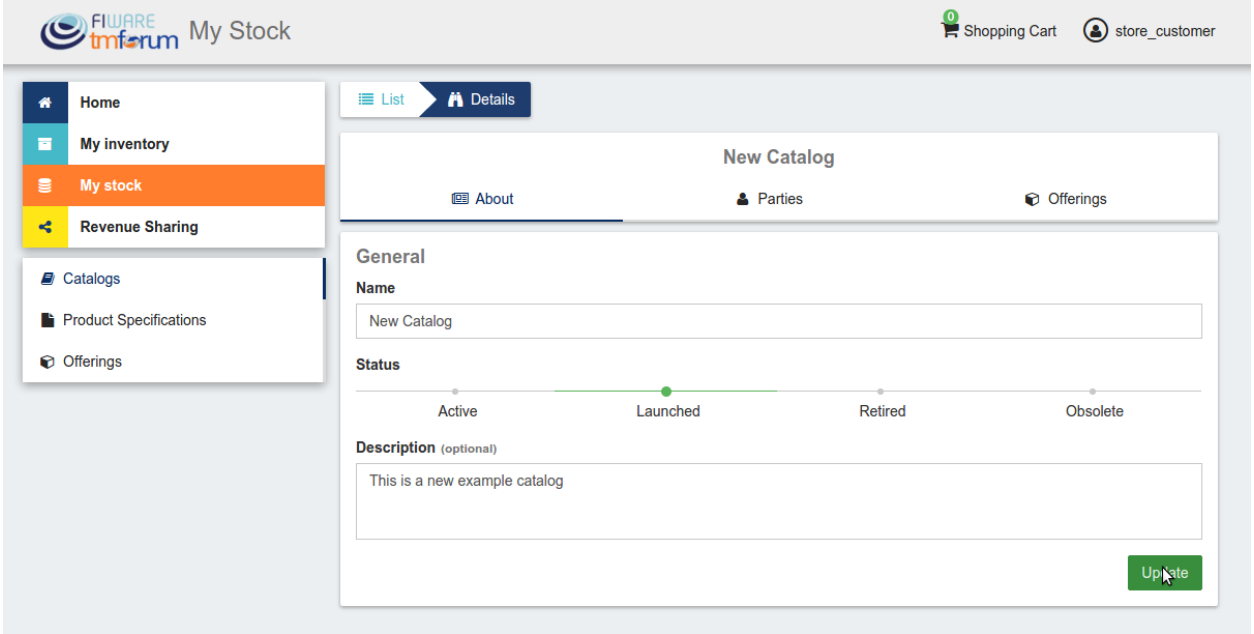

My Stock
Shopping Cart
store_customer
Home
My inventory
My stock
Revenue Sharing
Catalogs
Product Specifications
Offerings
List
New
New catalog
1 General
2 Finish
Step 1: General
Enter a name
New Catalog
Enter a description (optional)
This is a new example catalog
Next


My Stock
Shopping Cart
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

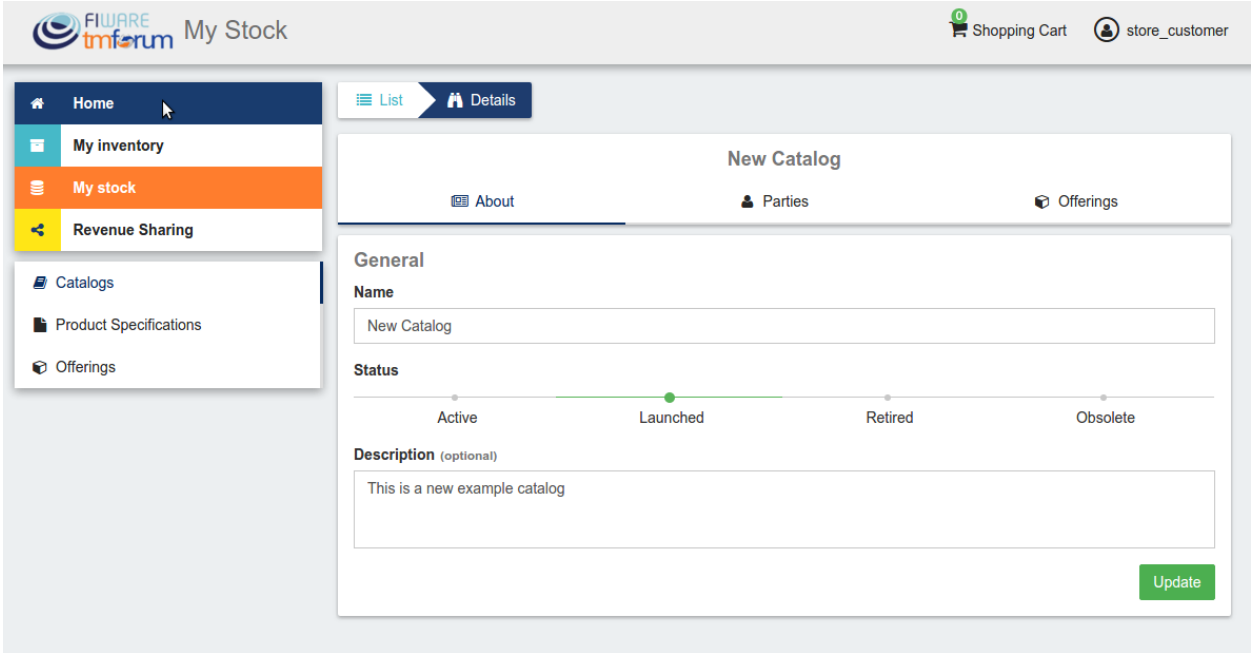


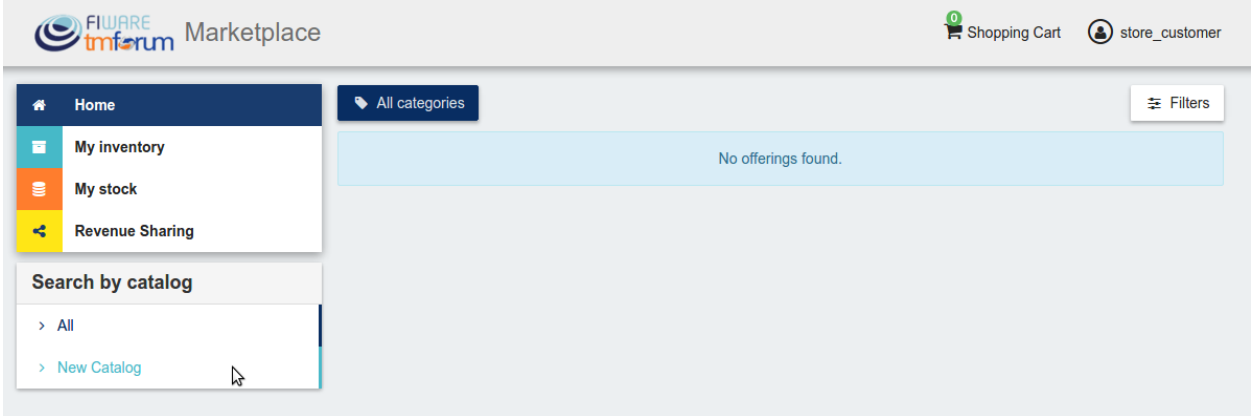

9. Click on *Launched*, and then click on *Update*






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






## 1.2 Configuration Guide

This guide covers the different configuration options that are available in order to setup a working Business API Ecosystem instance. The different Business API Ecosystem components can be configured using two different mecahnisms, settings files and environment variables.

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.

### 1.2.1 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:

```
SITE = 'http://localhost:8004/'
LOCAL_SITE = 'http://localhost:8006/'

CATALOG = 'http://localhost:8080/DSProductCatalog'
INVENTORY = 'http://localhost:8080/DSProductInventory'
ORDERING = 'http://localhost:8080/DSProductOrdering'
BILLING = 'http://localhost:8080/DSBillingManagement'
RSS = 'http://localhost:8080/DSRevenueSharing'
USAGE = 'http://localhost:8080/DSUsageManagement'
AUTHORIZE_SERVICE = 'http://localhost:8004/authorizeService/apiKeys'
```

These settings point to the different APIs accessed by the charging backend. In particular:

- SITE: External URL of the complete Business API Ecosystem using for Href creation
- LOCAL_SITE: URL where the Charging Backend is going to run
- 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.

These settings can be configured using the following environment variables:

```
BAE_SERVICE_HOST=http://proxy.docker:8004/
BAE_CB_LOCAL_SITE=http://charging.docker:8006/
BAE_CB_CATALOG=http://apis.docker:8080/DSProductCatalog
BAE_CB_INVENTORY=http://apis.docker:8080/DSProductInventory
BAE_CB_ORDERING=http://apis.docker:8080/DSProductOrdering
BAE_CB_BILLING=http://apis.docker:8080/DSBillingManagement
BAE_CB_RSS=http://rss.docker:8080/DSRevenueSharing
BAE_CB_USAGE=http://apis.docker:8080/DSUsageManagement
BAE_CB_AUTHORIZE_SERVICE=http://proxy.docker:8004/authorizeService/apiKeys
```

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": "djongo",
        "NAME": "wstore_db",
        "ENFORCE_SCHEMA": False,
        "CLIENT": {
            "host": "localhost",
            "port": 27017
            "username': "mongoadmin",
            "password': "mongopass"
        },
    }
}
```

This setting contains the following fields:

- ENGINE: Database engine, must be fixed to djongo
- NAME: Name of the database to be used
- **CLIENT: Configuration for connecting to MongoDB**
  - 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
  - username: 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

These settings can be configured using the environment with the following variables:

```
BAE_CB_MONGO_SERVER=mongo
BAE_CB_MONGO_PORT=27017
BAE_CB_MONGO_DB=charging_db
BAE_CB_MONGO_USER=user
BAE_CB_MONGO_PASS=passwd
```

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

These parameters can be configured with the environment using:

```
BAE_LP_OAUTH2_ADMIN_ROLE=admin
BAE_LP_OAUTH2_SELLER_ROLE=seller
BAE_LP_OAUTH2_CUSTOMER_ROLE=customer
```

The charging backend is the component in charge of maintaining the supported currencies and the timeframe of the different periods using in recurring pricing models. To configure both, the following settings are used:

```
CURRENCY_CODES = [
    ('EUR', 'Euro'),
    ('AUD', 'Australia Dollar'),
    ...
]
CHARGE_PERIODS = {
    'daily': 1,  # One day
    'weekly': 7,  # One week
    'monthly': 30,  # One month
    ...
}
```

- CURRENCY_CODES: Includes the list of currencies supported by the system as a tuple of currency code and currency name.
- CHARGE_PERIODS: Includes the list of supported periods for recurring models, specifing the time (in days) between periodic charges

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

These settings can be configured with the environment using:

```
BAE_CB_EMAIL=charging@email.com
BAE_CB_EMAIL_USER=user
BAE_CB_EMAIL_PASS=pass
BAE_CB_EMAIL_SMTP_SERVER=smtp.server.com
BAE_CB_EMAIL_SMTP_PORT=587
```

**Note:** The email configuration in 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
```

These 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

In addition, these settings can be configured using the following environment variables:

```
BAE_CB_PAYMENT_METHOD=paypal
BAE_CB_PAYPAL_CLIENT_ID=client_id
BAE_CB_PAYPAL_CLIENT_SECRET=client_secret
```

The charging backend component can be configured to expect or not the user access token to be propagated from the business logic proxy component, depending on the use case and the expected plugins to be installed. This can be configured with the following setting:

```
PROPAGATE_TOKEN = True
```

This setting can be also configured using the environment as follows:

```
export BAE_CB_PROPAGATE_TOKEN=true
```

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
```

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']),
    ('0 4 * * *', 'django.core.management.call_command', ['resend_upgrade']
]
```

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

```
$ python3 manage.py crontab add
```

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

```
$ python3 manage.py crontab show

$ python3 manage.py crontab remove
```

### 1.2.2 Configuring the Logic Proxy

Configuration of the Logic Proxy is located at *config.js* and can be provided in two different ways: providing the values in the file or using the defined environment variables. Note that the environment variables override the values in *config.js*.

The first setting to be configured is the port and host where the proxy is going to run, these settings are located in *config.js*

```
config.port = 80;
config.host = 'localhost';
```

In addition, the environment variables *BAE_LP_PORT* and *BAE_LP_HOST* can be used to override those values.

```
export BAE_LP_PORT=80
export BAE_LP_HOST=localhost
```

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*).

In order to provide the HTTPS configuration using the environment, the following variables has been defined.

```
export BAE_LP_HTTPS_ENABLED=true
export BAE_LP_HTTPS_CERT=cert/cert.crt
```

(continues on next page)

(continued from previous page)

```
export BAE_LP_HTTPS_CA=cert/key.key
export BAE_LP_HTTPS_KEY=cert/ca.crt
export BAE_LP_HTTPS_PORT=443
```

The logic proxy supports the BAE to be deployed behind a proxy (or NGINX, Apache, etc) not sending X-Forwarding headers. In this regard, the following setting is used in order to provide information about the actual endpoint which is used to access to the Business API Ecosystem:

```
config.proxy = {
    enabled: true,
    host: 'store.lab.fiware.org',
    secured: true,
    port: 443
};
```

Which can be also configured using the *BAE_SERVICE_HOST* environment variable.

```
export BAE_SERVICE_HOST=https://store.lab.fiware.org/
```

Then, it is possible to modify some of the URLs of the system. In particular, 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';
```

In addition, it is possible to configure the theme to be used by providing its name. Details about the configuration of Themes are provided in the *Configuring Themes* section:

```
config.theme = '';
```

The theme can be configured using the *BAE_LP_THEME* variable.

```
export BAE_LP_THEME=fiwaretheme
```

The BAE supports multiple external IDPs to be configured in order to allow organizations to login using their own IDP, when registered in a trust provider like iShare. To enable such feature the following setting needs to be configured:

```
config.extLogin = true;
```

This setting can be also configured using the environment as follows:

```
export BAE_LP_EXT_LOGIN=true
```

In addition, it is possible to configure whether the proxy component should propagate user access token to the backend components (charging backend, RSS and APIs), depending on the use case and the plugins installed. To configure such setting, the following is used:

```
config.propagateToken = true;
```

That can be configured using the environment as follows:

```
export BAE_LP_PROPAGATE_TOKEN=true
```

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*).

In addition, the database connection can be configured with the environment as following:

```
export BAE_LP_MONGO_USER=user
export BAE_LP_MONGO_PASS=pass
export BAE_LP_MONGO_SERVER=localhost
export BAE_LP_MONGO_PORT=27017
export BAE_LP_MONGO_DB=belp
```

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': 'DSProductCatalog',
        'host': 'localhost'
        'port': '8080',
        'appSsl': false
    },
    'ordering': {
        'path': 'DSProductOrdering',
        '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

---

Each of the different APIs can be configured with environment variables with the following pattern:

```
export BAE_LP_ENDPOINT_CATALOG_PATH=DSProductCatalog
export BAE_LP_ENDPOINT_CATALOG_PORT=8080
export BAE_LP_ENDPOINT_CATALOG_HOST=localhost
export BAE_LP_ENDPOINT_CATALOG_SECURED=false
```

The Business API Ecosystem uses an indexes system managed by the Logic Proxy in order to perform queries, searches, and paging the results. Starting in version 7.6.0 it is possible to use elasticsearch for the indexing rather than using the local file system. The indexing system is configured with the following settings.

```
config.indexes = {
    'engine': 'elasticsearch', // local or elasticsearch
    'elasticHost': 'elastic.docker:9200'
    'apiVersion': '7.5'
};
```

The *engine* setting can be used to chose between *local* indexes and *elasticsearch* indexes. If the later is chosen the URL of elasticsearch is provided with *elasticHost*.

These settings can be configured using the environment as follows:

```
export BAE_LP_INDEX_ENGINE=elasticsearch
export BAE_LP_INDEX_URL=elasticsearch:9200
export BAE_LP_INDEX_API_VERSION=7
```

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. They can be configured with environment variables with *BAE_LP_REVENUE_MODEL* and *BAE_LP_USAGE_CHART*

### Identity Management

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 includes the IDP and login configuration. The BAE supports multiple IPD implementations. In particular:

- FIWARE Keyrock
- Keycloak
- GitHub
- FIWARE Keyrock + iShare protocol
- OIDC with discovery server

To configure the IPD integration thw setting *oauth2* is used. The following example shows an example configuration using Keyrock

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

(continues on next page)

(continued from previous page)

```
        'admin': 'admin',
        'customer': 'customer',
        'seller': 'seller',
        'orgAdmin': 'orgAdmin'
    }
};
```

In this settings, the value of *provider* is used to configure the IDP type. Then, 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*).

In addition, the different roles allow to specify what users are admins of the system (*Admin*), what users can create products and offerings (*Seller*), and what users are admins of a particular organization, enabling to manage its information (*orgAdmin*). Note that while *admin* and *seller* roles are granted directly to the users in the Business API Ecosystem application, the *orgAdmin* role has to be granted to users within IdM organizations.

---

**Note:** Admin, Seller, and orgAdmin roles are configured in the Proxy settings, so any name can be chosen for them in the IDM

---

The OAuth2 settings can be configured using the environment as follows:

```
export BAE_LP_OAUTH2_PROVIDER=fiware
export BAE_LP_OAUTH2_SERVER=https://account.lab.fiware.org
export BAE_LP_OAUTH2_CLIENT_ID=client_id
export BAE_LP_OAUTH2_CLIENT_SECRET=client_secret
export BAE_LP_OAUTH2_CALLBACK=http://<proxy_host>:<proxy_port>/auth/fiware/callback
export BAE_LP_OAUTH2_ADMIN_ROLE=admin
export BAE_LP_OAUTH2_SELLER_ROLE=seller
export BAE_LP_OAUTH2_ORG_ADMIN_ROLE=orgAdmin
```

For Keycloak provider some extra settings need to be provided. The following is an example of a Keycloak configuration:

```
config.oauth2 = {
    provider: 'keycloak',
    server: 'http://keycloak.docker:8080',
    clientID: 'bae',
    clientSecret: 'df68d1b9-f85f-4b5e-807c-c8be3ba27388',
    callbackURL: 'http://proxy.docker:8004/auth/keycloak/callback',
    realm: 'bae',
    roles: {
        admin: 'admin',
        customer: 'customer',
        seller: 'seller',
        orgAdmin: 'manager'
    }
}
```

It can be seen that the *provider* setting is set to keycloak and that the *realm* setting is used to specify the Keycloak realm. Such setting can be configured using the environment using:

```
export BAE_LP_OIDC_REALM=bae
```

When using the iShare protocol, the configuration requires the certificate issues by iShare to be provided in order to generate and sign the JWT used in such a protocol. Such info can be provided by the settings *tokenCrt* and *tokenKey* or via enviroment with:

```
export BAE_LP_OIDC_TOKEN_KEY=...
export BAE_LP_OIDC_TOKEN_CRT=...
```

Finally, if the OIDC protocol is used the following settings need to be configured:

- oidcScopes: Scopes requested in the OIDC request
- oidcDiscoveryURI: Discovery endpoint for the OIDC protocol
- oidcTokenEndpointAuthMethod: Method used for retriving the access token in the OIDC server

Such settings can be configured with the environ using:

```
BAE_LP_OIDC_SCOPES
BAE_LP_OIDC_DISCOVERY_URI
BAE_LP_OIDC_TOKEN_AUTH_METHOD
```

### 1.2.3 Configuring the TMF APIs

When the TMF APIs are deployed from sources, the connection to the MySQL database is configured during the installation process setting up the jdbc connection as described in the *Installation and Administration* guide.

On the other hand, the Docker image biz-ecosystem-apis, which is used to the deploy TMF APIs using Docker, uses two environment variables for configuring such connection.

```
MYSQL_ROOT_PASSWORD=my-secret-pw
MYSQL_HOST=mysql
```

Finally, the TMF APIs can optionally use a configuration file called *settings.properties* which is located by default at */etc/default/apis*. This file include a setting *server* which allows to provide the URL used to access to the Business API Ecosystem and, in particular, by the APIs in order to generate *hrefs* with the proper reference.

```
server=https://store.lab.fiware.org/
```

This setting can also be configured using the environment variable *BAE_SERVICE_HOST*

```
export BAE_SERVICE_HOST=https://store.lab.fiware.org/
```

### 1.2.4 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

In addition, database settings can be configured using the environment. In particular, using the following variables:

```
export BAE_RSS_DATABASE_URL=jdbc:mysql://mysql:3306/RSS
export BAE_RSS_DATABASE_USERNAME=root
export BAE_RSS_DATABASE_PASSWORD=my-secret-pw
export BAE_RSS_DATABASE_DRIVERCLASSNAME=com.mysql.jdbc.Driver
```

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

```
config.grantedRole=admin
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

Those settings can also be configured using the environment as

```
export BAE_RSS_OAUTH_CONFIG_GRANTEDROLE=admin
export BAE_RSS_OAUTH_CONFIG_SELLERROLE=Seller
export BAE_RSS_OAUTH_CONFIG_AGGREGATORROLE=Aggregator
```

### 1.2.5 Configuring Themes

The Business API Ecosystem provides a basic mechanism for the creation of themes intended to customize the web portal of the system. Themes include a set of files which can override any of the default portal files located in the *public/resources* or *views* directories of the logic proxy. To do that, themes map the directory structure and include files with the same name of the default ones to be overridden.

The Logic Proxy can include multiple themes which should be stored in the *themes* directory located at the root of the project.

To enable themes, the *config.theme* setting is provided within the *config.js* file of the Logic Proxy. Themes are enabled by providing the name of the theme directory in this setting.

```
config.theme = 'dark-theme';
```

**Note:** Setting *config.theme* to an empty string makes the Business API Ecosystem to use its default theme

To start using a theme the following command has to be executed:

```
$ node collect_static.js
```

This command merges the theme files and the default ones into a *static* directory used by the Logic Proxy to retrieve portal static files.

### 1.2.6 Enabling Production

The default installation of the Business API Ecosystem deploys its different components in *debug* mode. This is useful for development and testing but it is not adequate for production environments.

Enabling the production mode makes the different components to start caching requests and views and minimizing JavaScript files.

To enable the production mode, the first step is setting the environment variable *NODE_ENV* to *production* in the machine containing the Logic Proxy.

```
$ export NODE_ENV=production
```

Then, it is needed to collect static files in order to compress JavaScript files.

```
$ node collect_static.js
```

Finally, change the setting *DEBUG* of the Charging Backend to False.

```
DEBUG=False
```

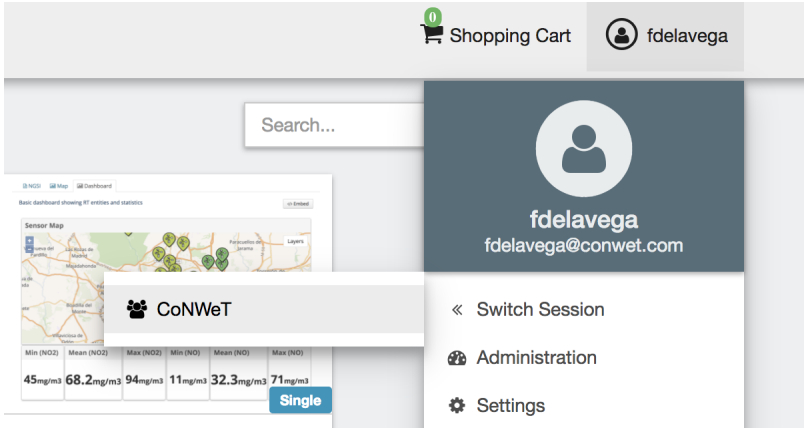
## 1.3 User Guide

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 particular user role are grouped together.

### 1.3.1 Using Organizations

The Business API Ecosystem supports organizations as defined by the FIWARE IdM. These organizations can use the system as if they were users, being possible to create organizations catalogs and offerings or acquire them.

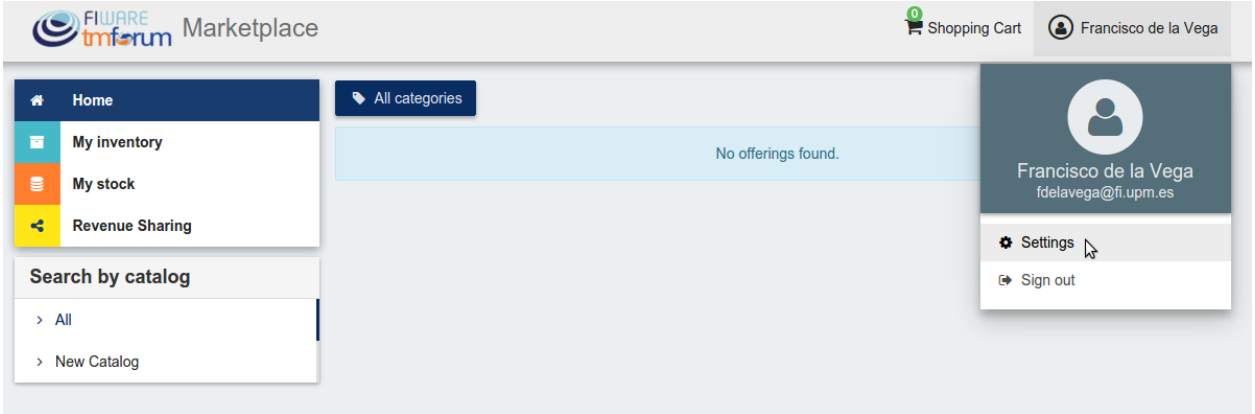
To use the platform on behalf an organization the user belongs, it is needed to change the platform context. To do that, it is used the *Switch Session* option of the user menu.



### 1.3.2 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.

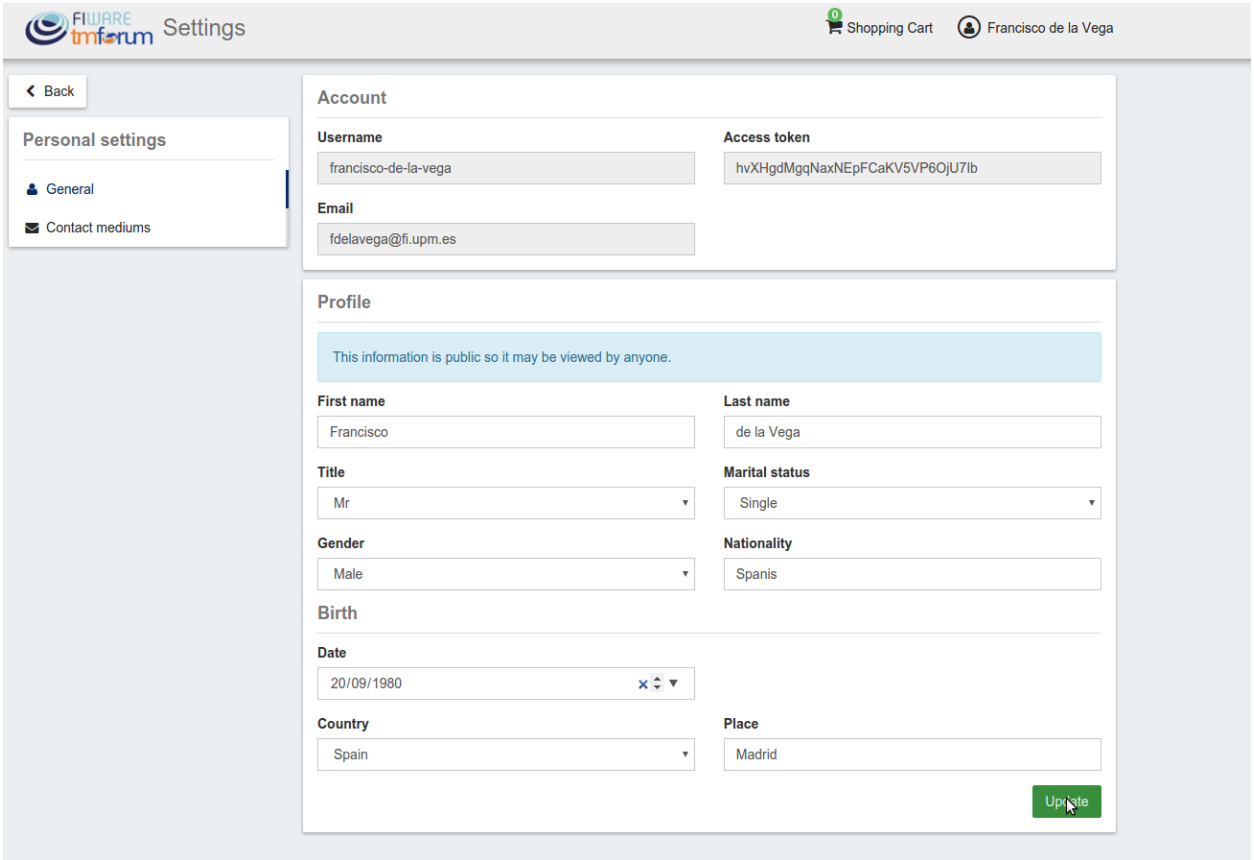
To configure the user profile, the first step is opening the user *Settings* located in the user menu.




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.

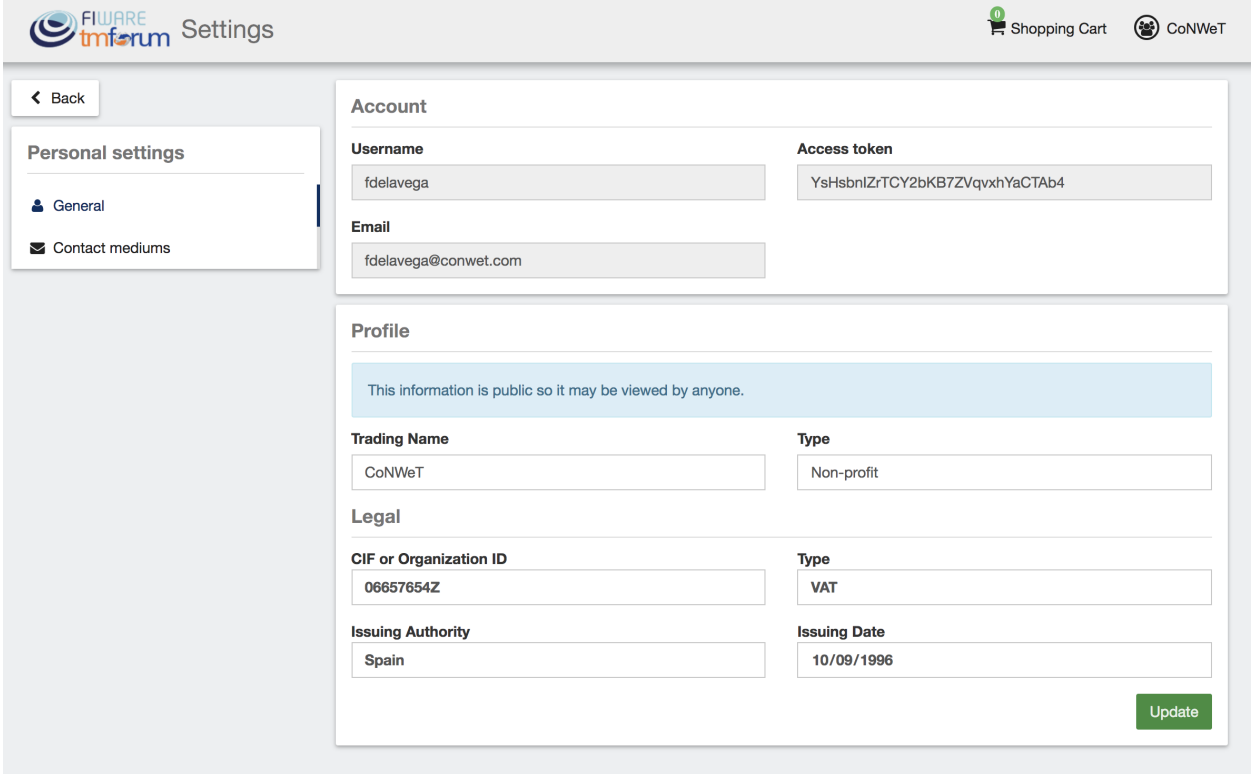
The profile to be updated depends on whether the user is acting on behalf an organization or himself. In both cases, to update the profile, fill in the required information and click on *Update*.

For users, personal information is provided.

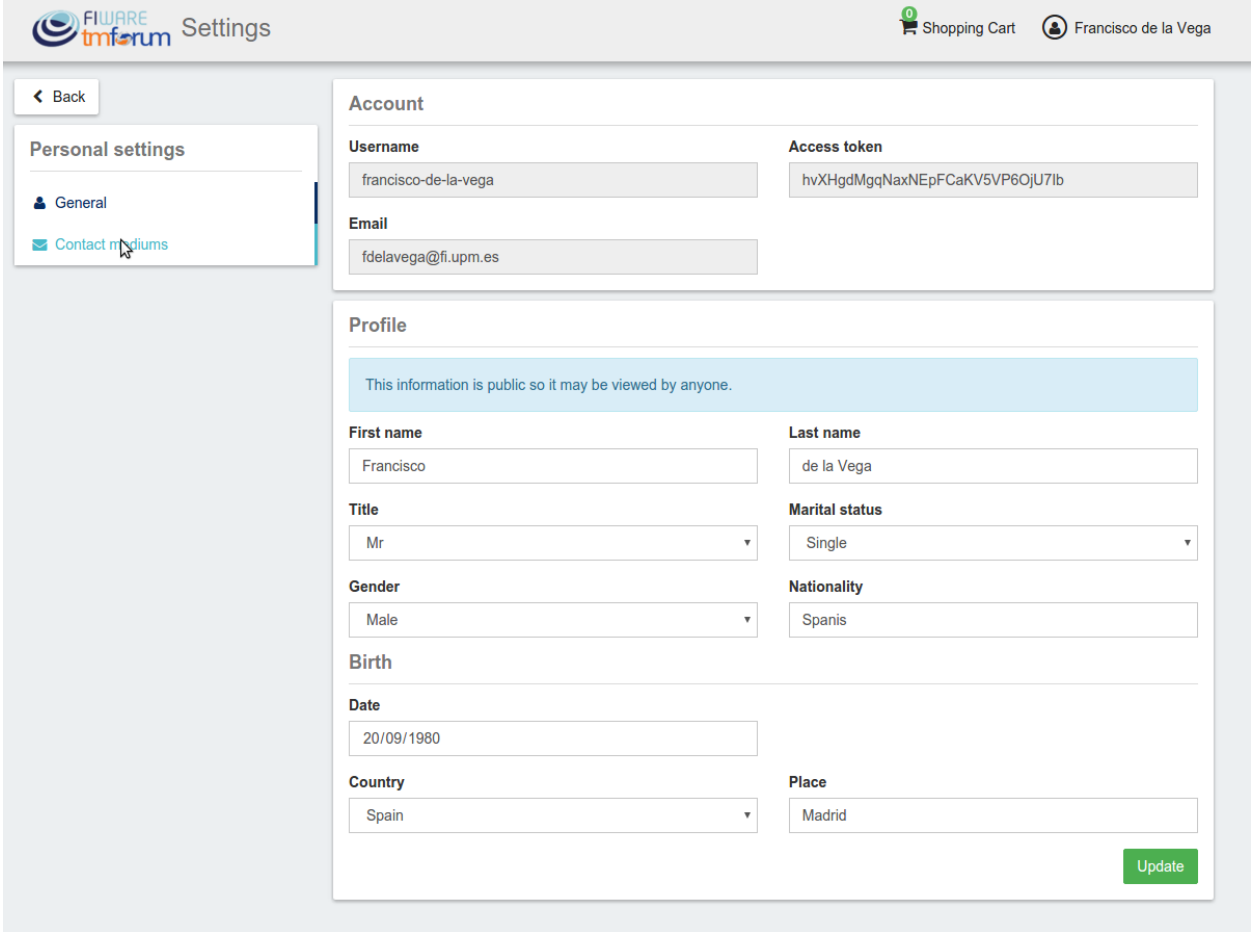


**Note:** Only the *First name* and *Last name* fields are mandatory

For organizations, general organization info is provided.

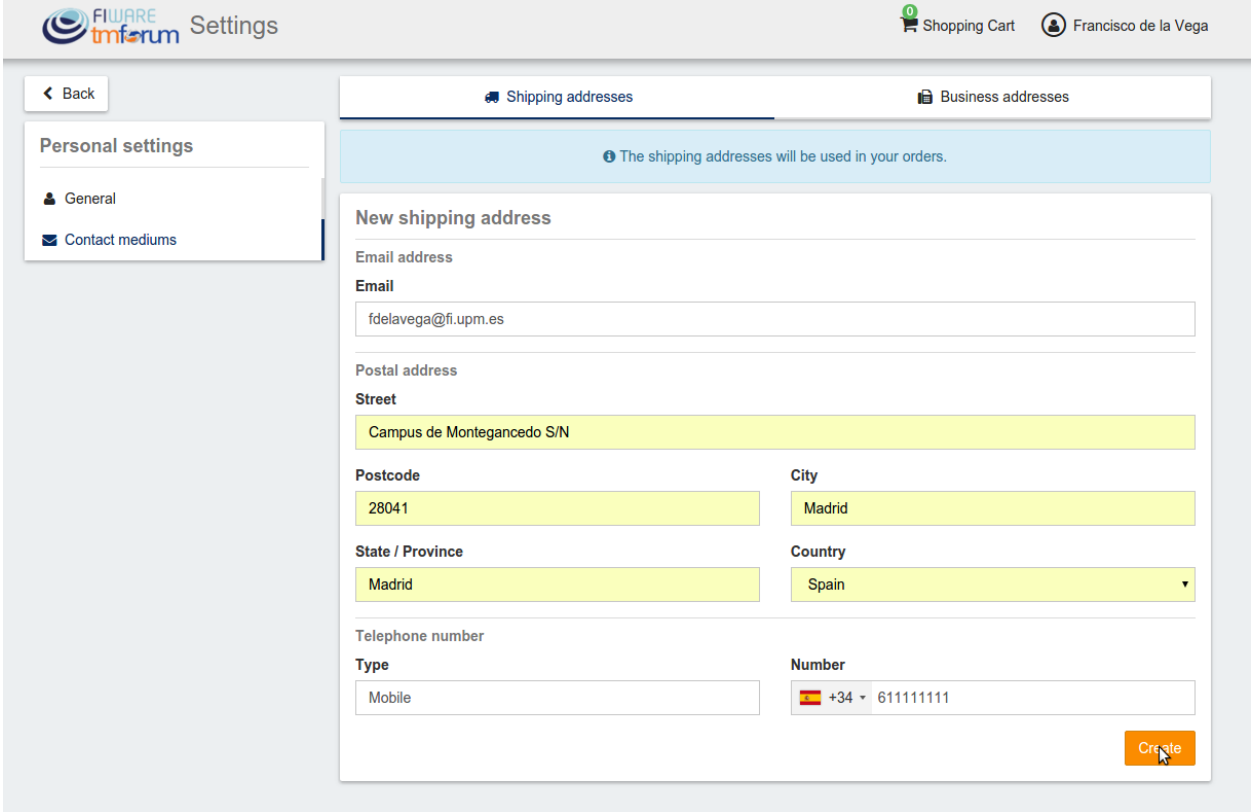



Once you have created your profile, you can include contact mediums by going to the *Contact mediums* section.

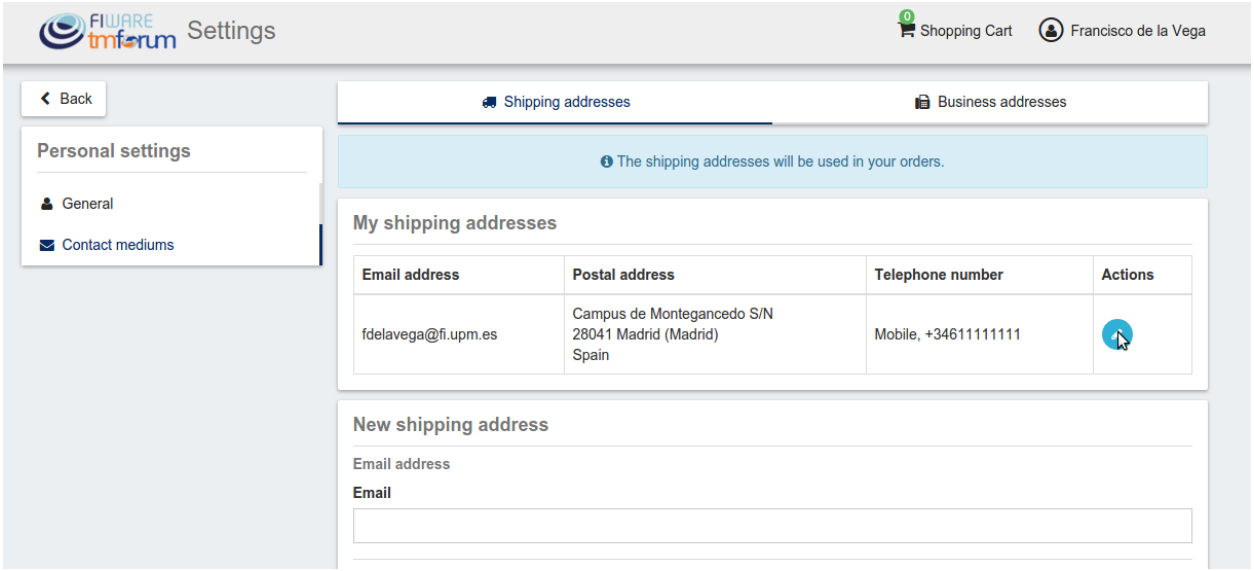


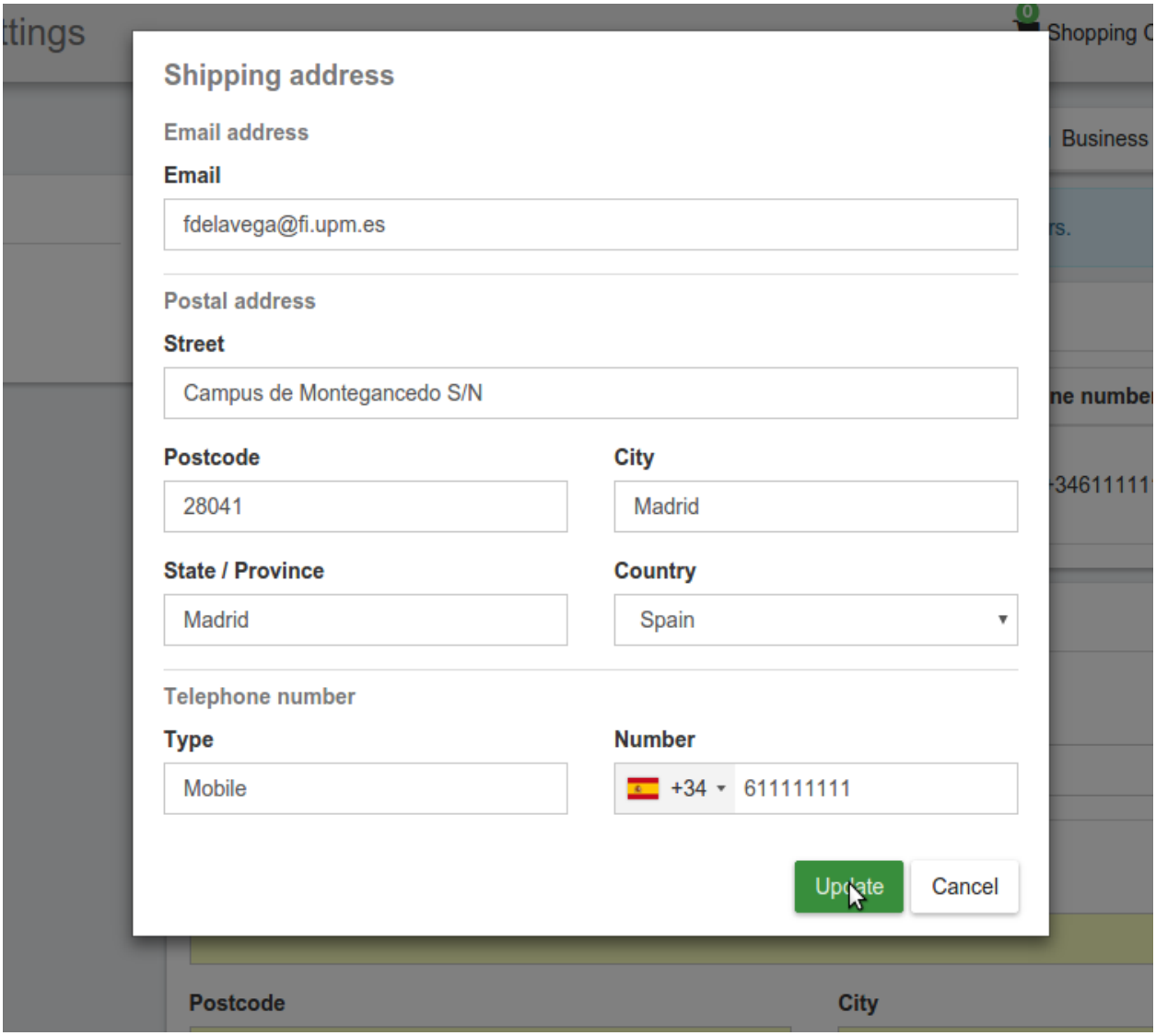

In the *Contact Medium* section, there are two different tabs. On the one hand, the *Shipping addresses* tab, where you can register the shipping addresses you will be able to use when creating orders and purchasing products.

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

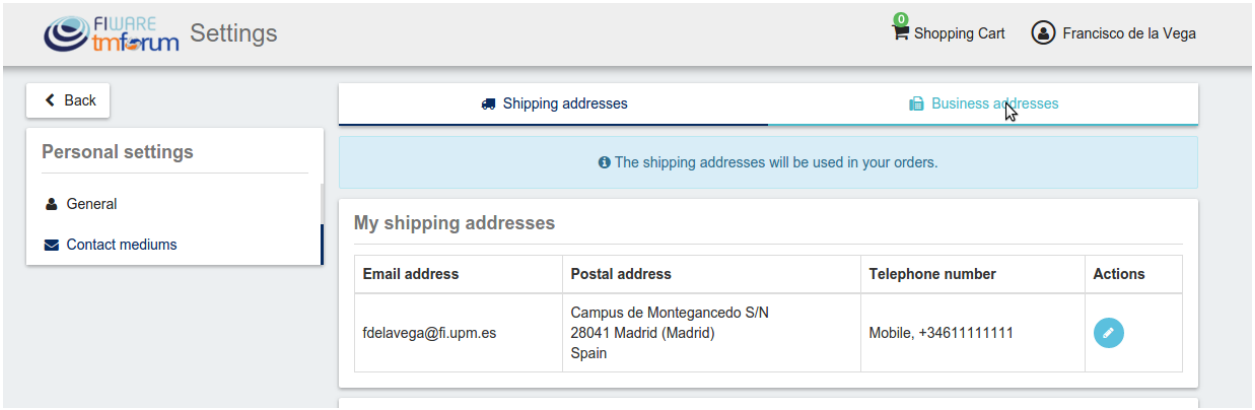



Once created, you can edit the address by clicking on the *Edit* button of the specific address, and changing the wanted fields.

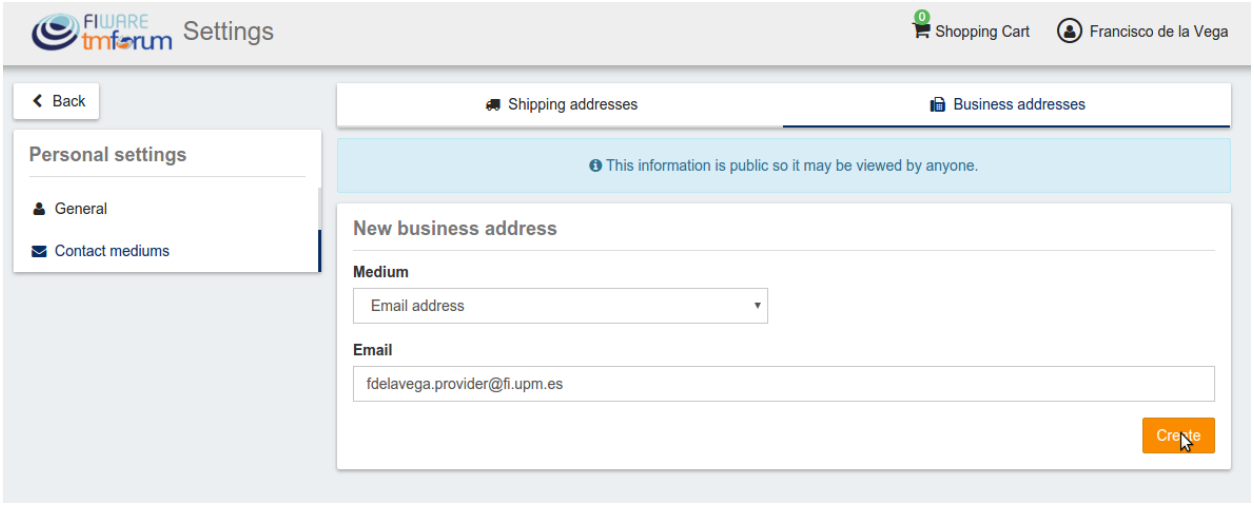


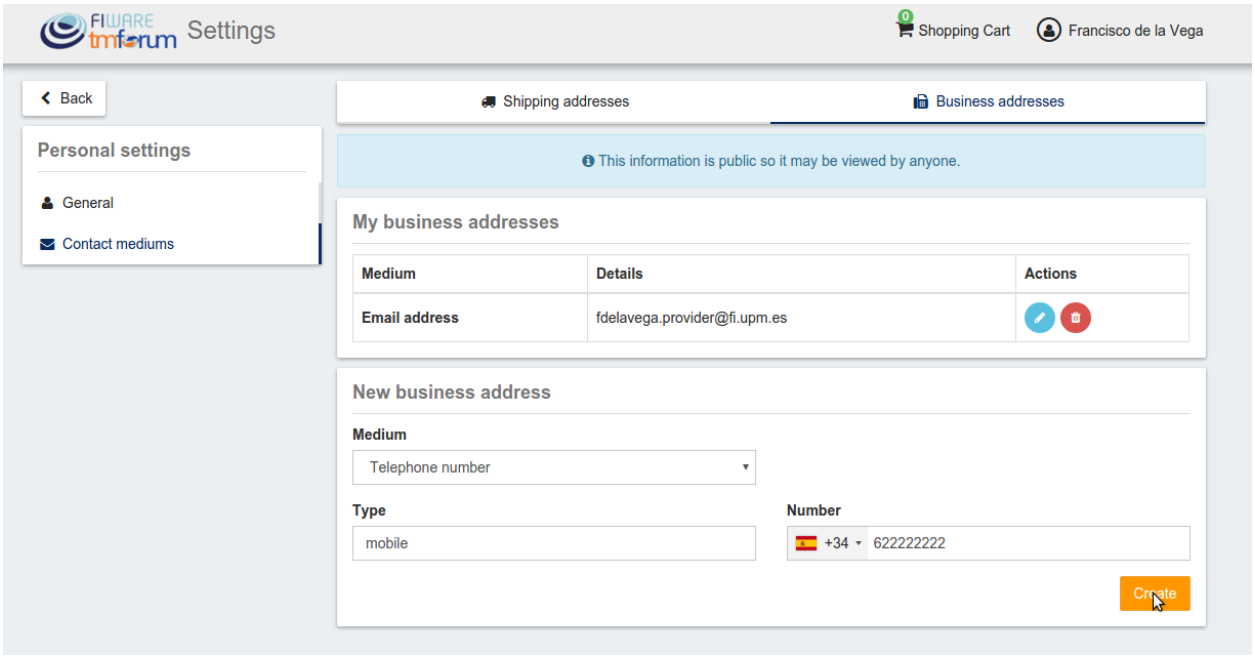


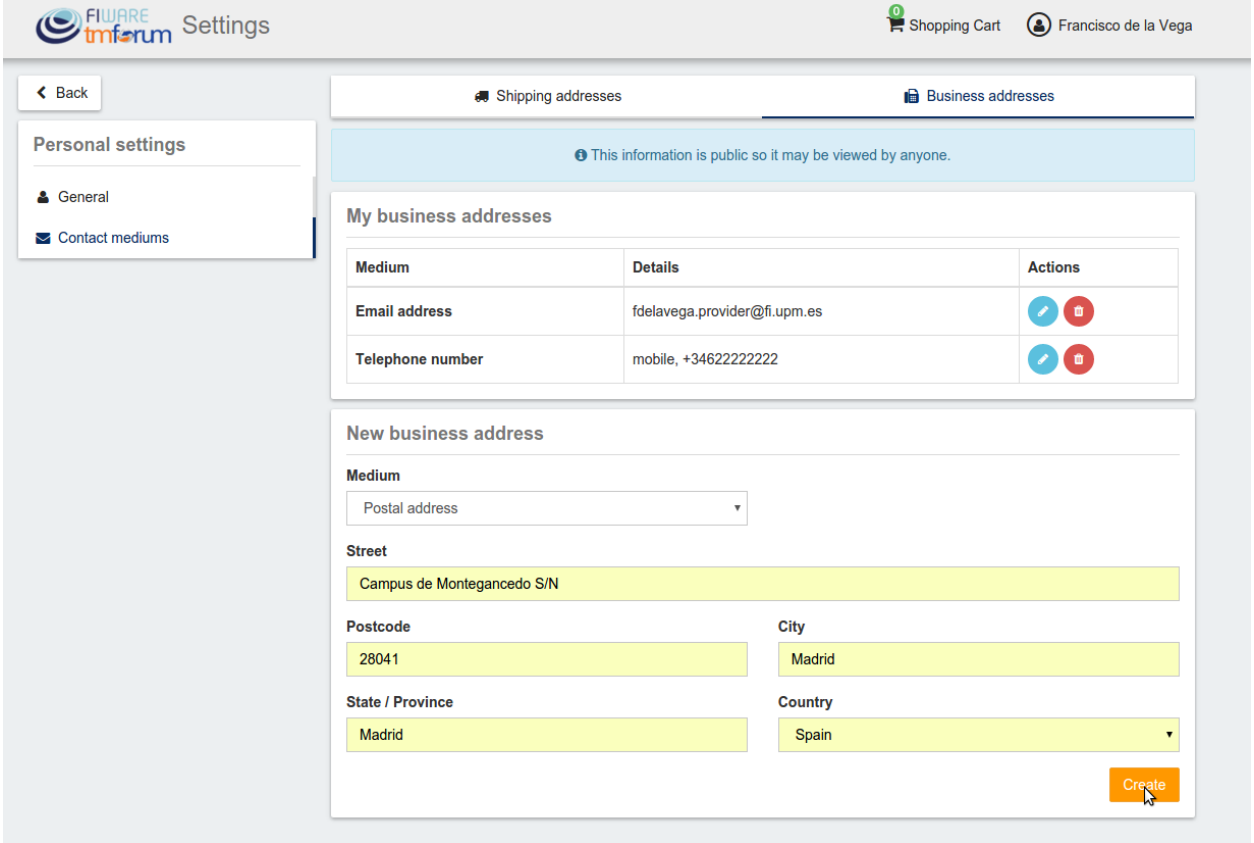

On the other hand, if you have the *Seller* role you can create *Business Addresses*, which can be used by your customers in order to allow them to contact you.



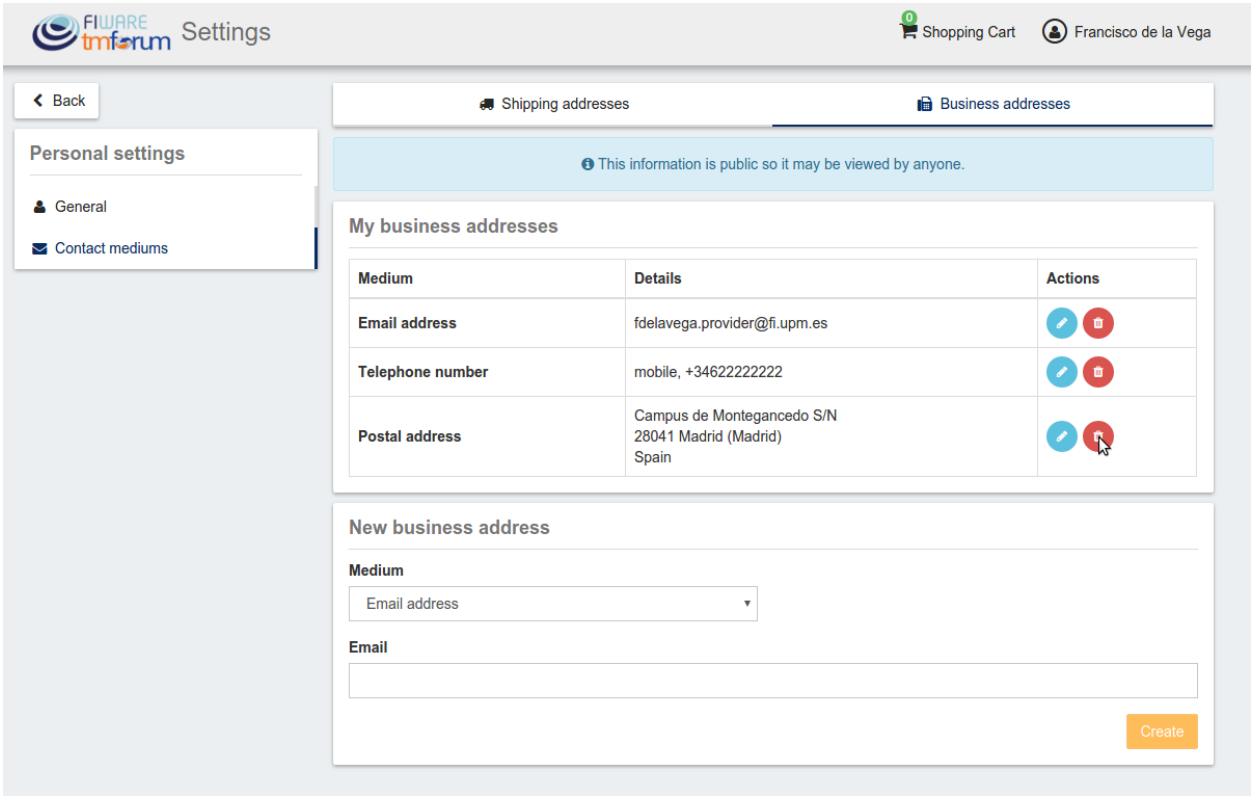

In the *Business Addresses* tab you can create, different kind of contact mediums, including emails, phones, and addresses. To create a contact medium, fill in the fields and click on *Create*


FIWARE tmforum Settings
Shopping Cart
Francisco de la Vega
Back
Shipping addresses
Business addresses
Personal settings
General
Contact mediums
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 tmforum Settings
Shopping Cart
Francisco de la Vega
Back
Shipping addresses
Business addresses
Personal settings
General
Contact mediums
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




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



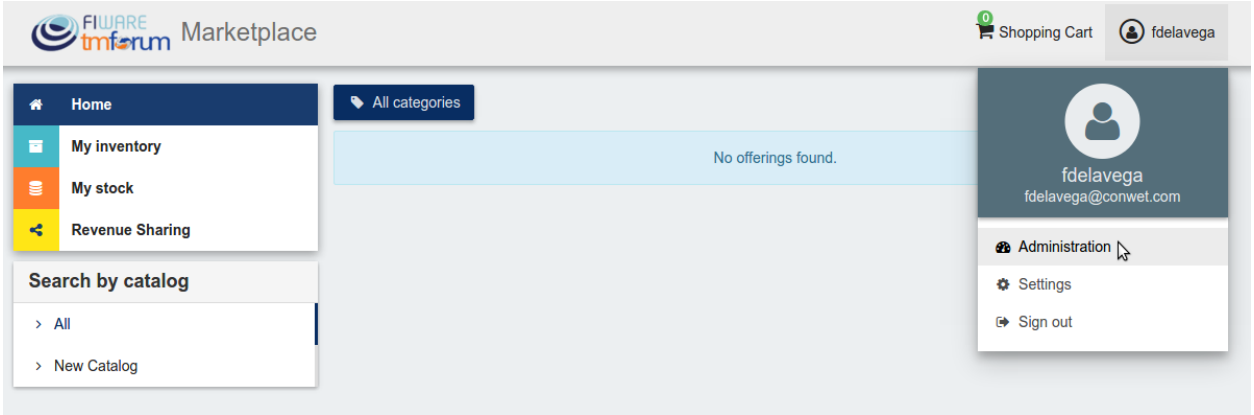
### 1.3.3 Admin

If the external IDPs feature is enabled, admins should login in the system with the local IDPs by directly accessing to the login URL in the browser:

```
https://[marketurl]/login
```

If the external IDP is disabled, the login button will use the local IDP.

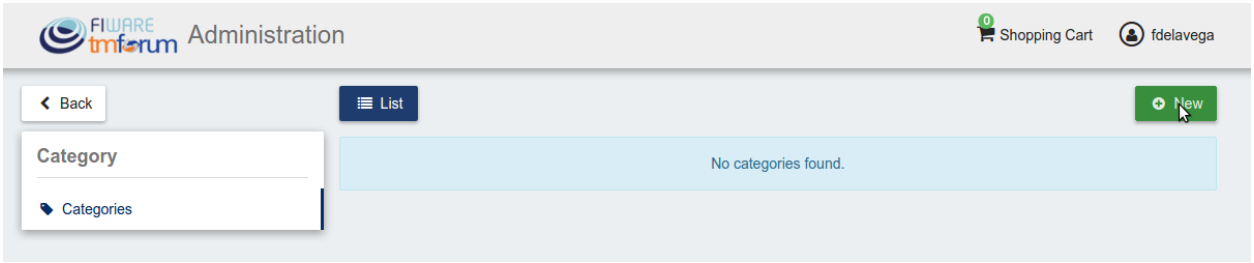
If the user of the Business API Ecosystem is an admin, he will be able to access the *Administration* section of the web portal. This section is located in the user menu.



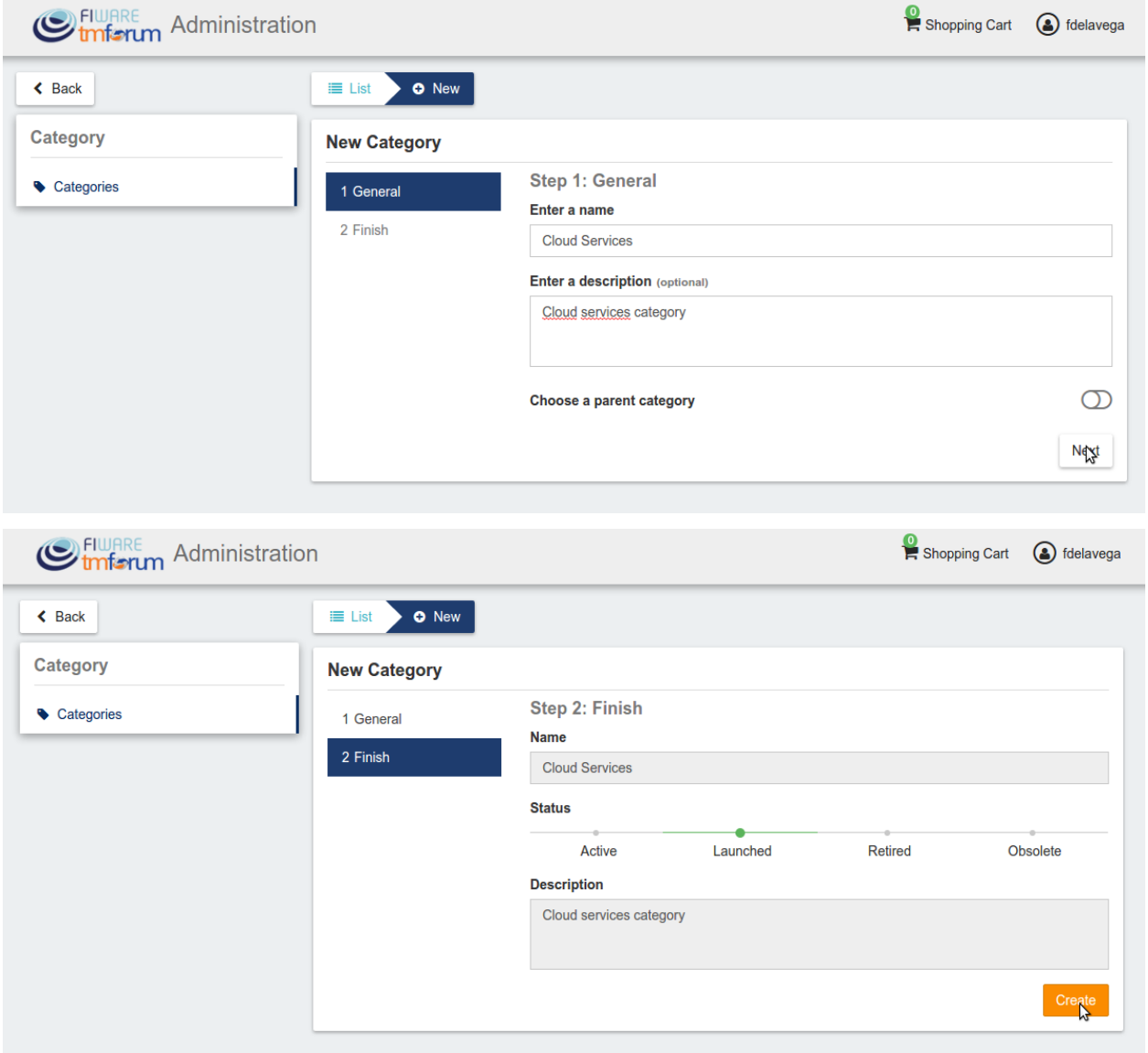

#### Manage Categories

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

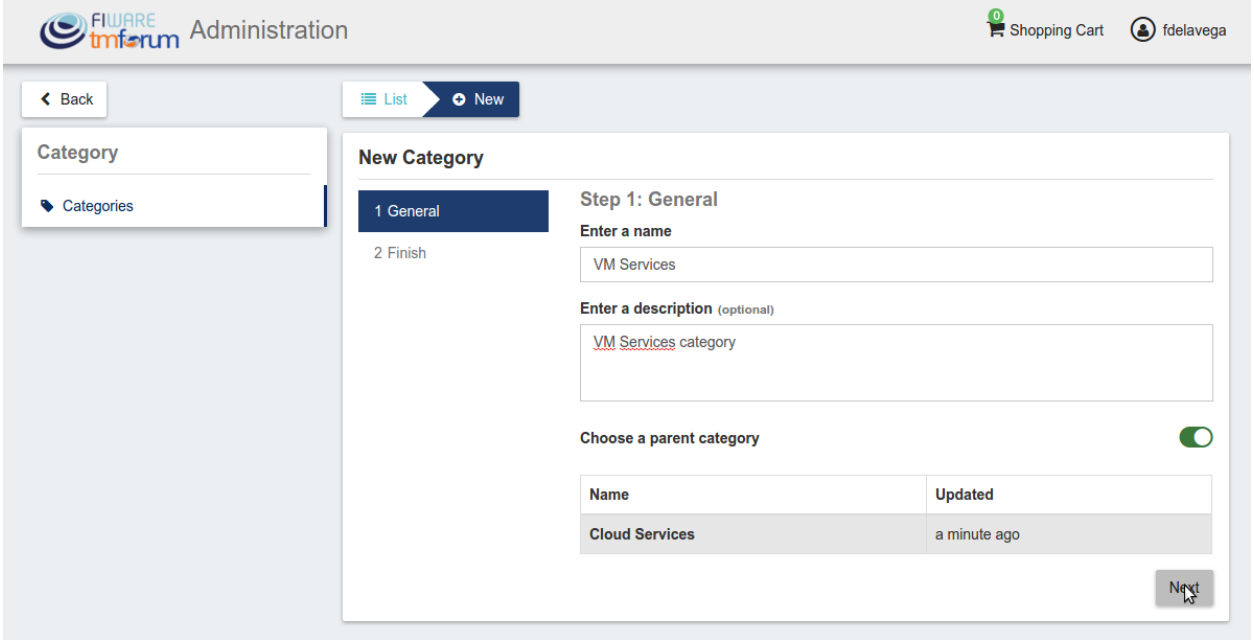
To create categories, go to the *Administration* section, and click on *New*



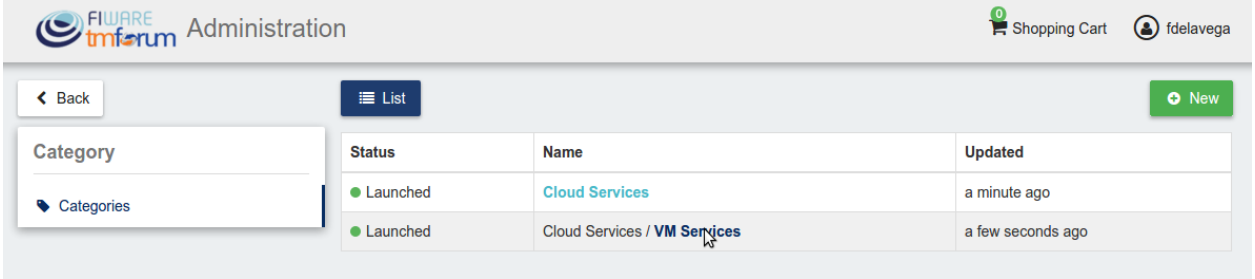

Then, provide a name and an optional description for the category. Once the information has been included, click on *Next*, and then on *Create*



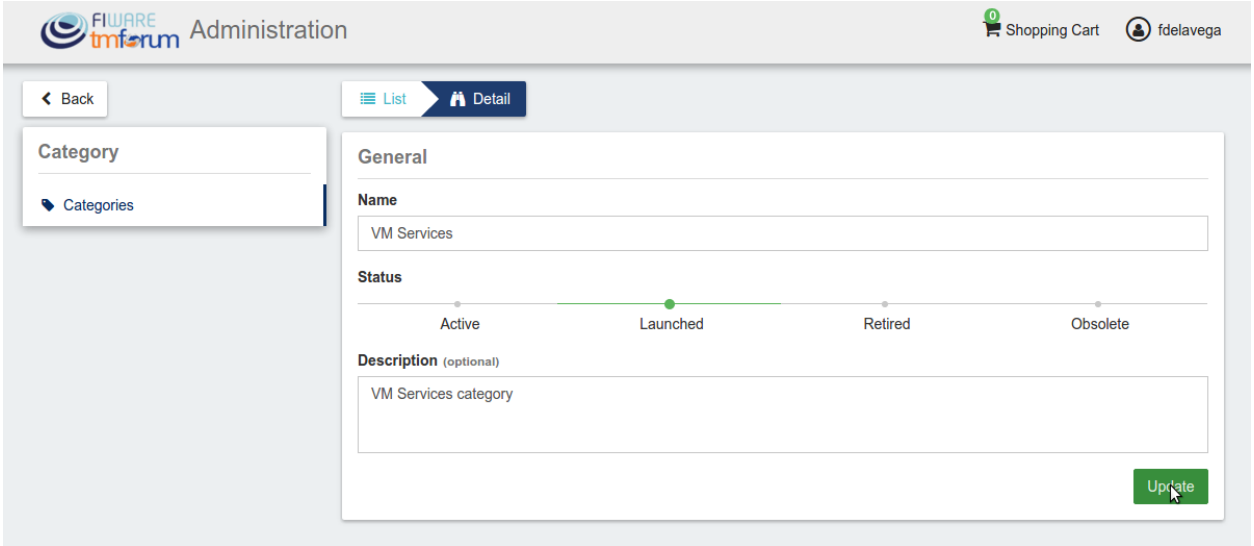

Categories in the Business API Ecosystem can be nested, so you can choose a parent category if you want while creating.




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



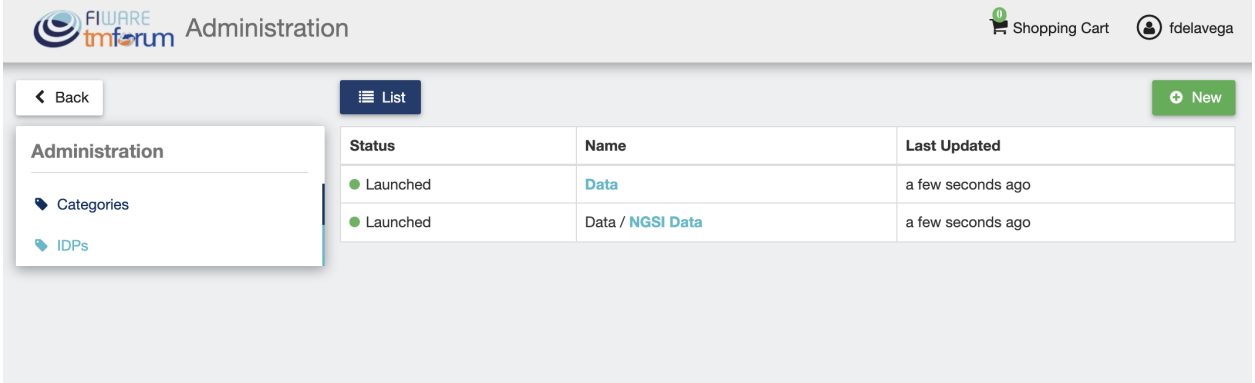

Then edit the corresponding fields and click on *Update*.

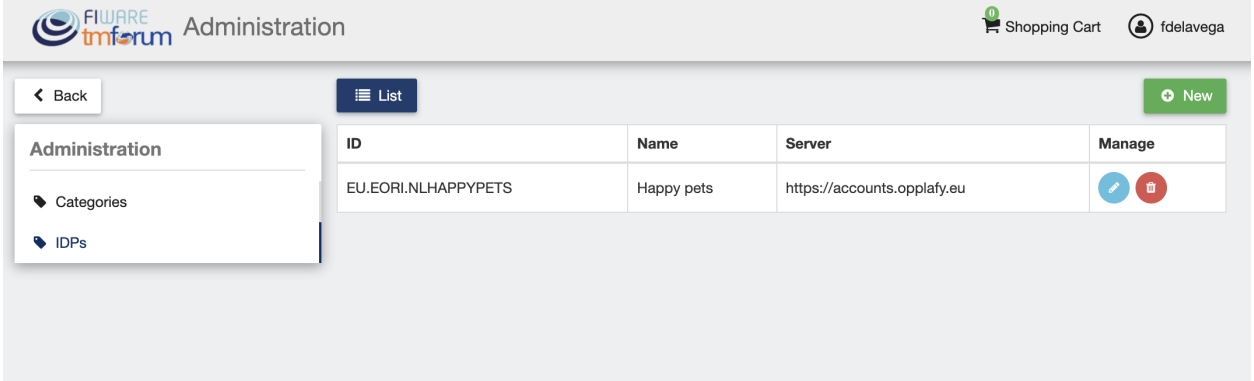


### Manage IDPs

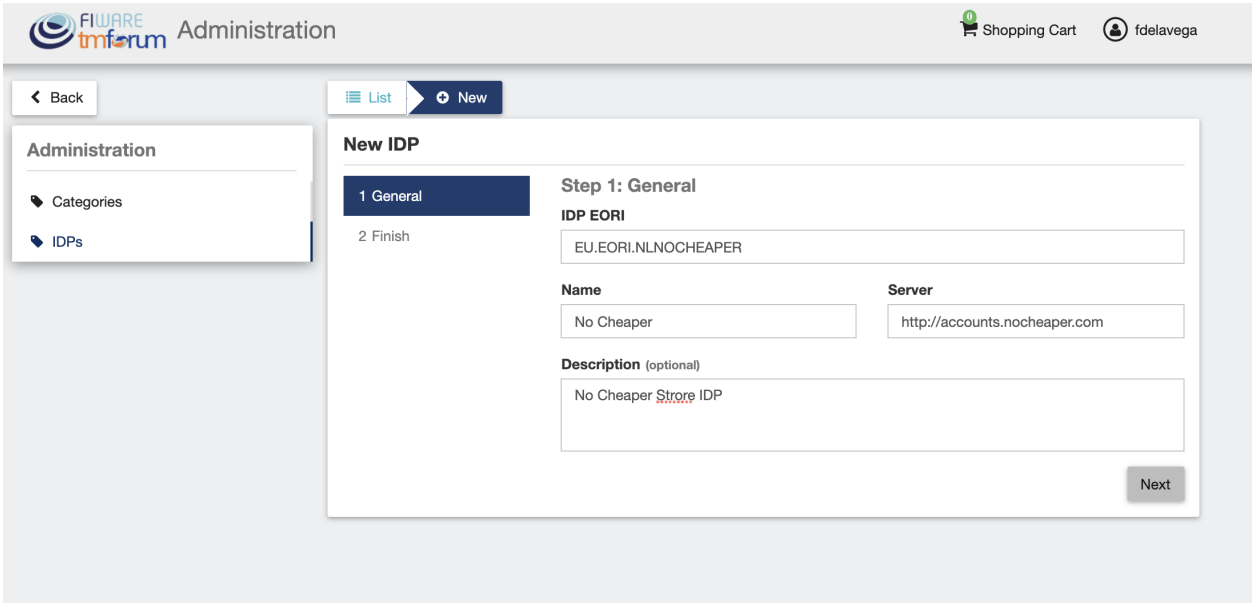
If the external IDPs option is enabled, *admins* are authorized to register them using the *Administration* section.

To list existing IDPs access to *IDPs*:



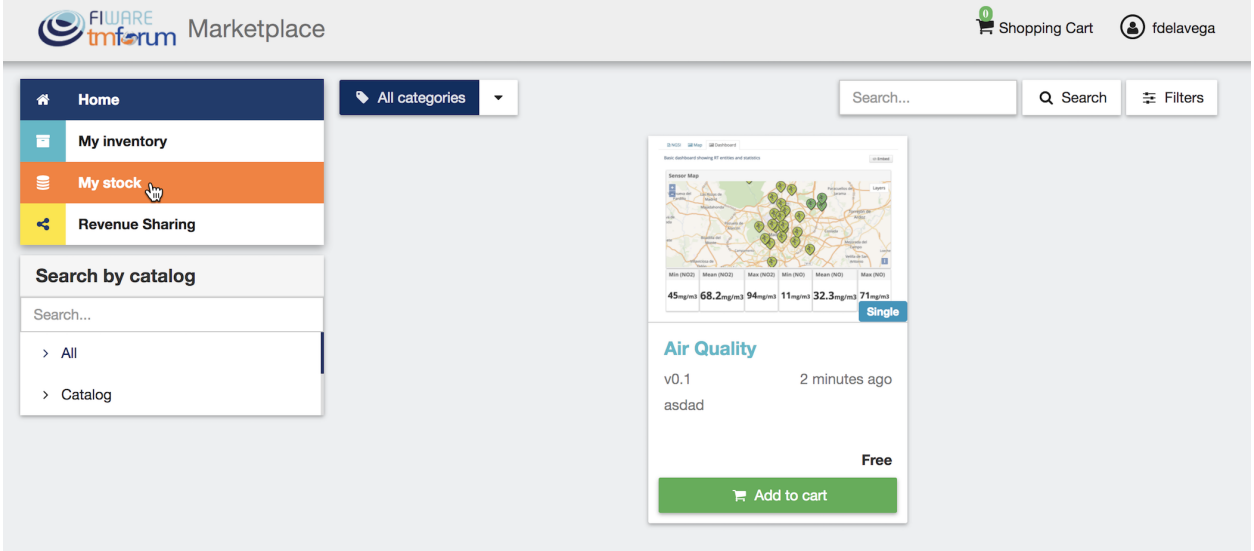




To register a new IDP click in *New*. In the displayed form, fill the *IDP EORI* with the EORI given to the IDP by the trust provider (i.e iShare). Provide a name and an optional description and fill *Server* with the URL of the IDP.



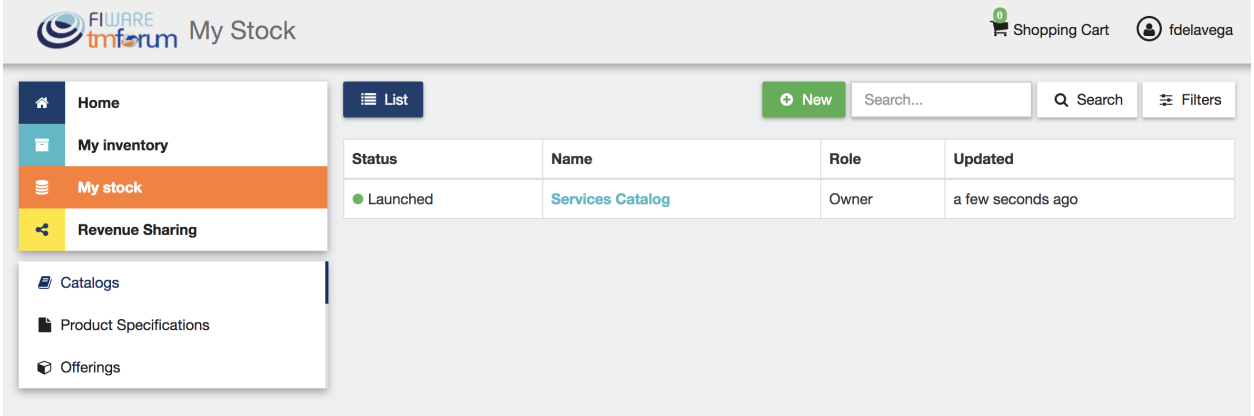
### 1.3.4 Seller

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.

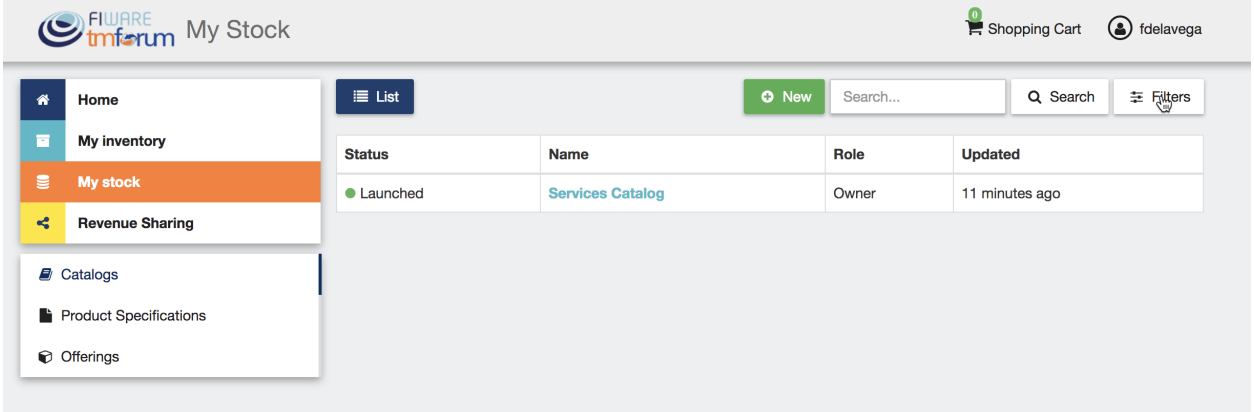


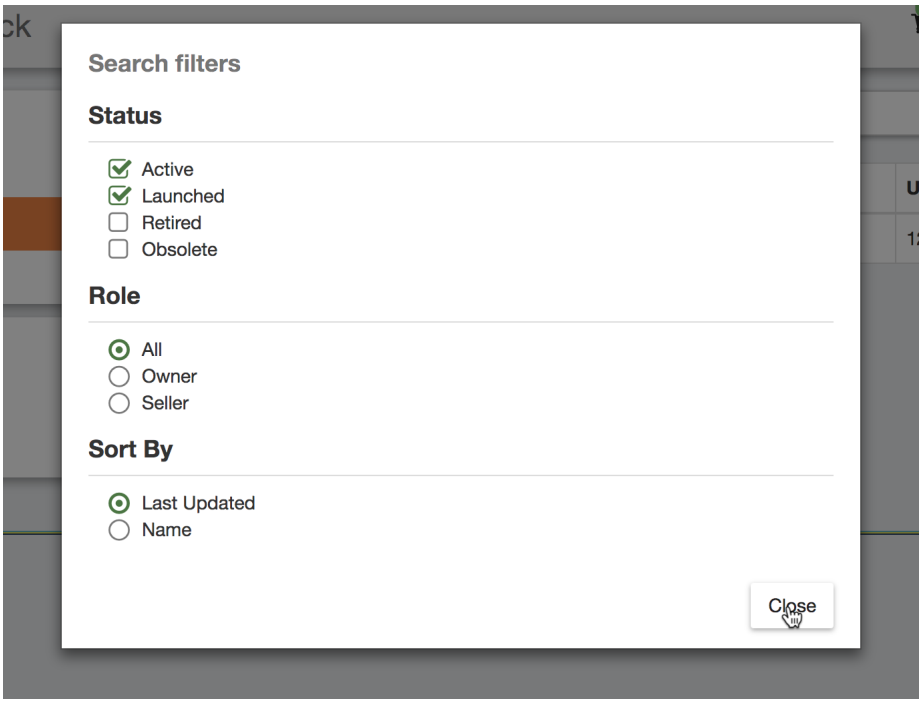

#### Manage Catalogs

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.

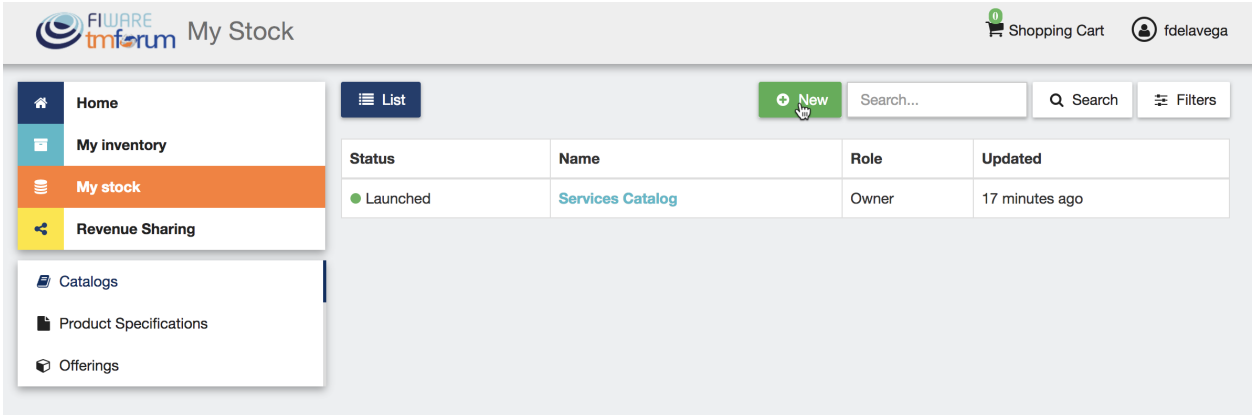



Additionally, it has been defined several mechanisms for searching and filtering the list of catalogs displayed. On the one hand, it is possible to search catalogs by keyword using the search input provided in the menu bar. On the other hand, it is possible to specify how catalog list should be sorted or filter the shown catalogs by status and the role you are playing. To do that, click on *Filters*, choose the required parameters, and click on *Close*.

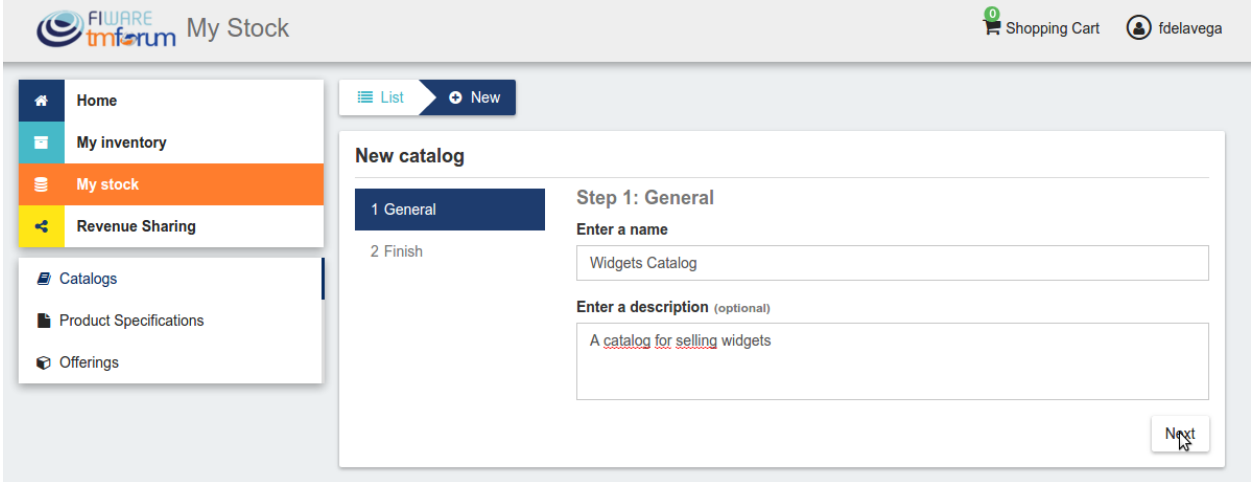


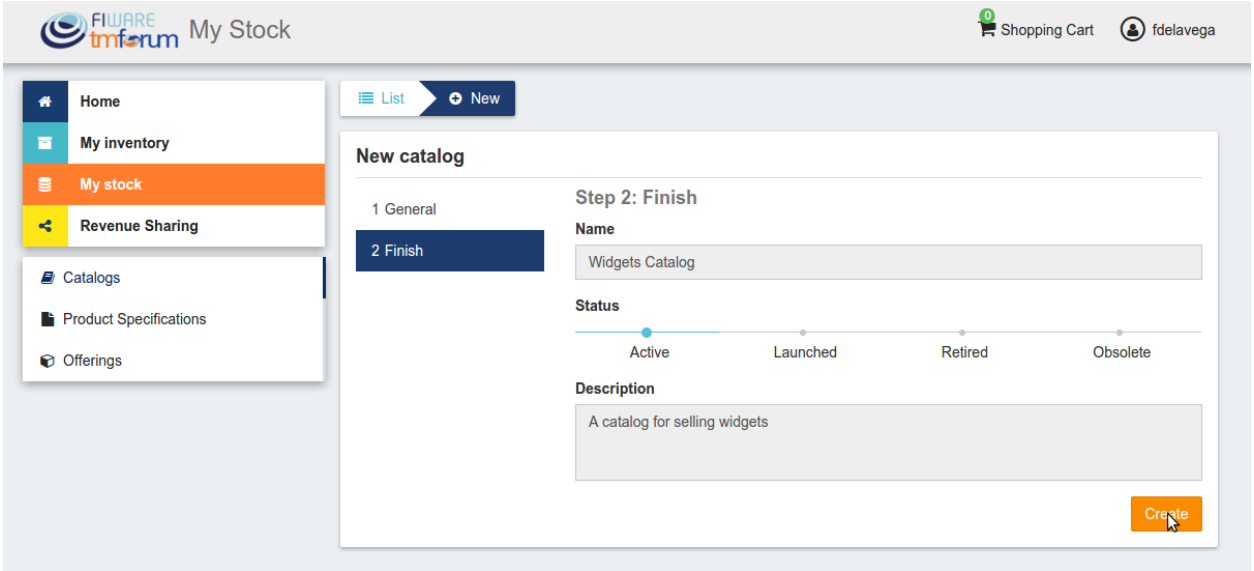




To create a new catalog click on the *New* button.

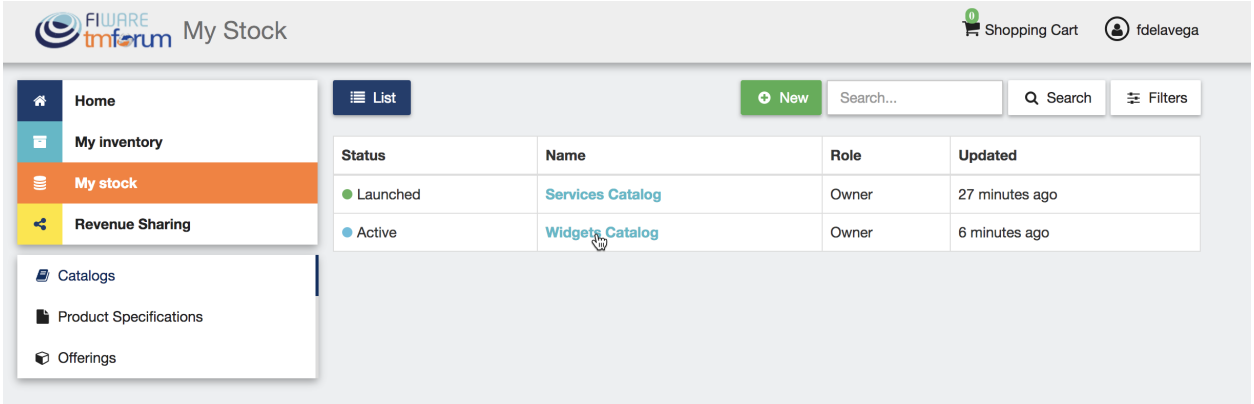



Then, provide a name and an optional description for the catalog. Once you have filled the fields, click on *Next*, and then on *Create*

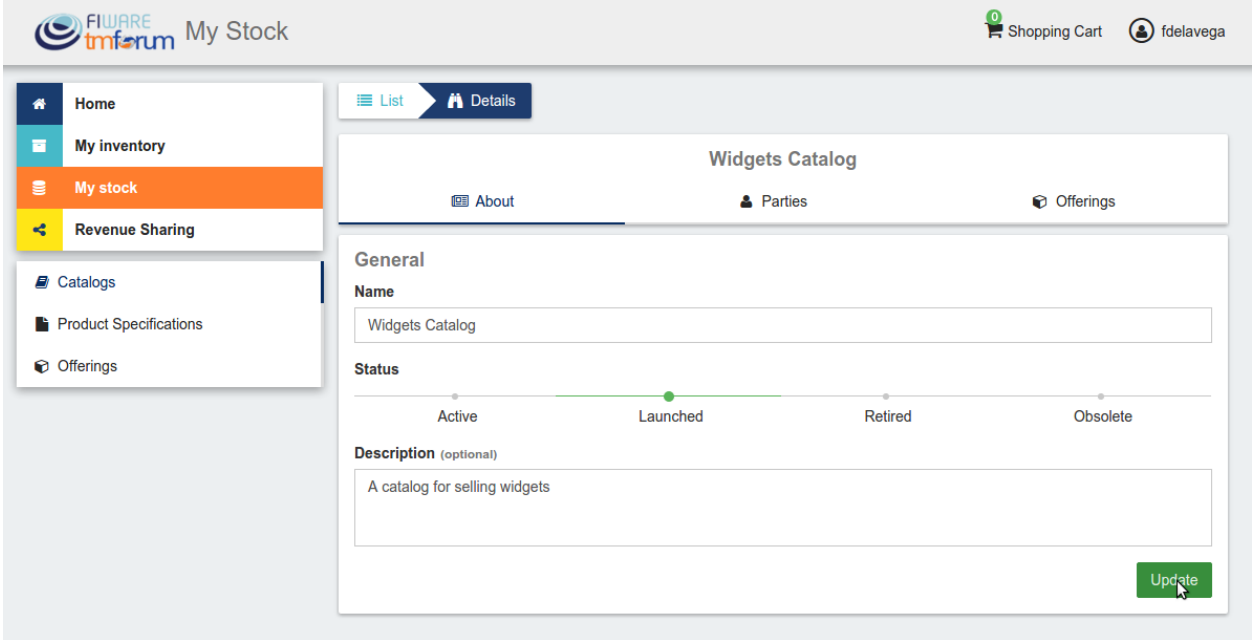



Sellers can also update their catalogs. To do that, click on the name of the catalog to open the update view.

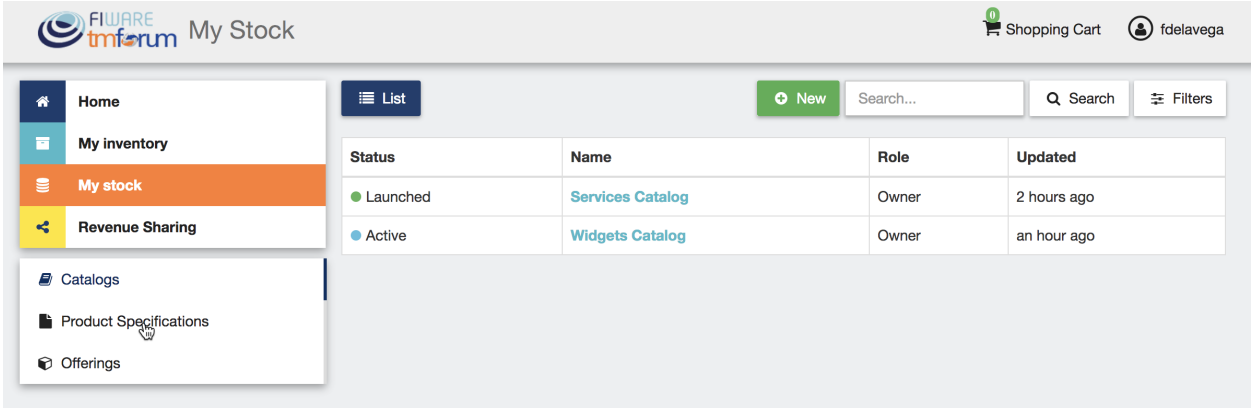



Then, update the fields you want to modify and click on *Update*. In this view, it is possible to change the *Status* of the catalog. To start monetizing the catalog, and make it appear in the *Home* you have to change its status to *Launched*

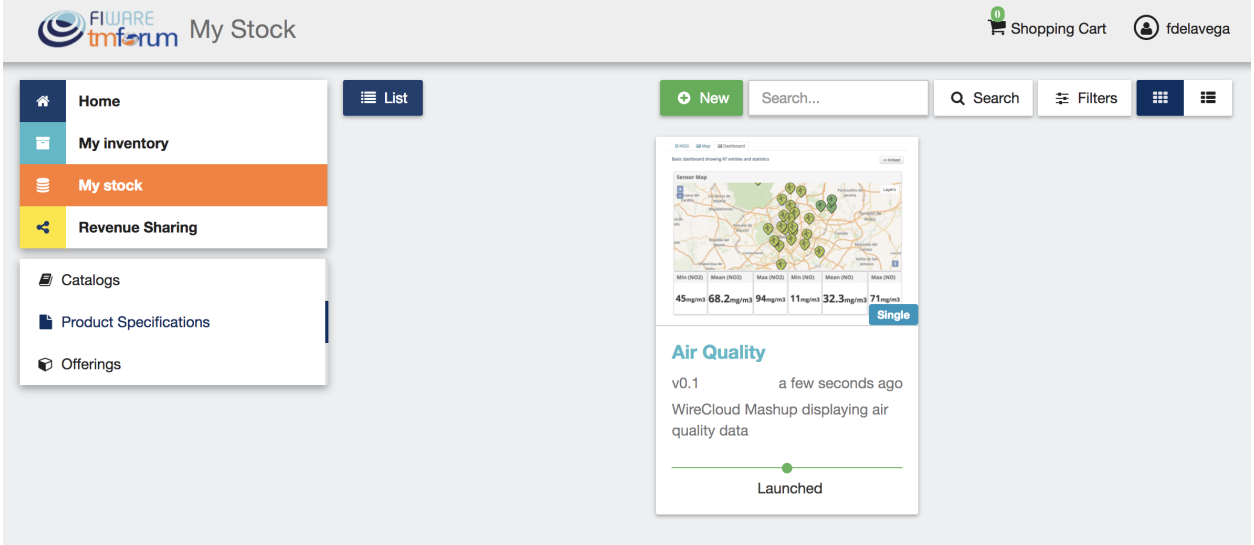


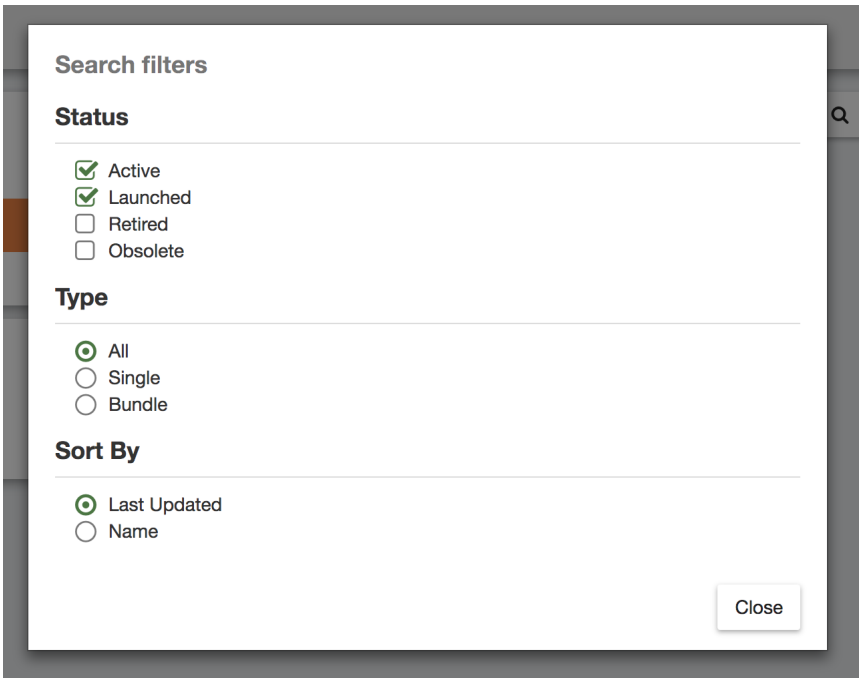

## Manage Product Specifications

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*

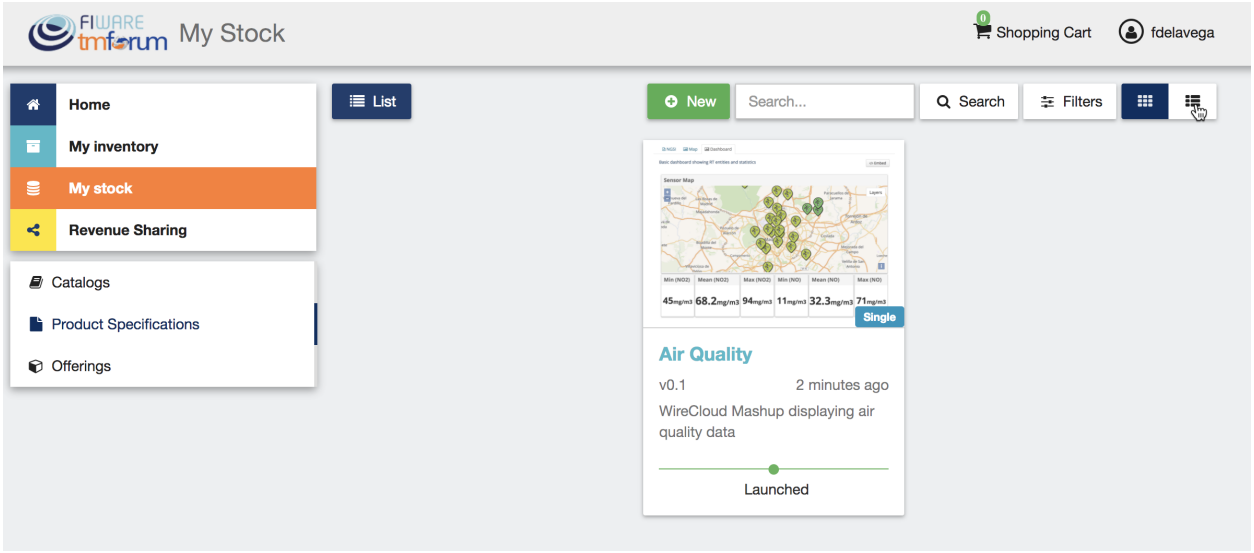


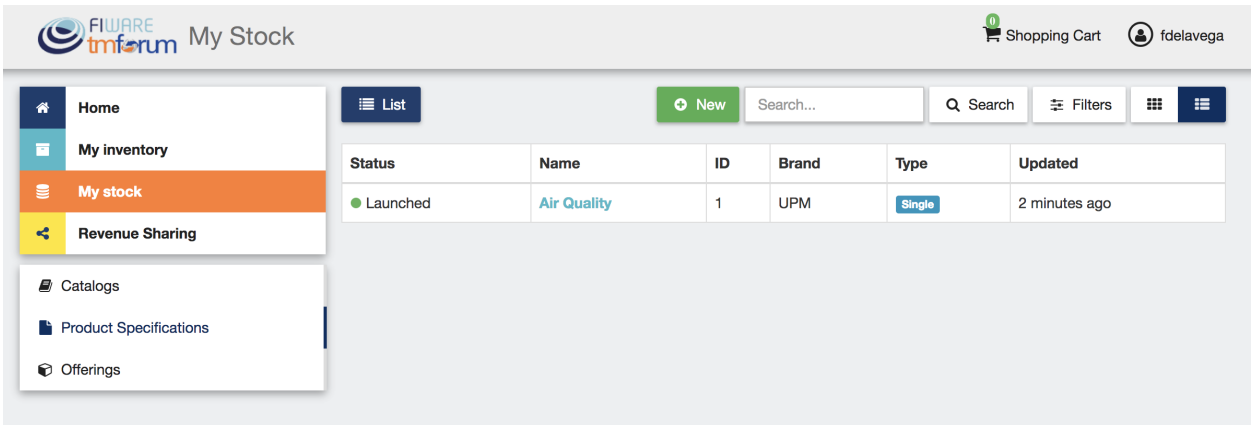

In the same way as catalogs, product specifications can be searched by keyword, sorted, or filtered by status and whether they are bundles or not. To filter or sort product specifications, click on *Filters*, choose the appropriate properties, and click on *Close*



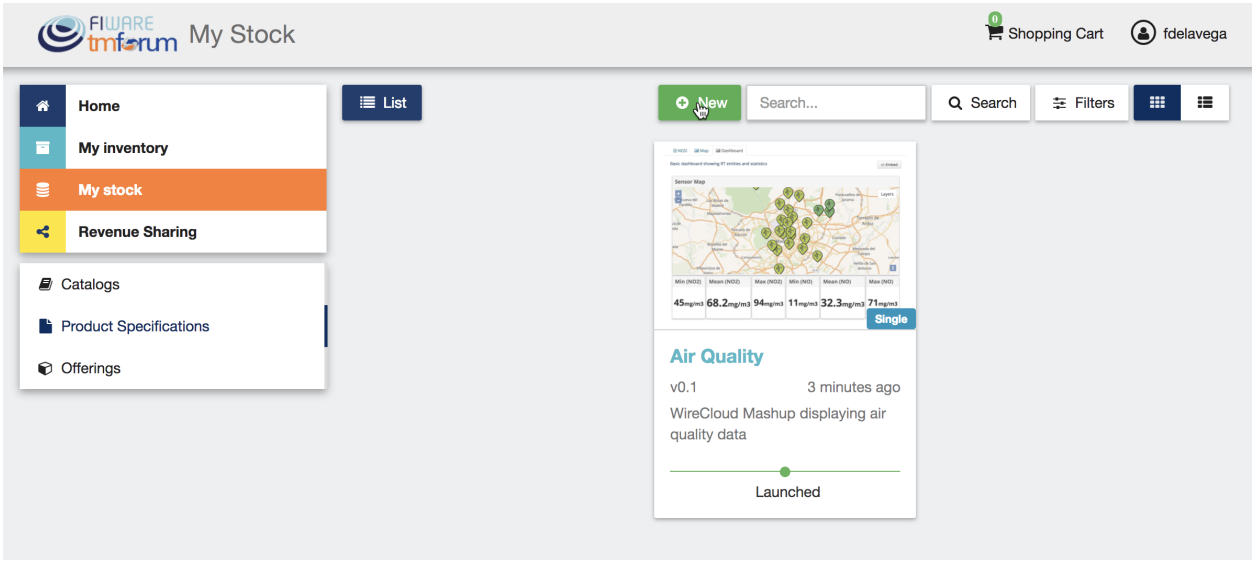




Additionally, it is possible to switch between the grid view and the tabular view using the provided buttons.

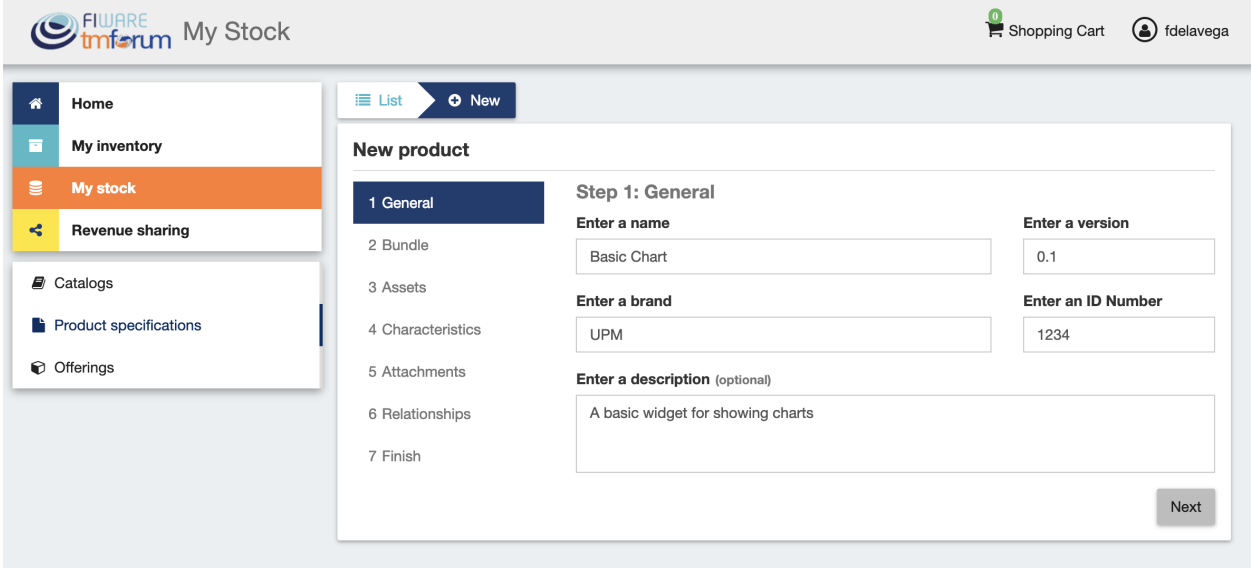


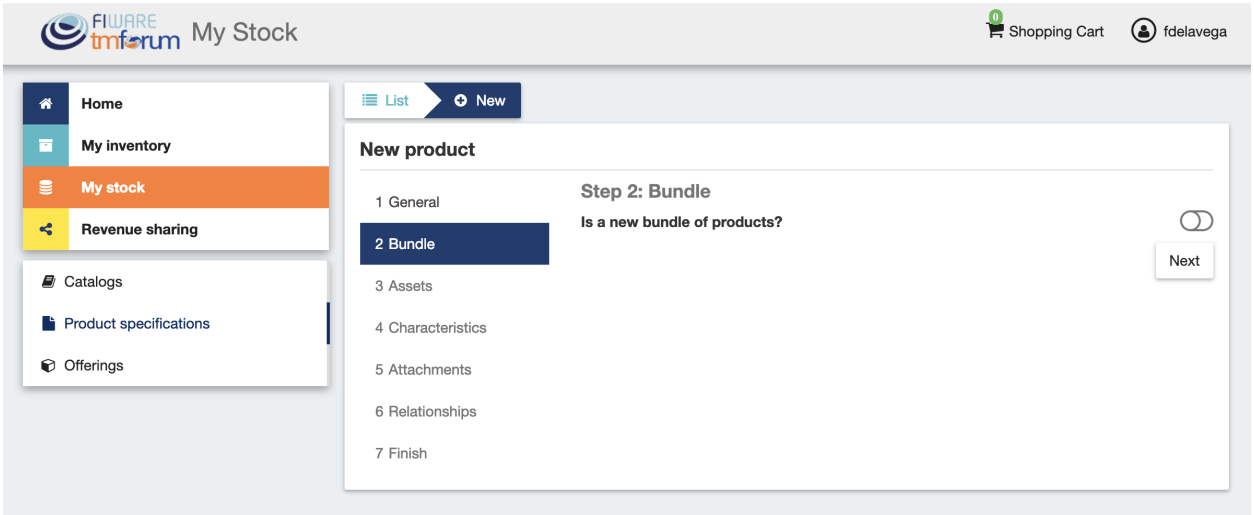

To create a new product specification click on *New*



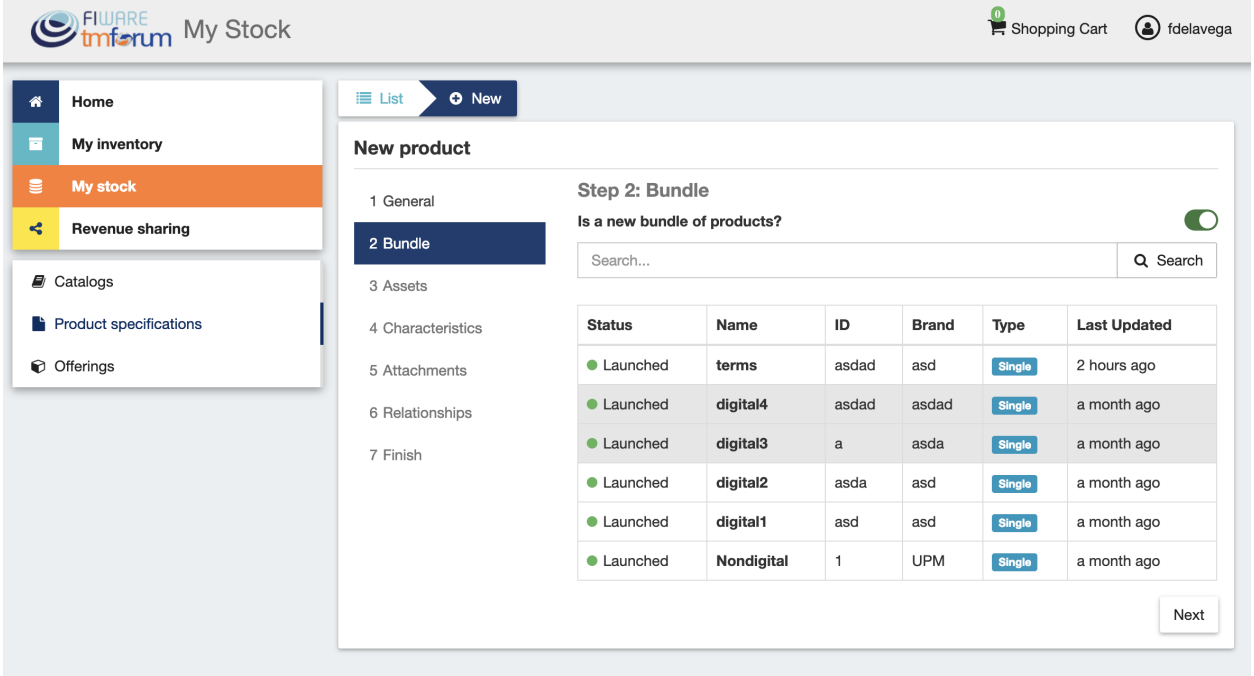

In the displayed view, provide the general information of the product spec. including its name, version, and an optional description. In addition, you have to include the product brand (Your brand), and an ID number which identifies the product in your environment. Then, click on *Next*.




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*




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.



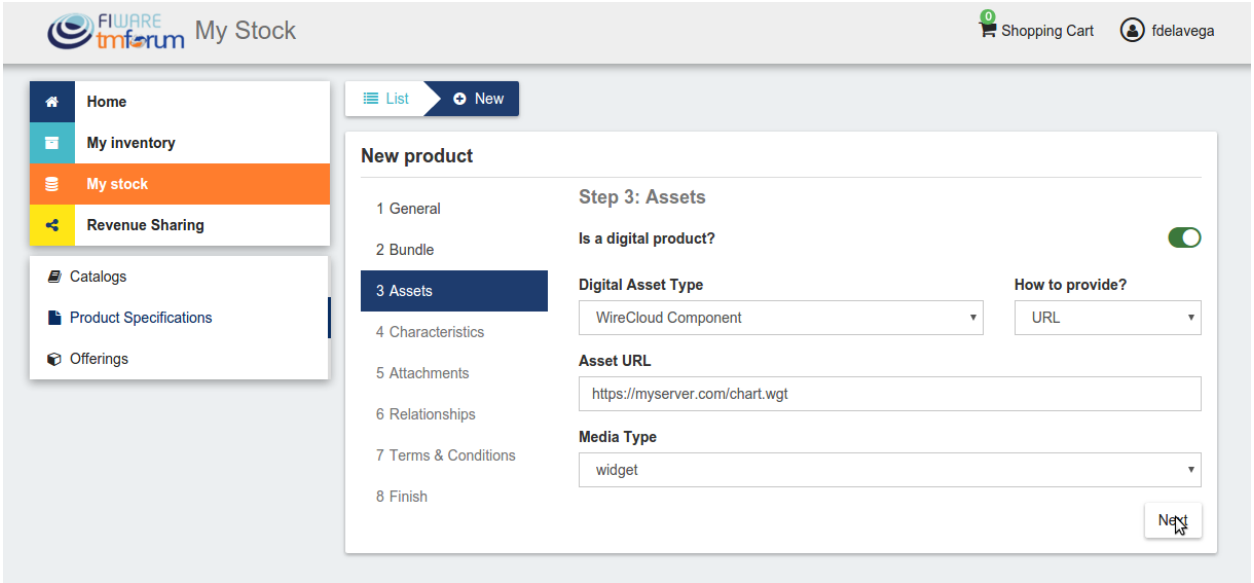

In the next step you can choose if your product is a digital product. If this is the case, you will be required to provide the asset.

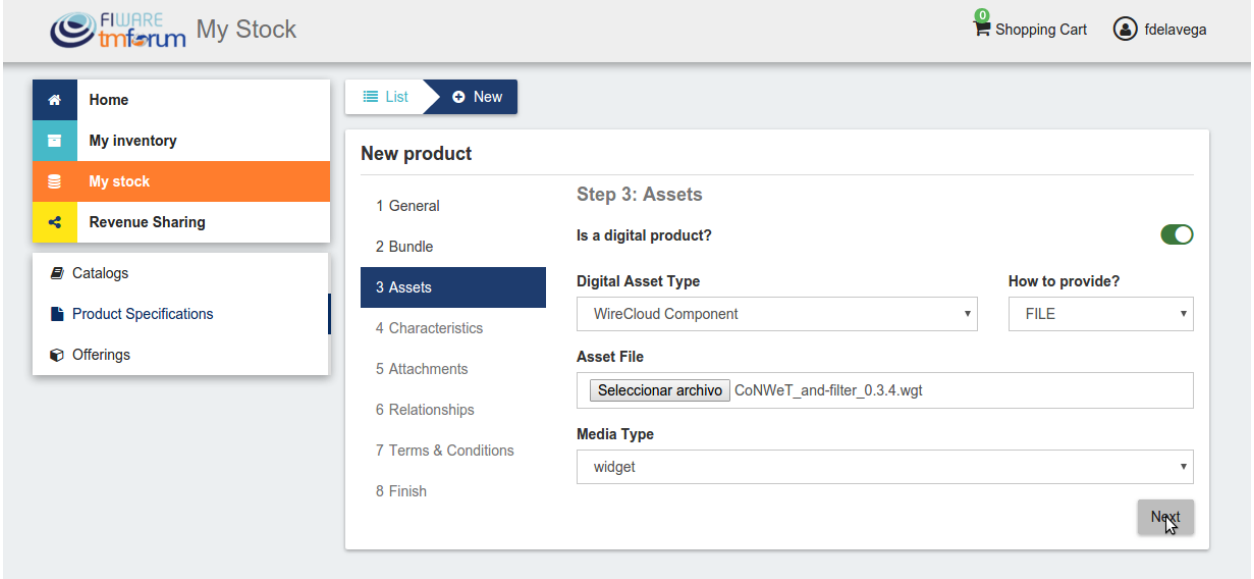
---

**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

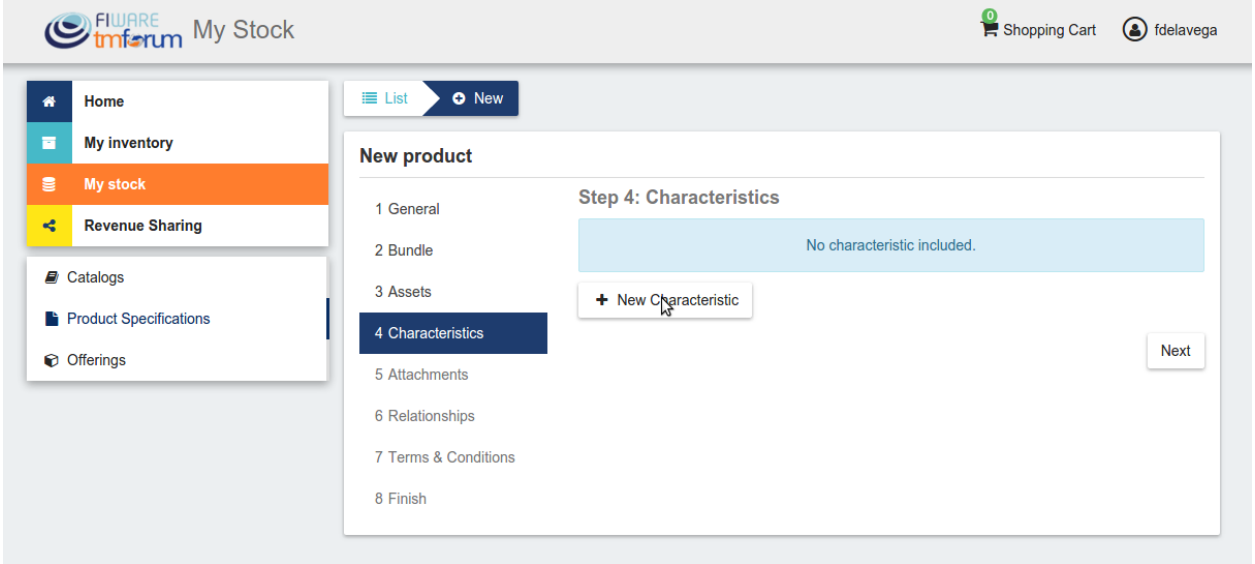
---

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.

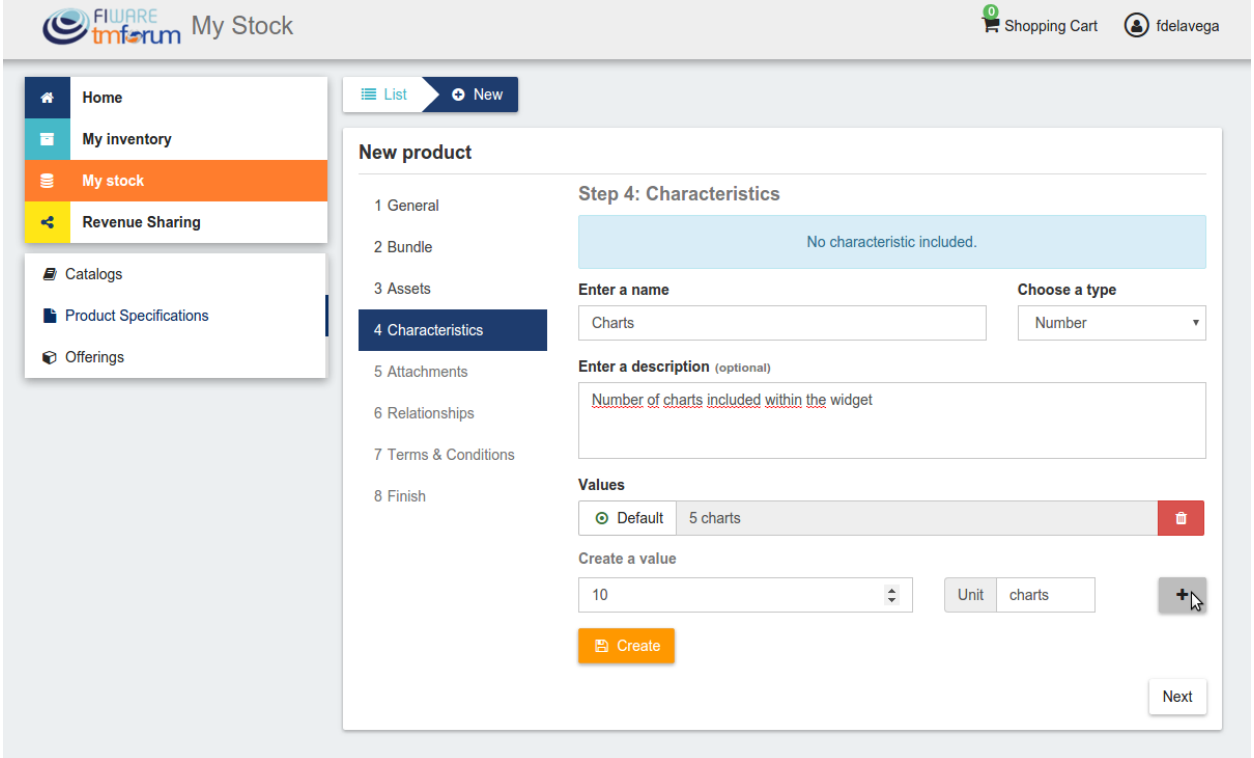




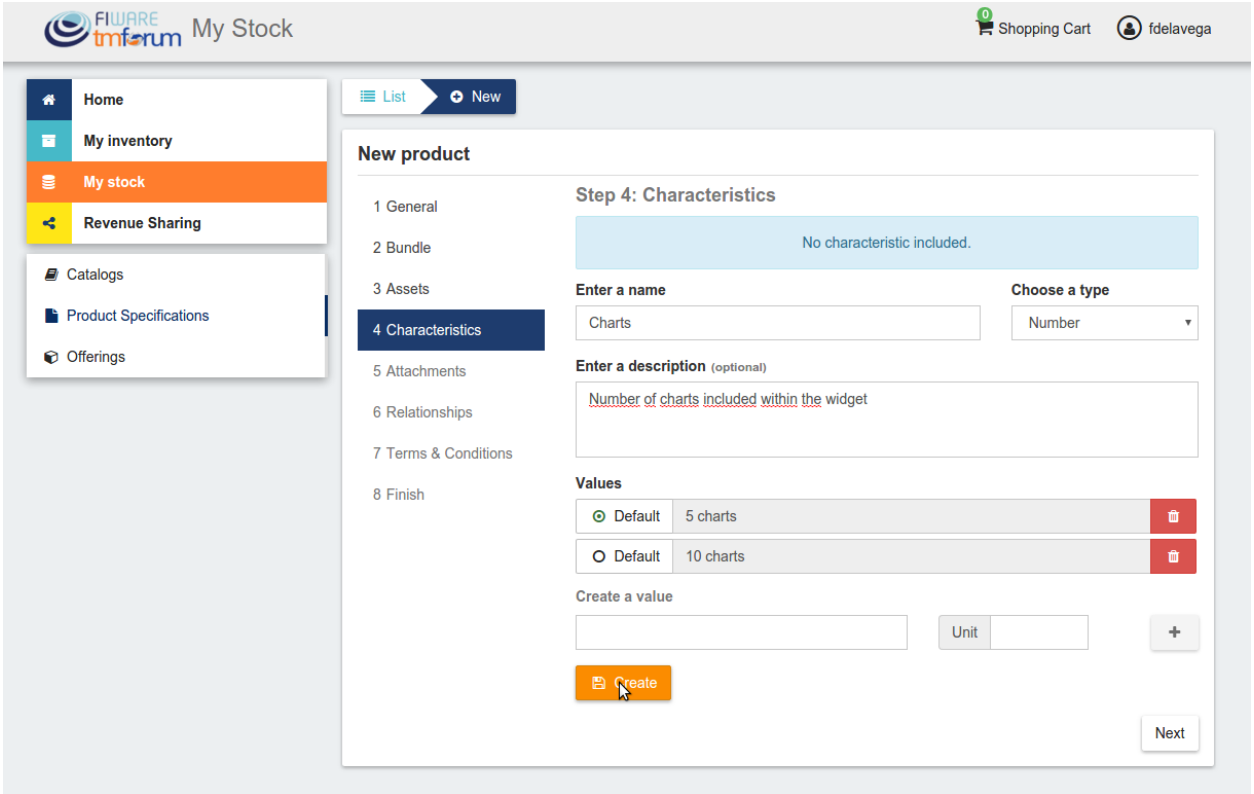

The next step in the creation of a product is including its characteristics. For including a new characteristic click on *New Characteristic*



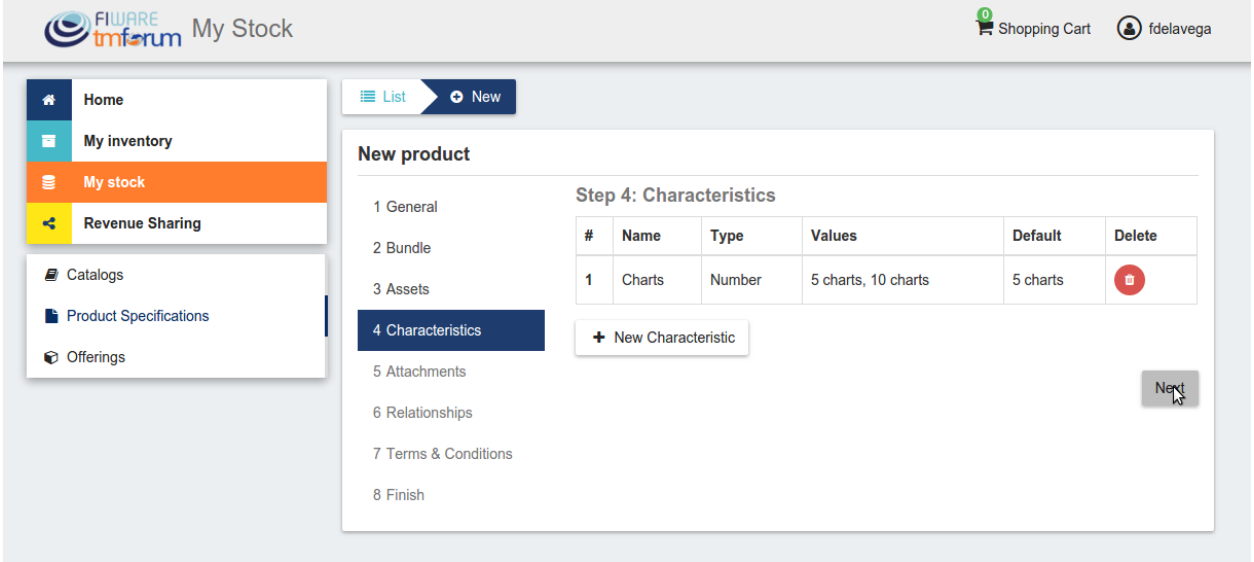

In the form, include the name, the type (string or number) and an optional description. Then create the values of the characteristic by filling the *Create a value* input and clicking on +.



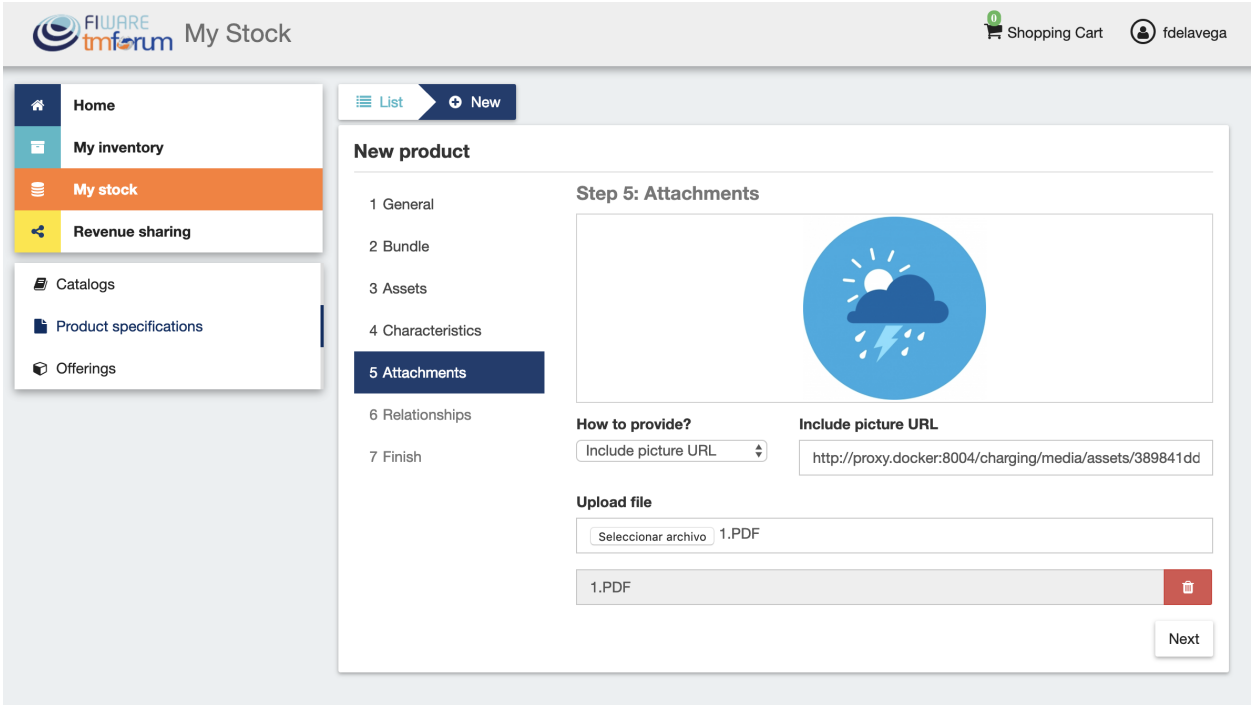

Once you have included all the characteristic info, save it clicking on *Create*

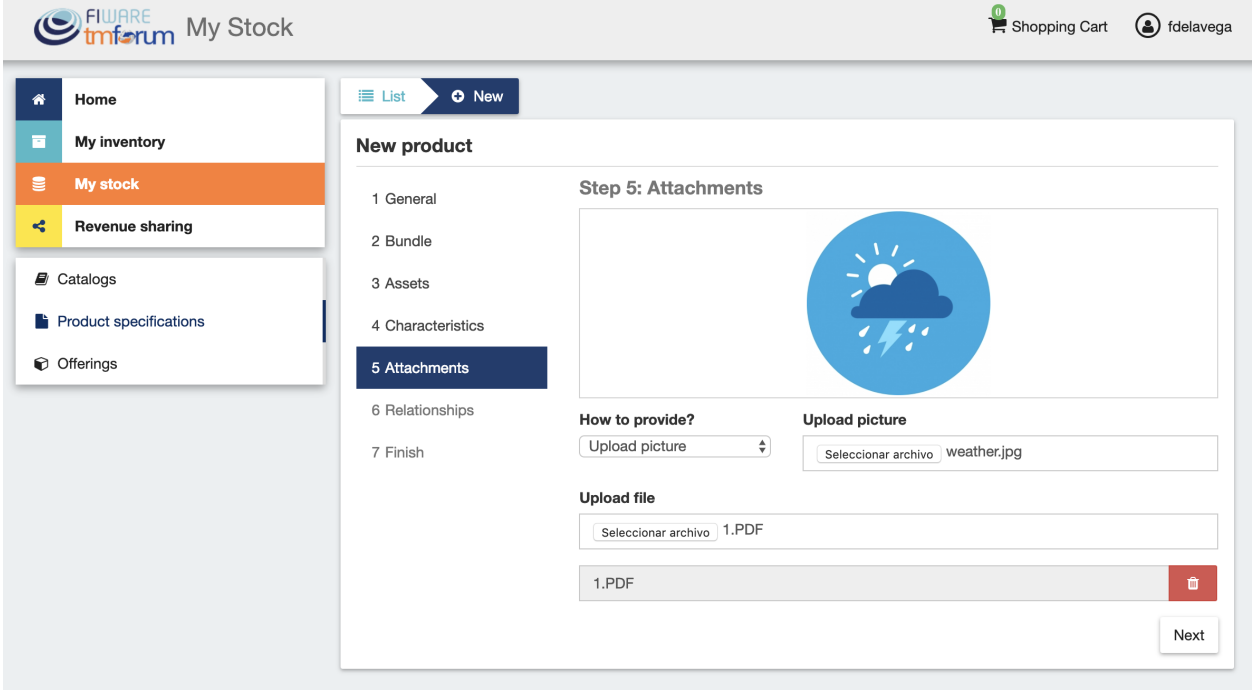



Once you have included all the required characteristics click on *Next*

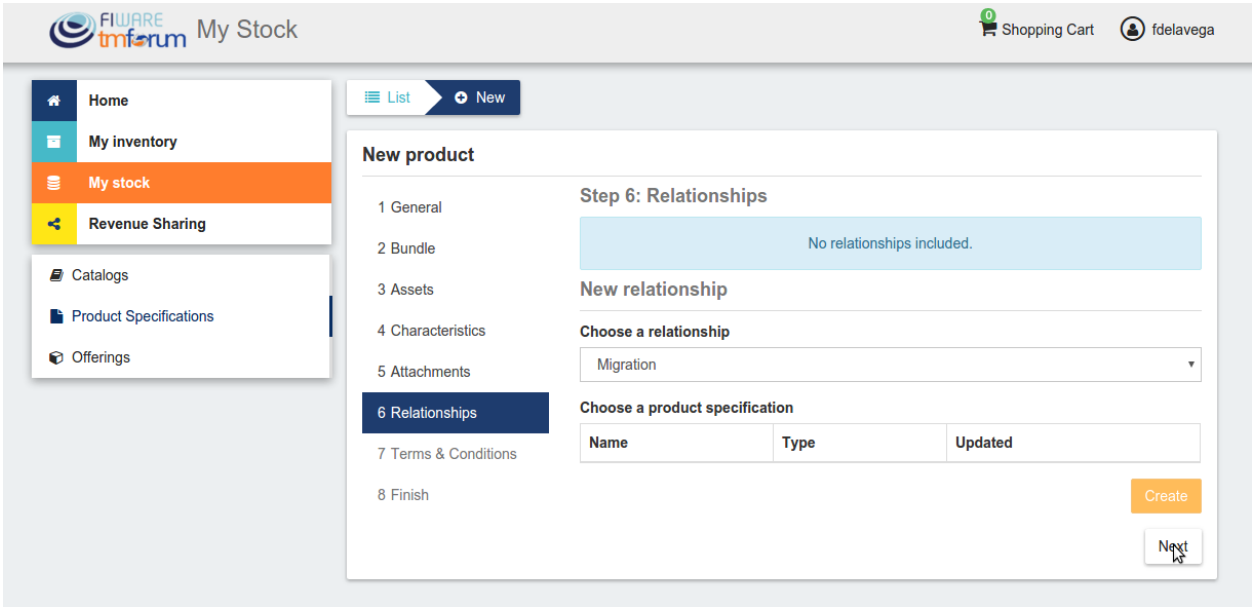



In the next step you can include a picture for your product spec. You have two options, providing an URL pointing to the picture or directly uploading it. In addition, it is possible to include multiple file attachments to the product spec, such as images, PDF documentation, etc. Once provided click *Next*

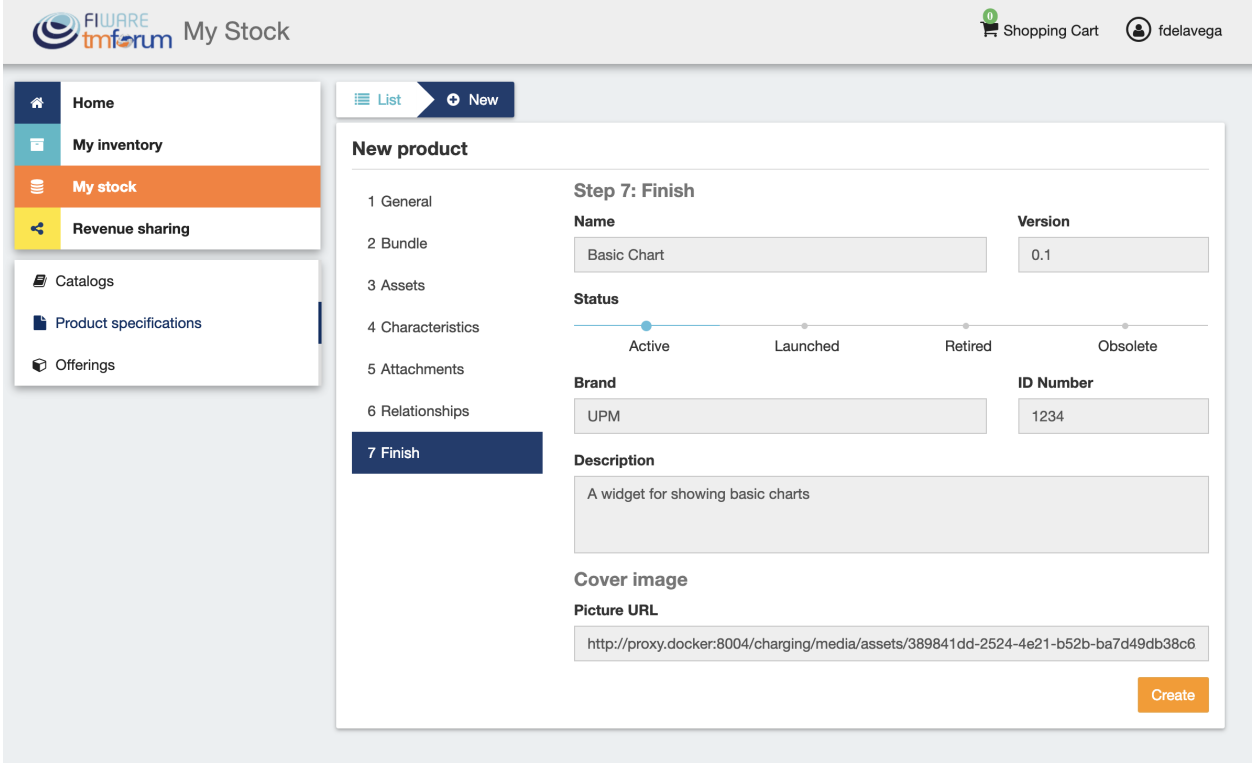




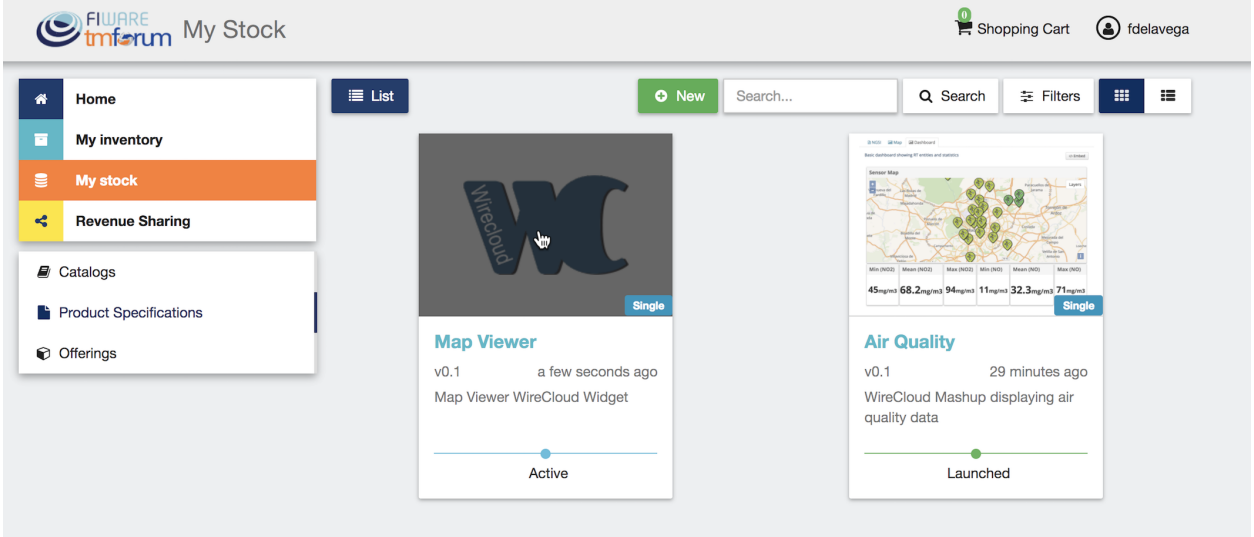

In the last step, you can specify relationships of the product you are creating with other of your product specs.



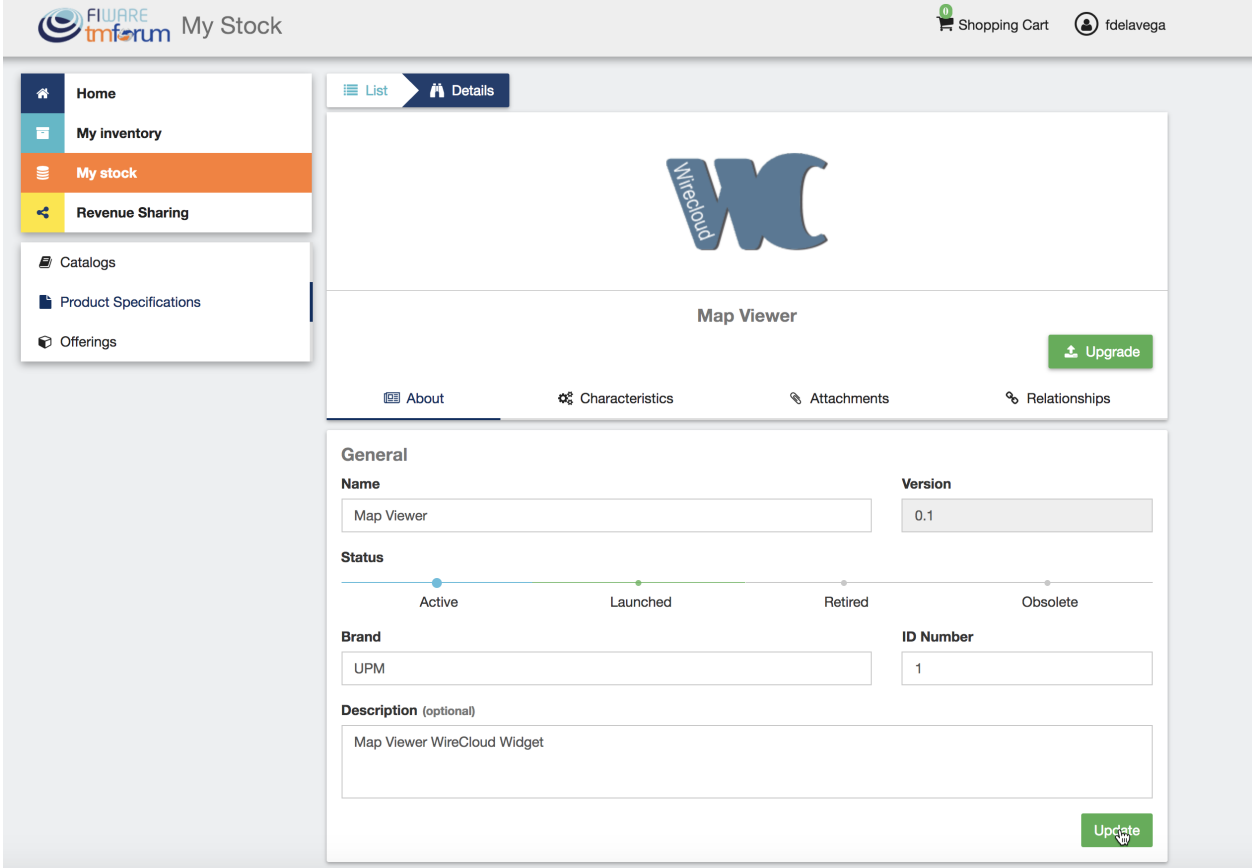

Once done click on *Next* and then on *Create*




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

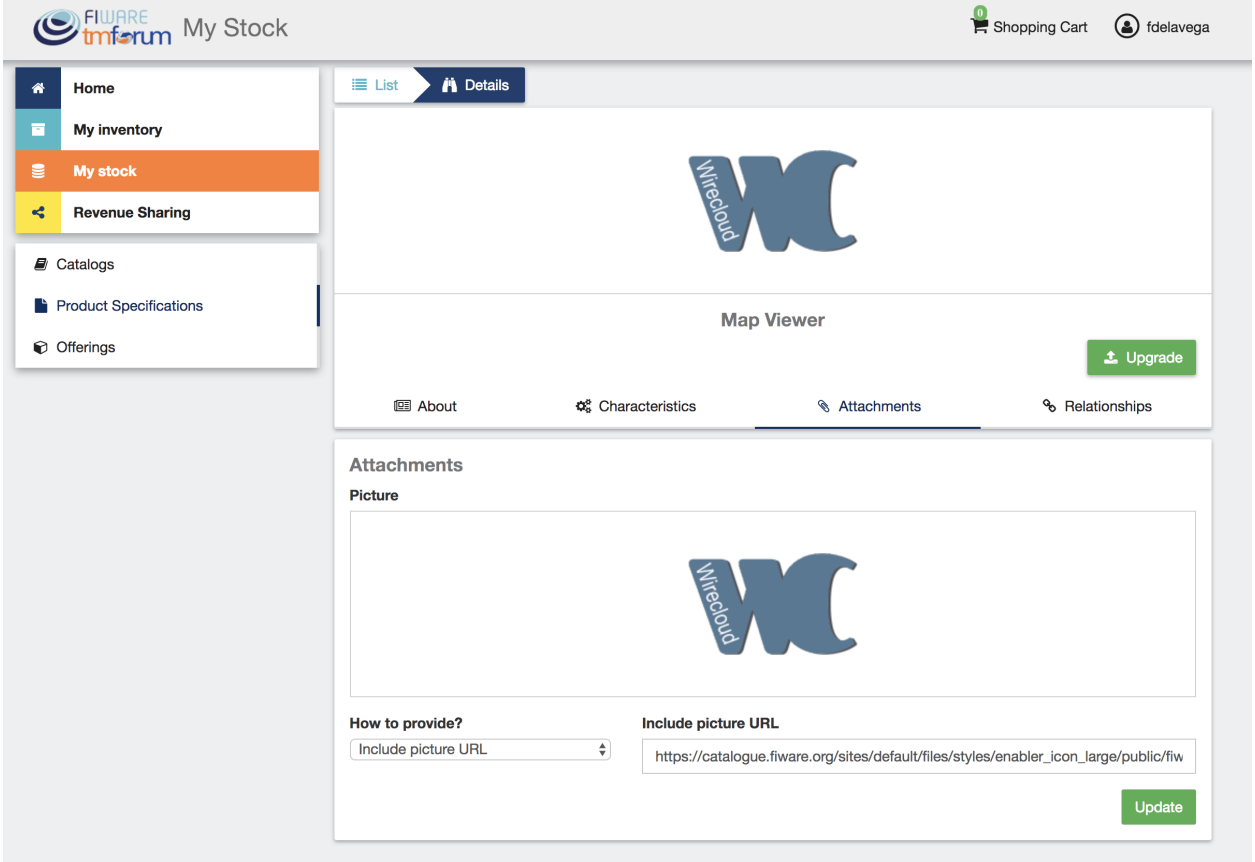


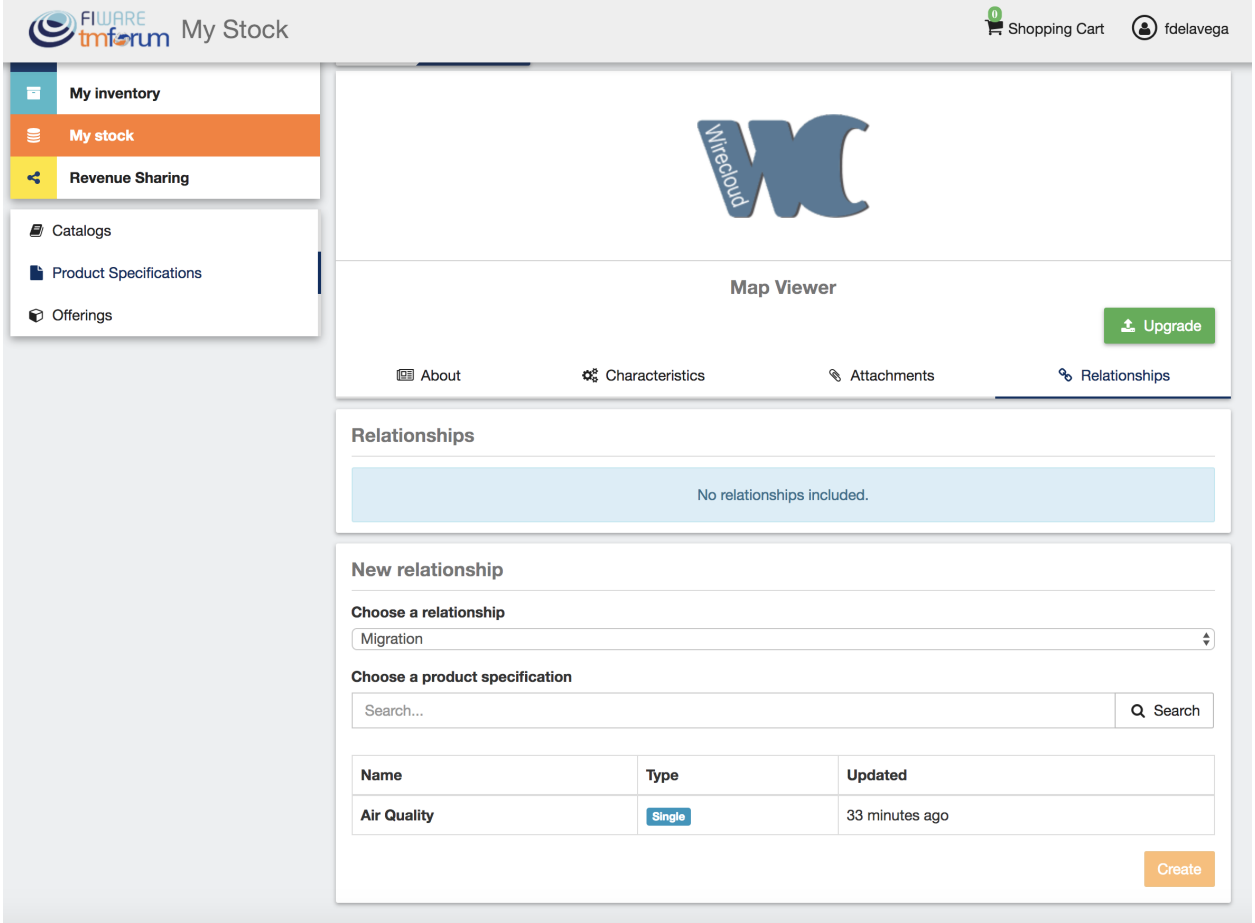

Update the required values and click on *Update*. Note that for start selling an offering that includes the product specification you will be required to change its status to *Launched*



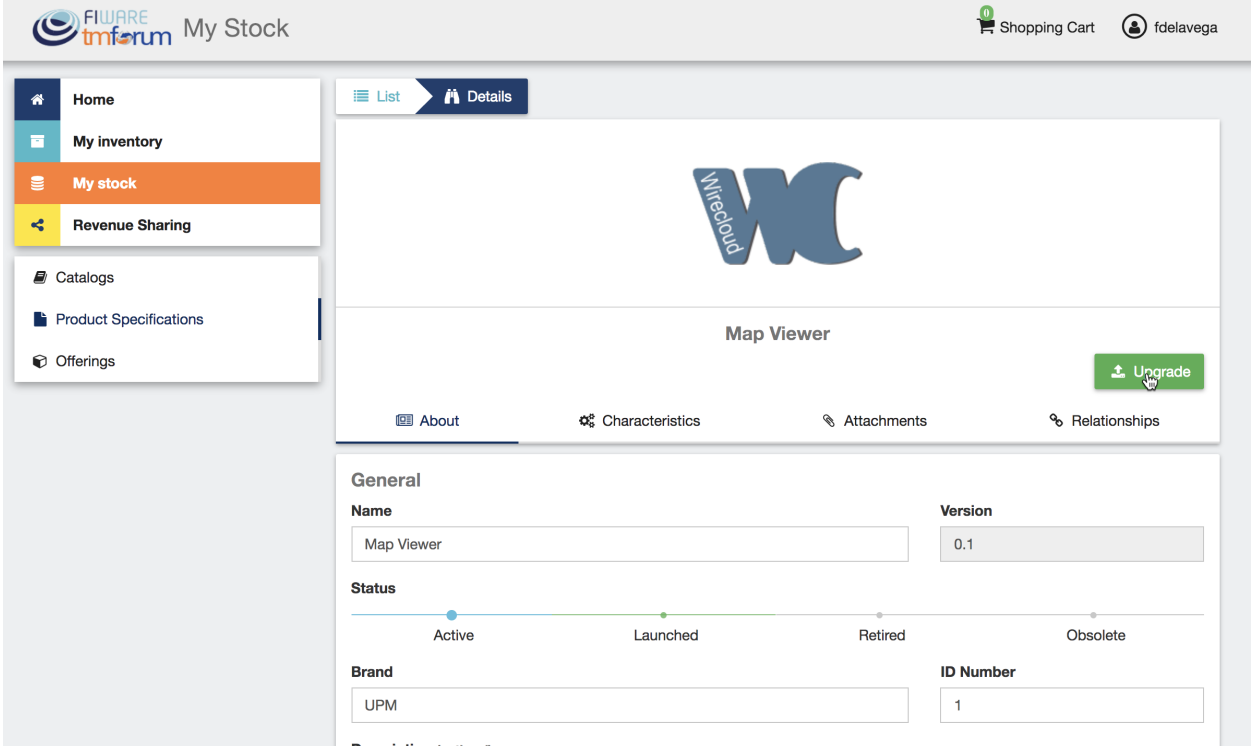

**Note:** For digital products it is not allowed to update the version using this form. Instead it is required to follow the process for upgrading the product version.

The basic information of the product specification is not the only information that can be updated, but it is also possible to update the *Attachments* and the *Relationships* by clicking of the related tab.

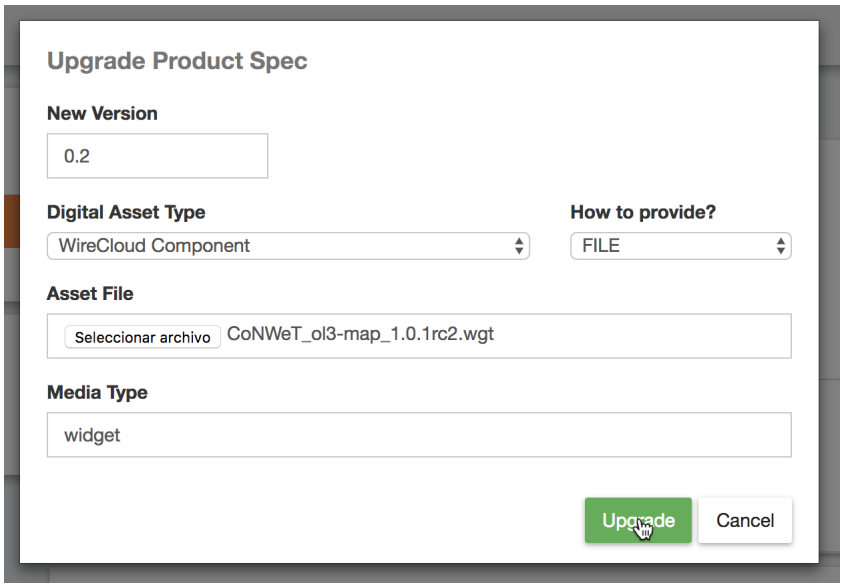

FIWARE tmforum My Stock
Shopping Cart
fdelavega
Home
My inventory
My stock
Revenue Sharing
Catalogs
Product Specifications
Offerings
List
Details
Wirecloud
Map Viewer
Upgrade
About
Characteristics
Attachments
Relationships
Attachments
Picture
Wirecloud
How to provide?
Include picture URL
Include picture URL
https://catalogue.fiware.org/sites/default/files/styles/enabler_icon_large/public/fiw
Update




The displayed details form can be used for digital products specifications in order to provide new versions of the digital assets being offered. This can be done by clicking on *Upgrade*.



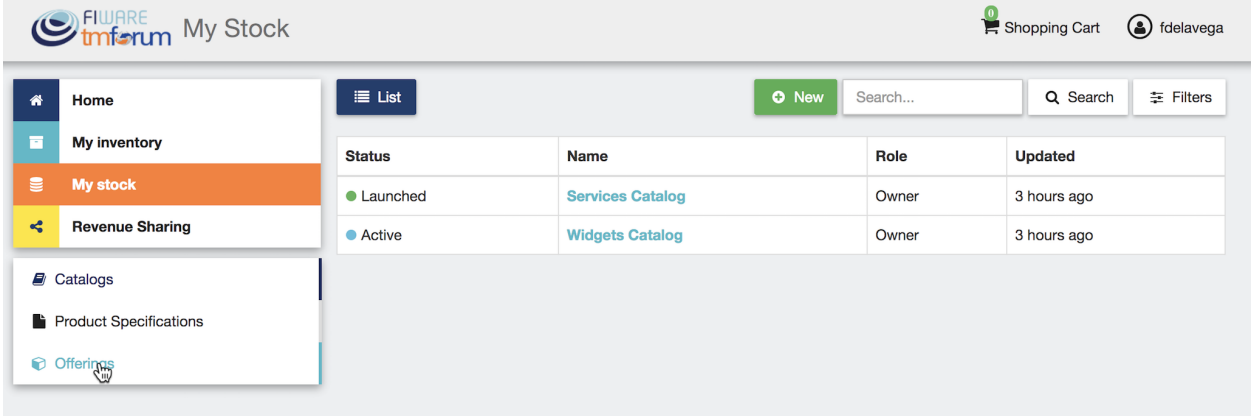

In the displayed form, it is required to include a new version for the product specification and to provide the new digital asset to be offered.



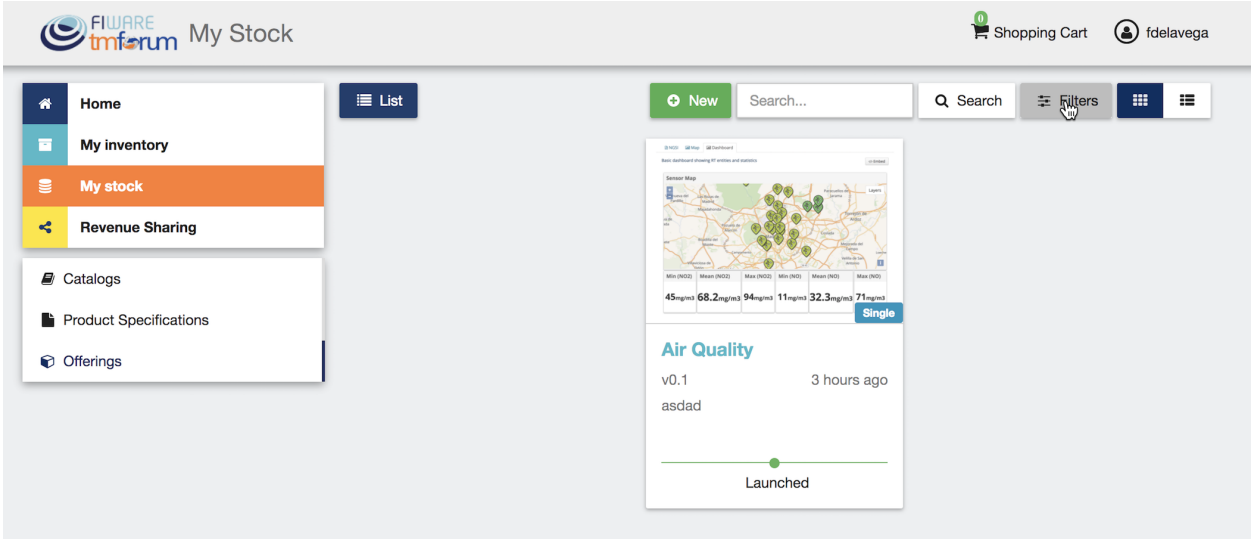

**Note:** All the customers who have acquired an offering including the current product specification will be able to access to the new version of the digital asset.

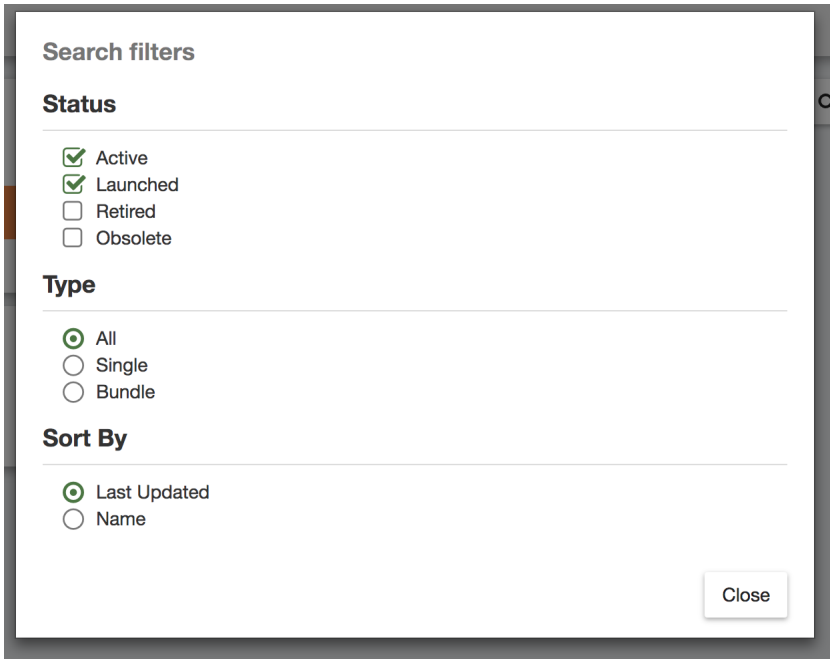
### Manage Product Offerings

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*

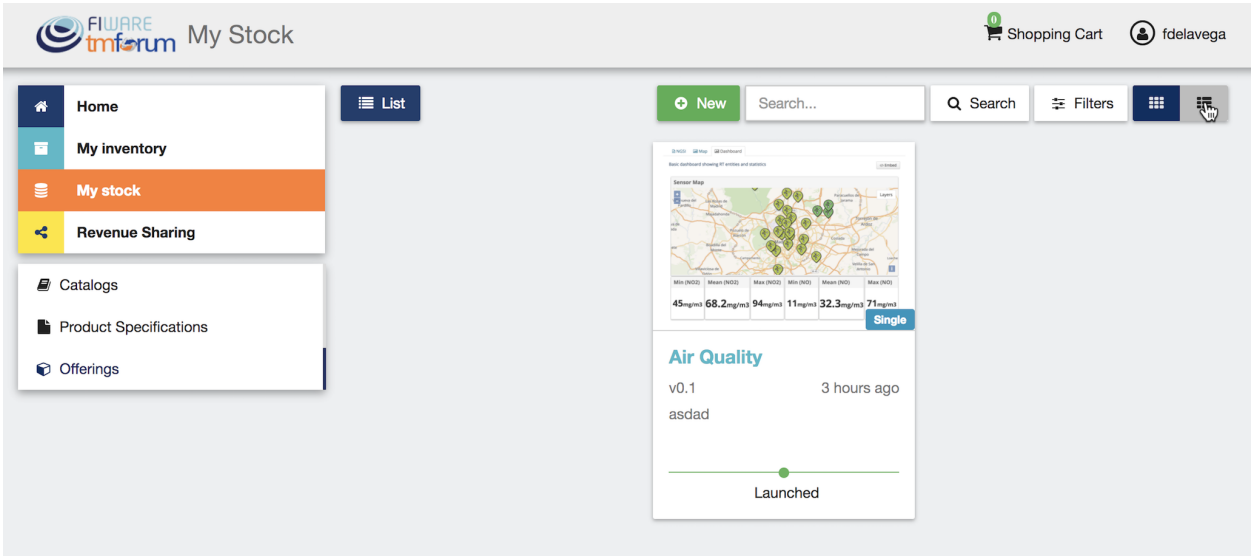


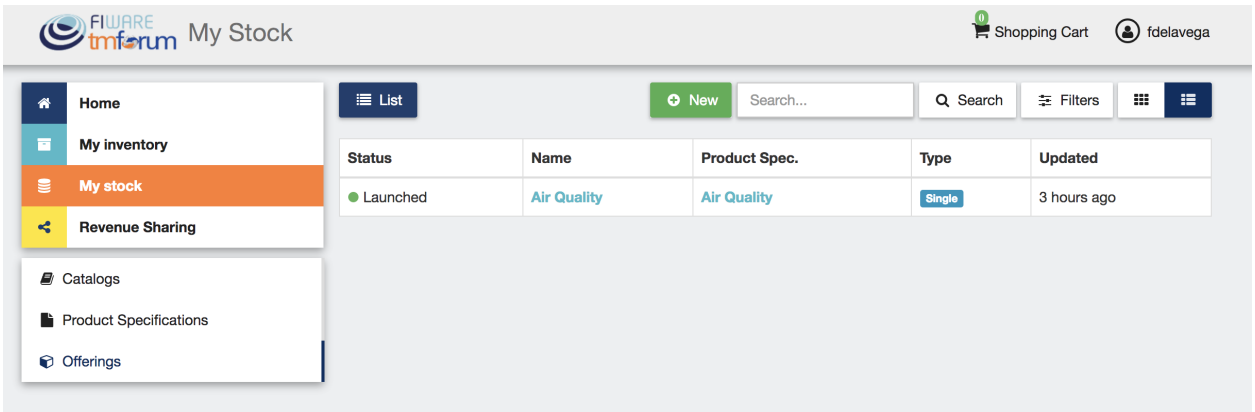

The existing product offerings can be searched by keyword, sorted, or filtered by status and whether they are bundles or not. To filter or sort product offerings, click on *Filters*, choose the appropriate properties, and click on *Close*



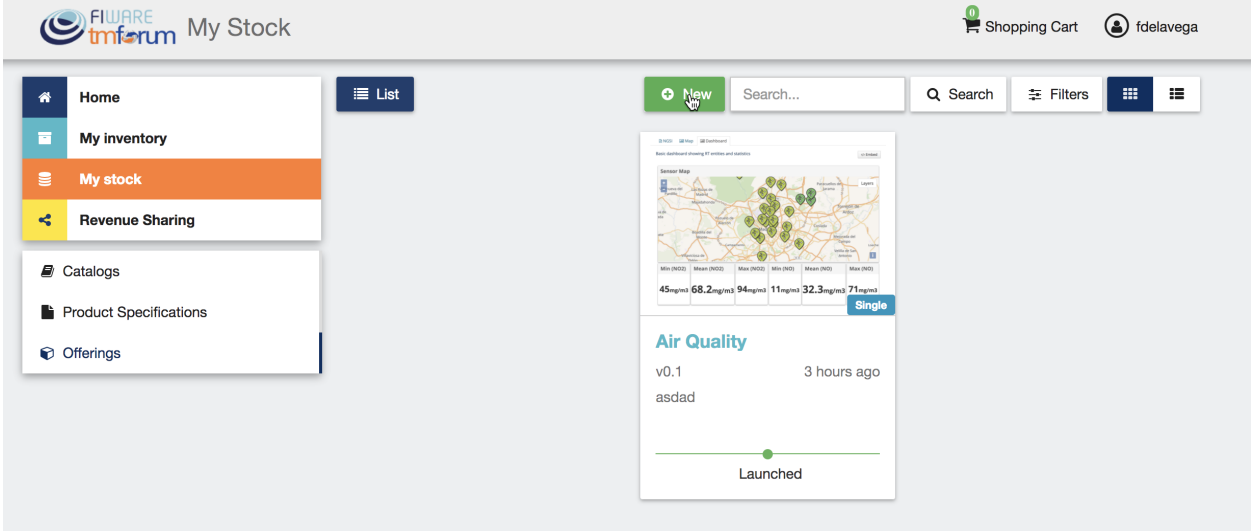



Additionally, it is possible to switch between the grid view and the tabular view by clicking on the specific button.

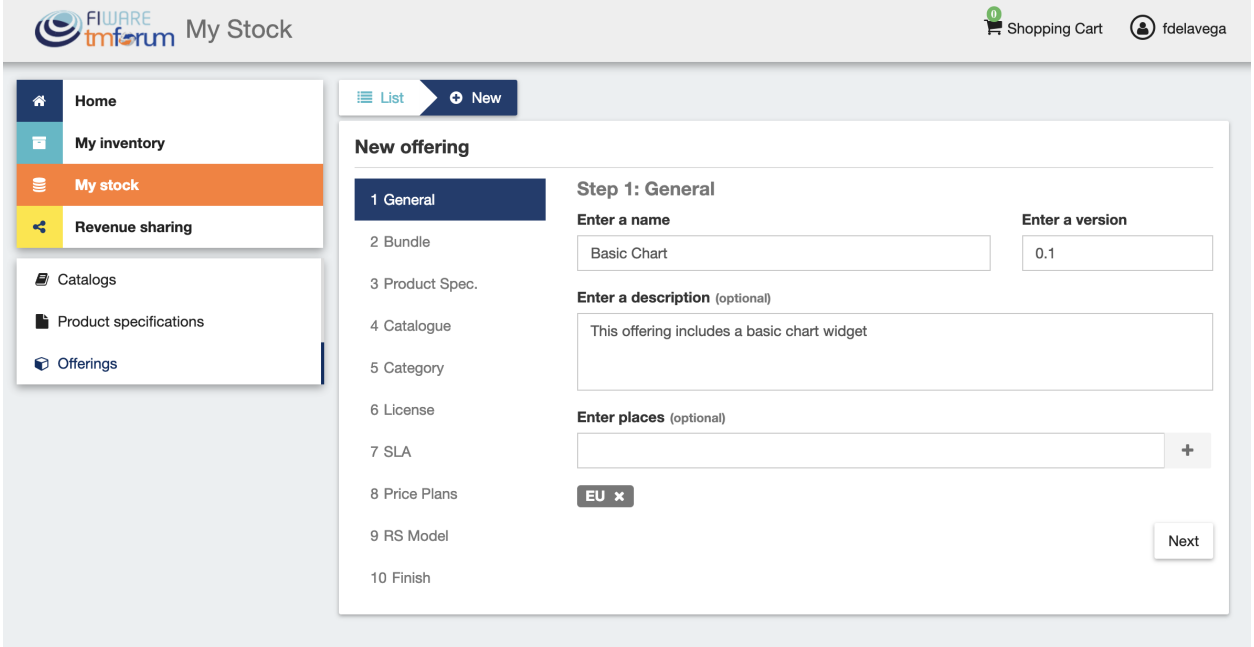


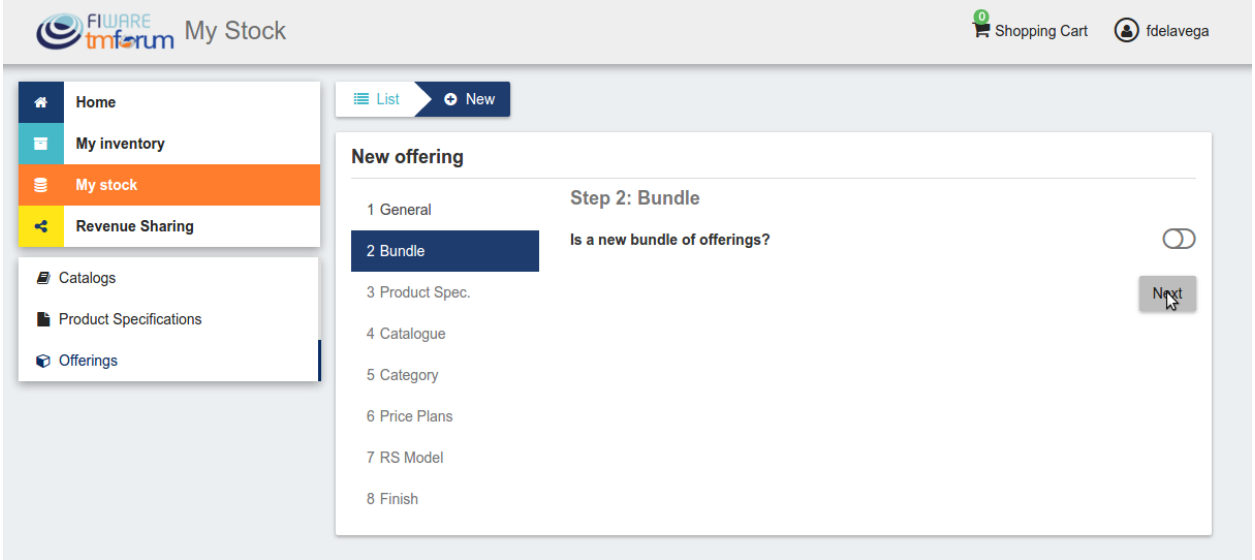
To create a new offering click on *New*



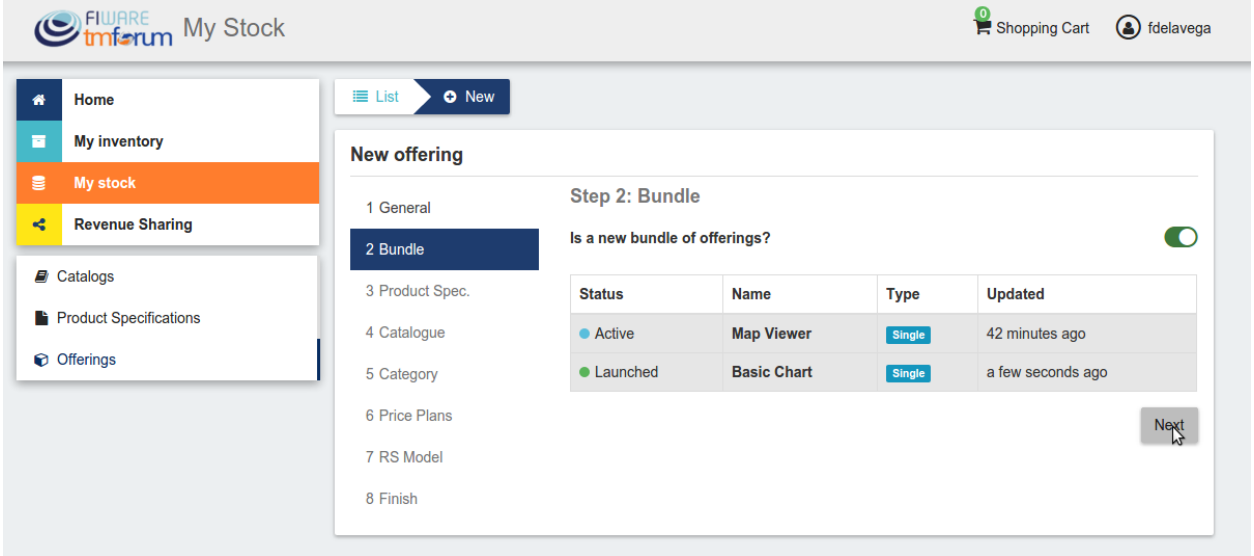

In the displayed form, include the basic info of the offering. Including, its name, version, an optional description, and an optional set of places where the offering is available. Once the information has been provided click on *Next*



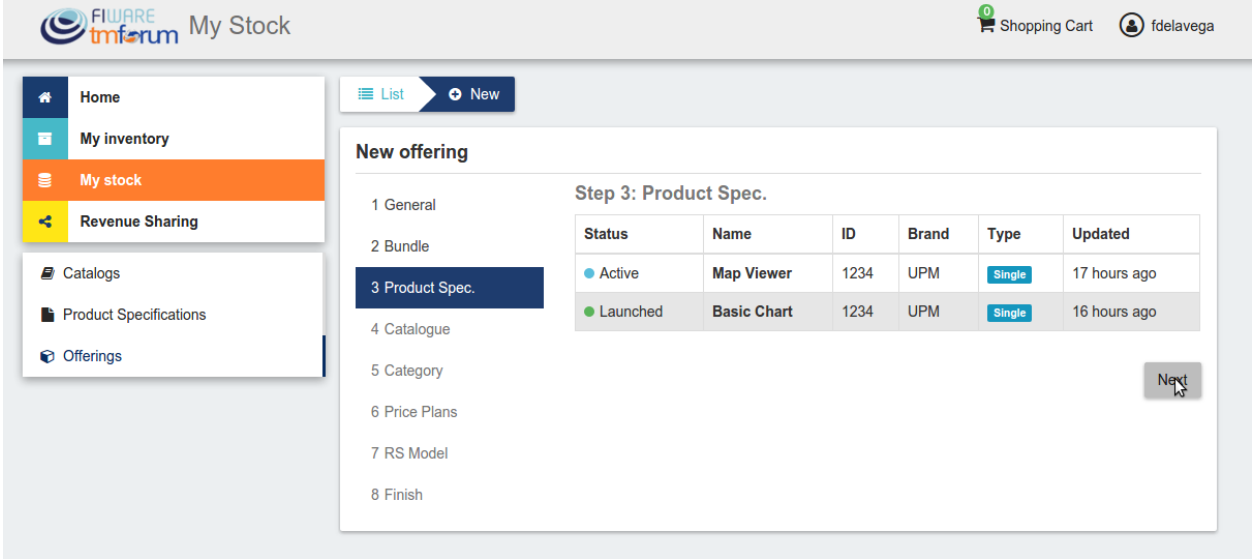

In the next step, you can choose whether your offering is a bundle or not. In this case, offering bundles are logical containers that allow you to provide new pricing models when a set of offerings are acquired together. Once selected click on *Next*




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

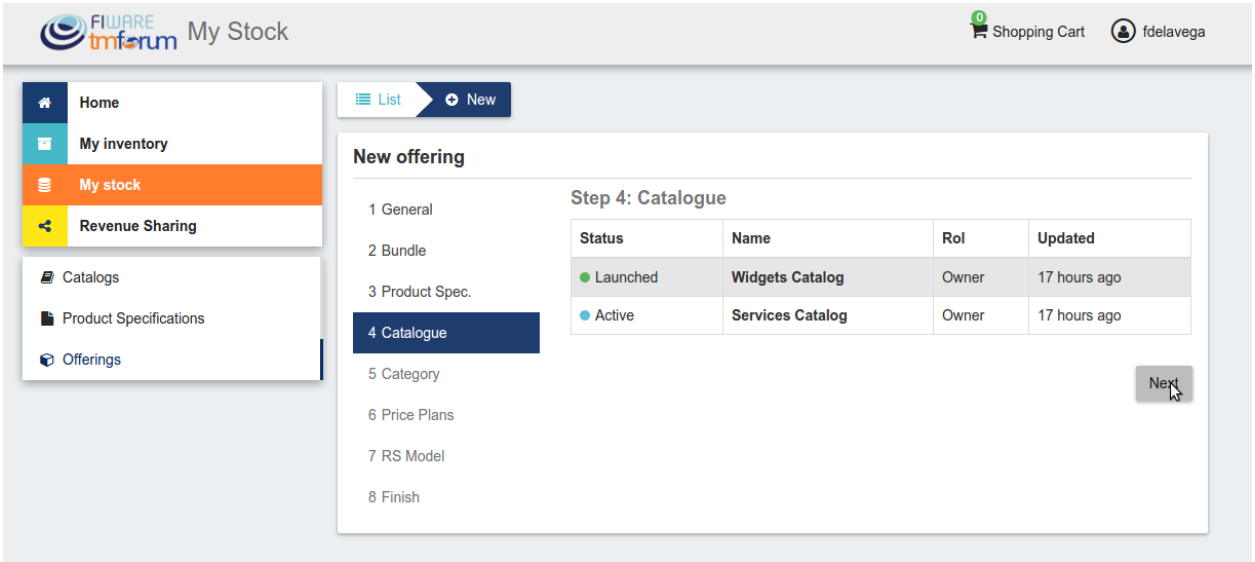



In the next step you have to select the product specification that is going to be monetized in the current offering. Once selected click on *Next*.

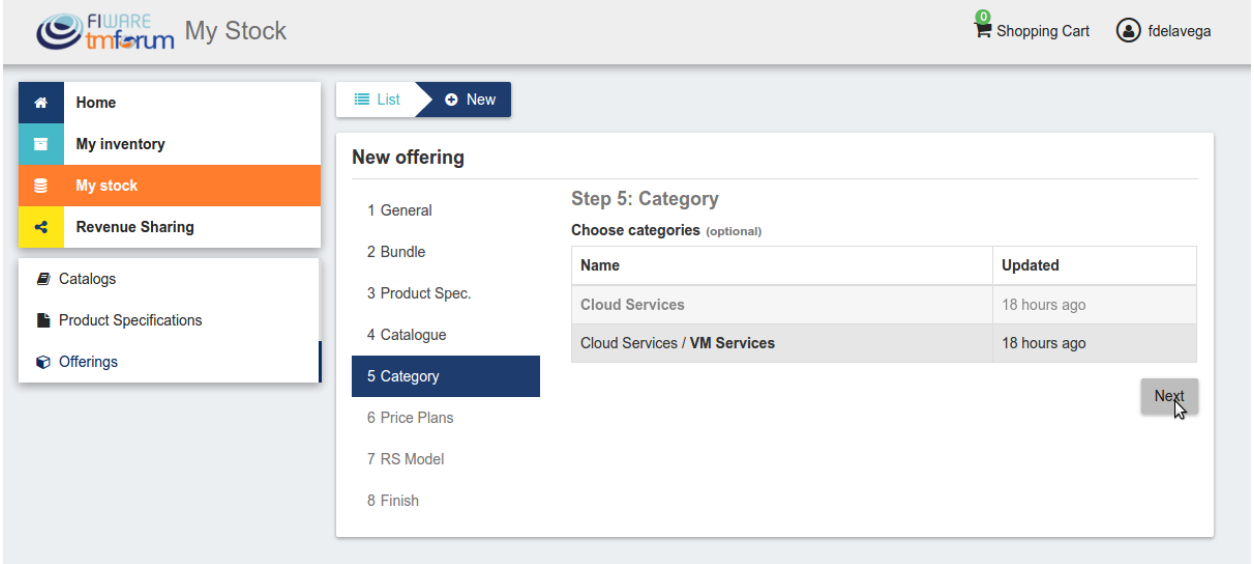



**Note:** If you are creating an offering bundle, you will not be allowed to include a product specification

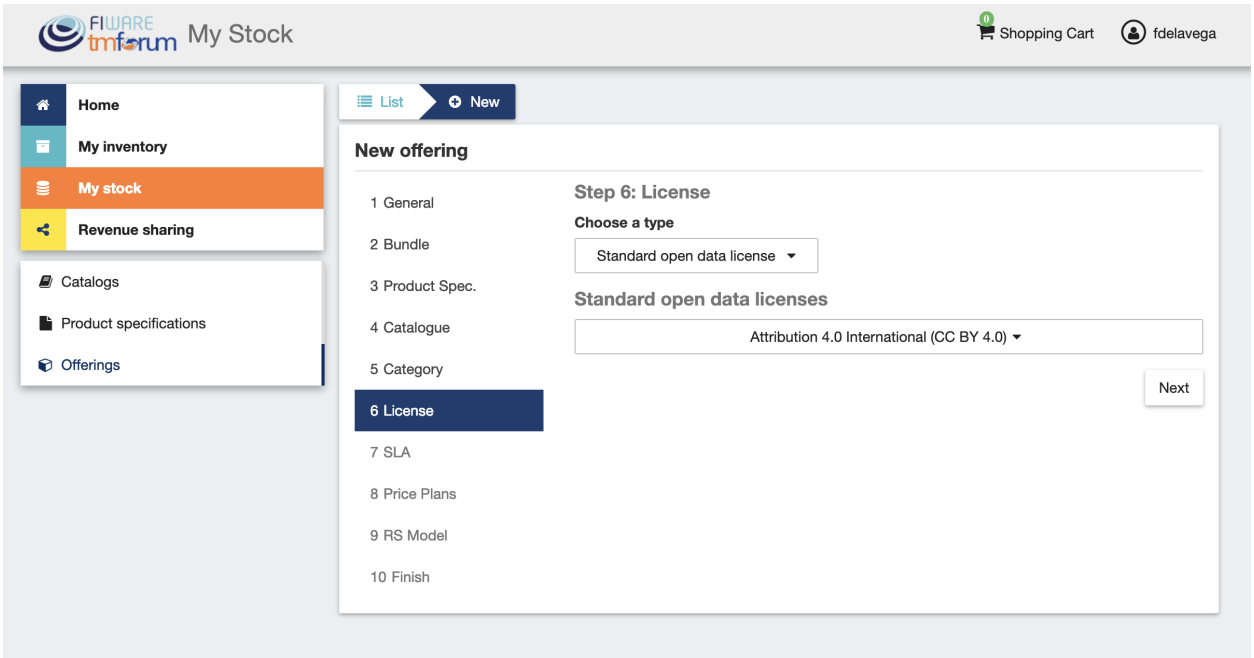
Then, you have to select the catalog where you want to publish you offering and click on *Next*

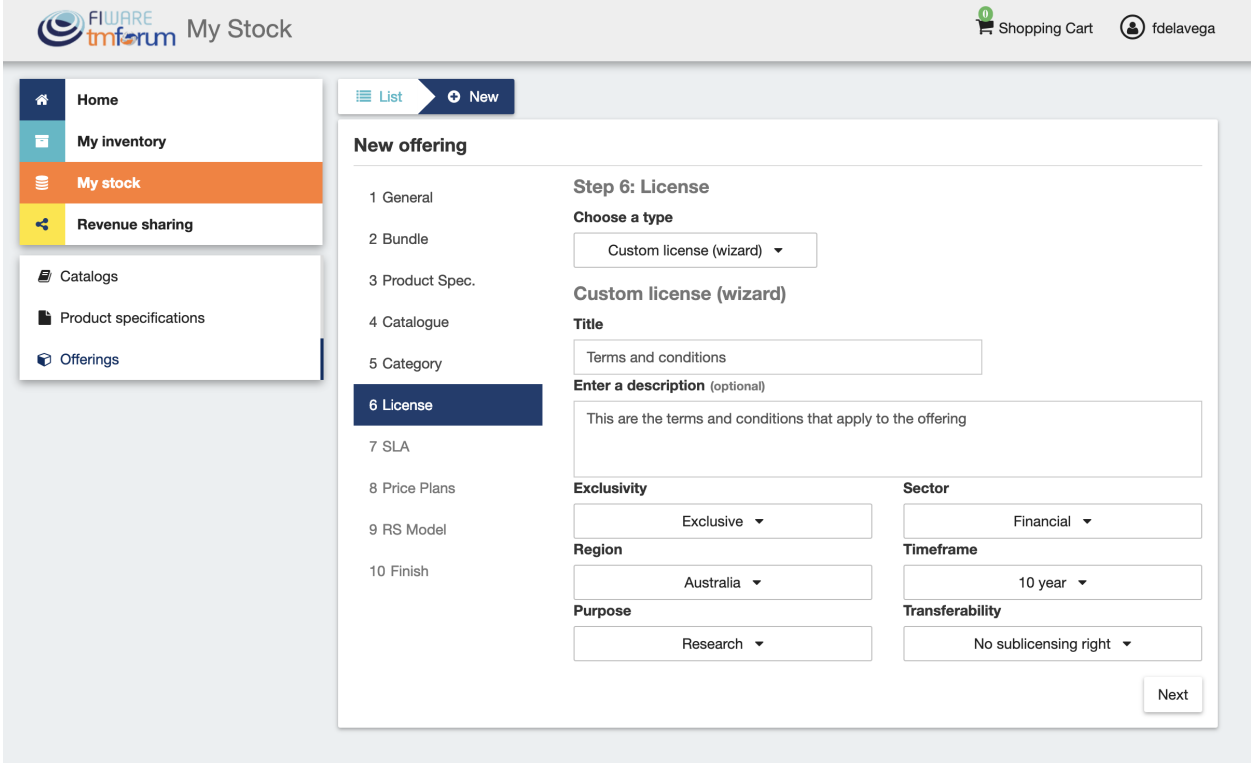


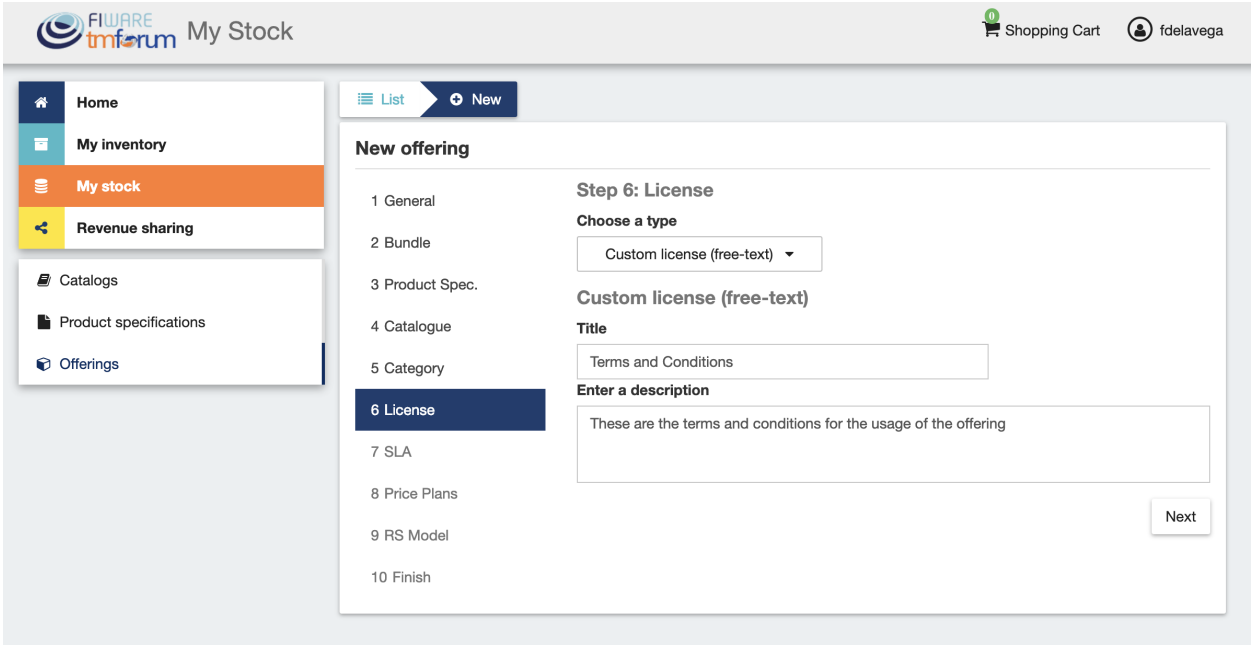

In the next step, you can optionally choose categories for you offering. Once done, click on *Next*



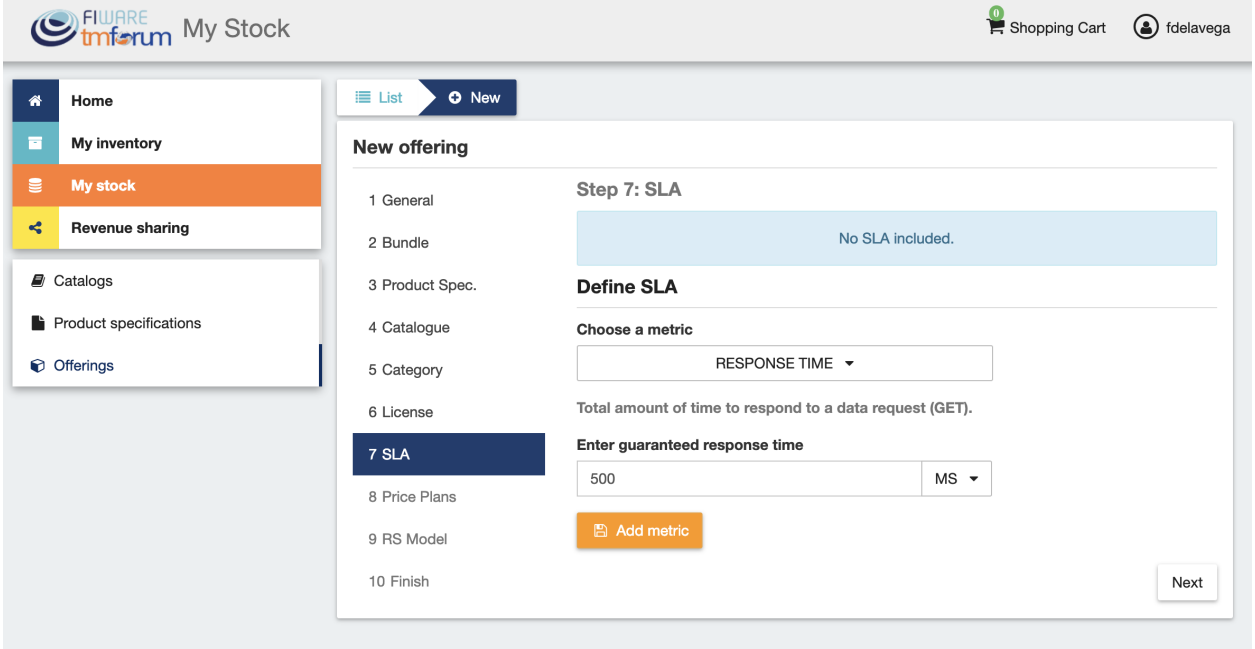

Next, it is possible to include the License or terms and conditions to be applied to the offering being created. There are three different options for privinding such information: (1) For data, there is a set of standard open data licenses that can be chosen, (2) providing custom terms and conditions using a wizzard, and (3) providing terms and conditions providing free text.



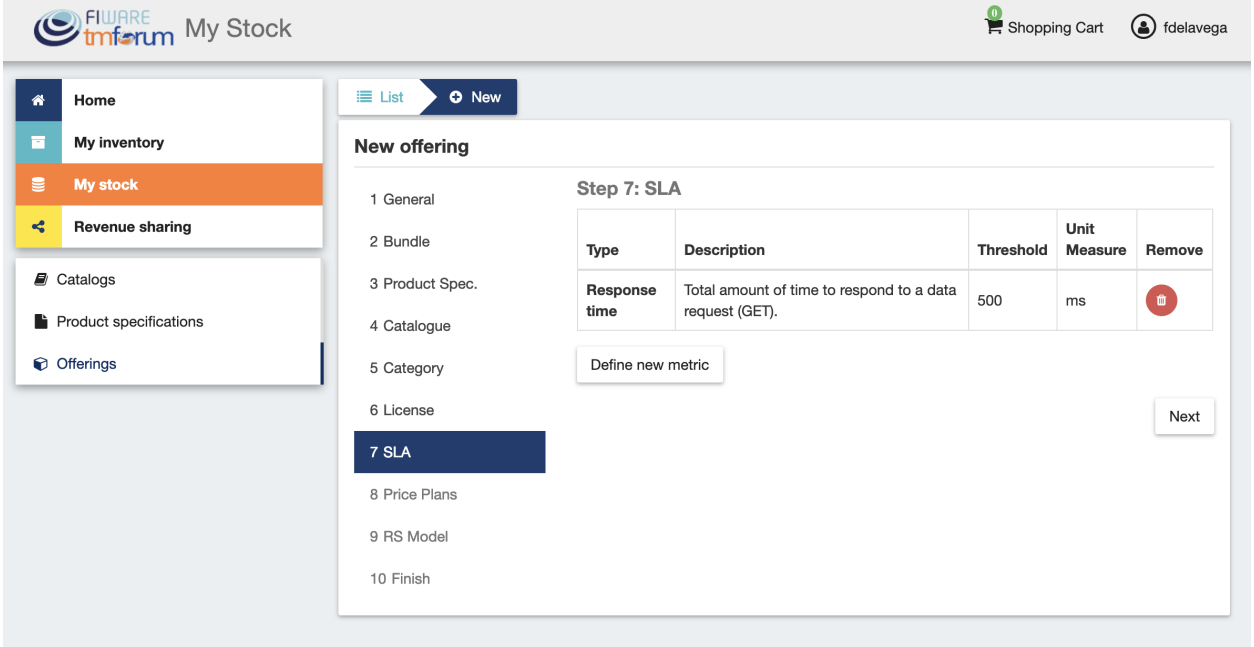



It is possible to include some SLA information attached to the offering in the step of the form. To do that, click on *Define new metric* button. In the displayed form. choose a metric, provide its value and click on *Add metric*.

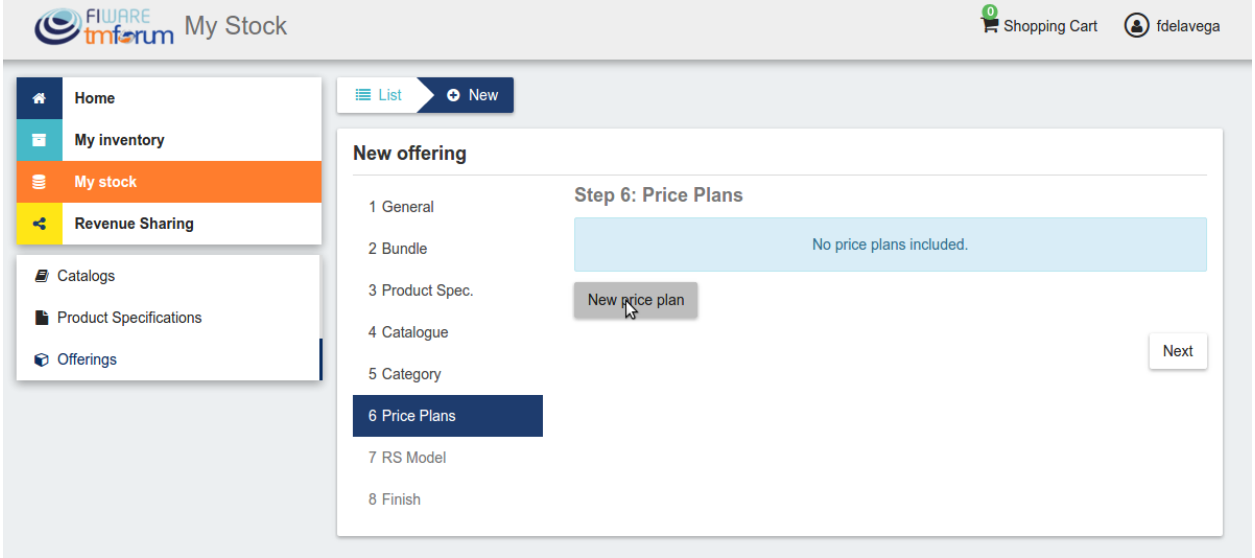



Once all the metrics have been provided click on *Next*




The next step is the most 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 it is considered free.

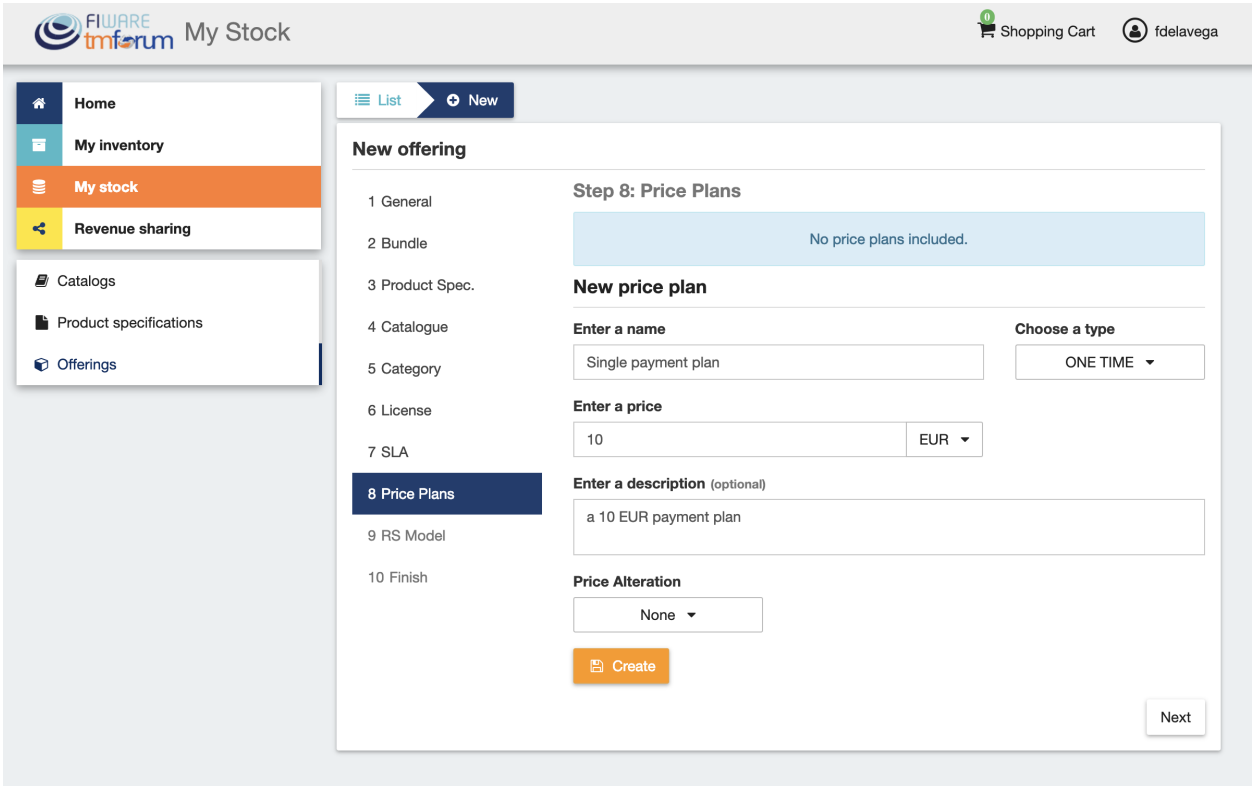
To include a new price plan the first step is clicking on *New Price Plan*



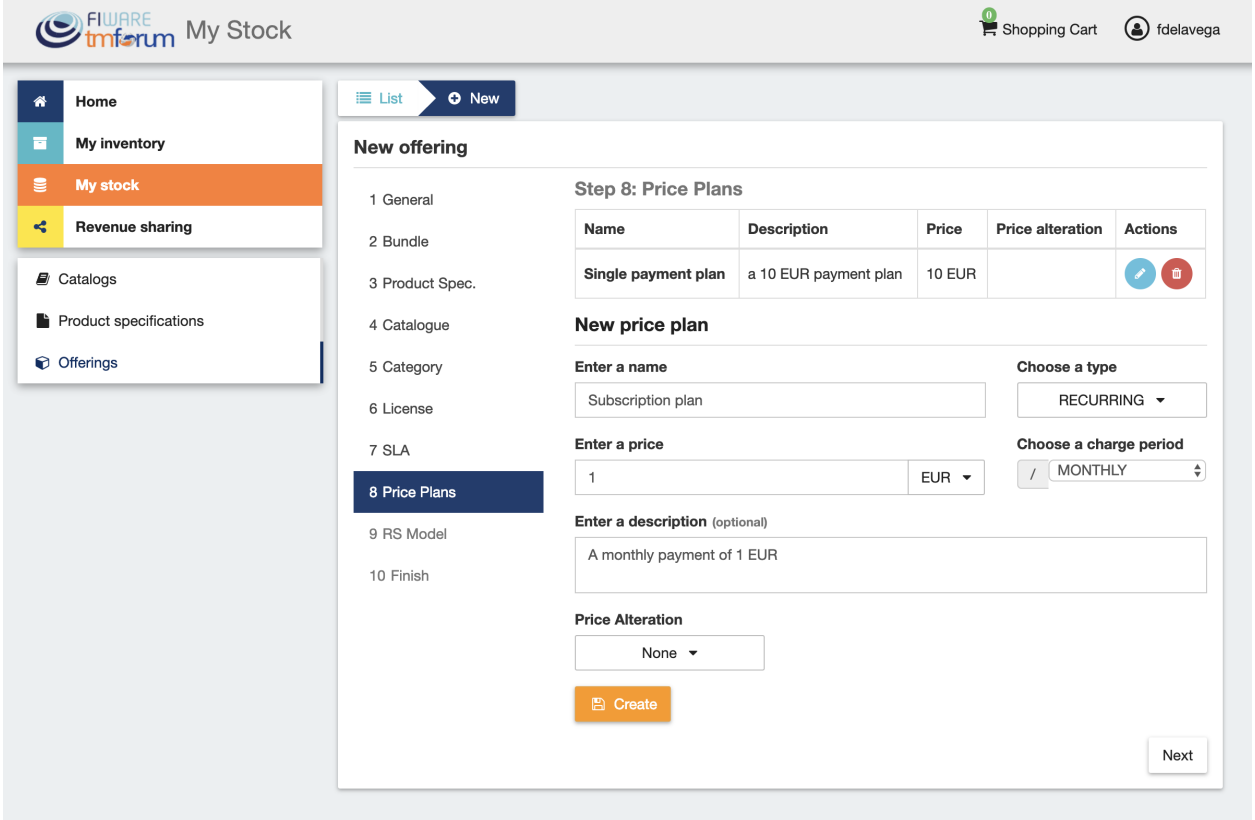

For creating the price plan, you have to provide a name, and an optional description. Then, you have to choose the type of price plan between the provided options.

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.

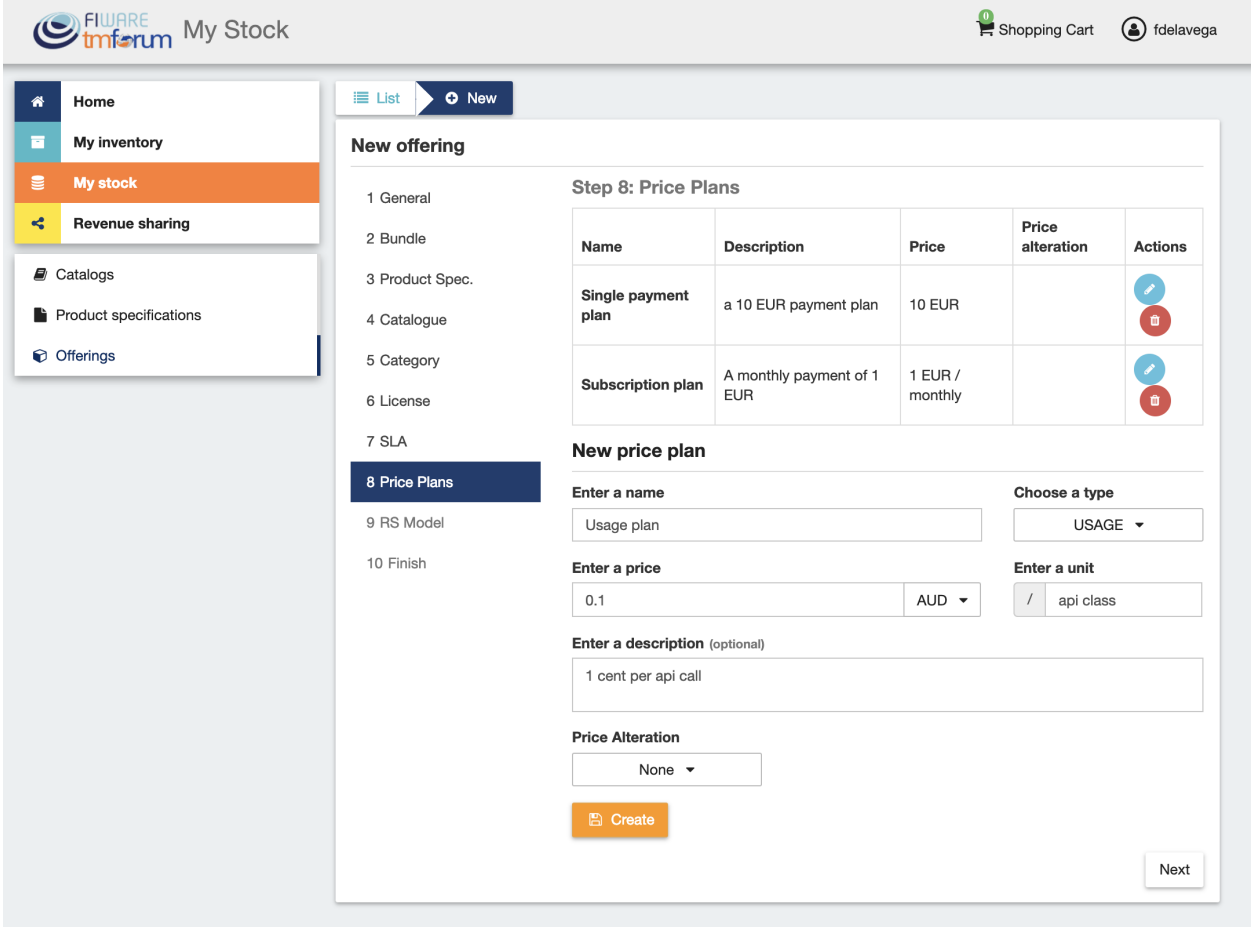
If you choose *one time*, you have to provide the price and the currency.



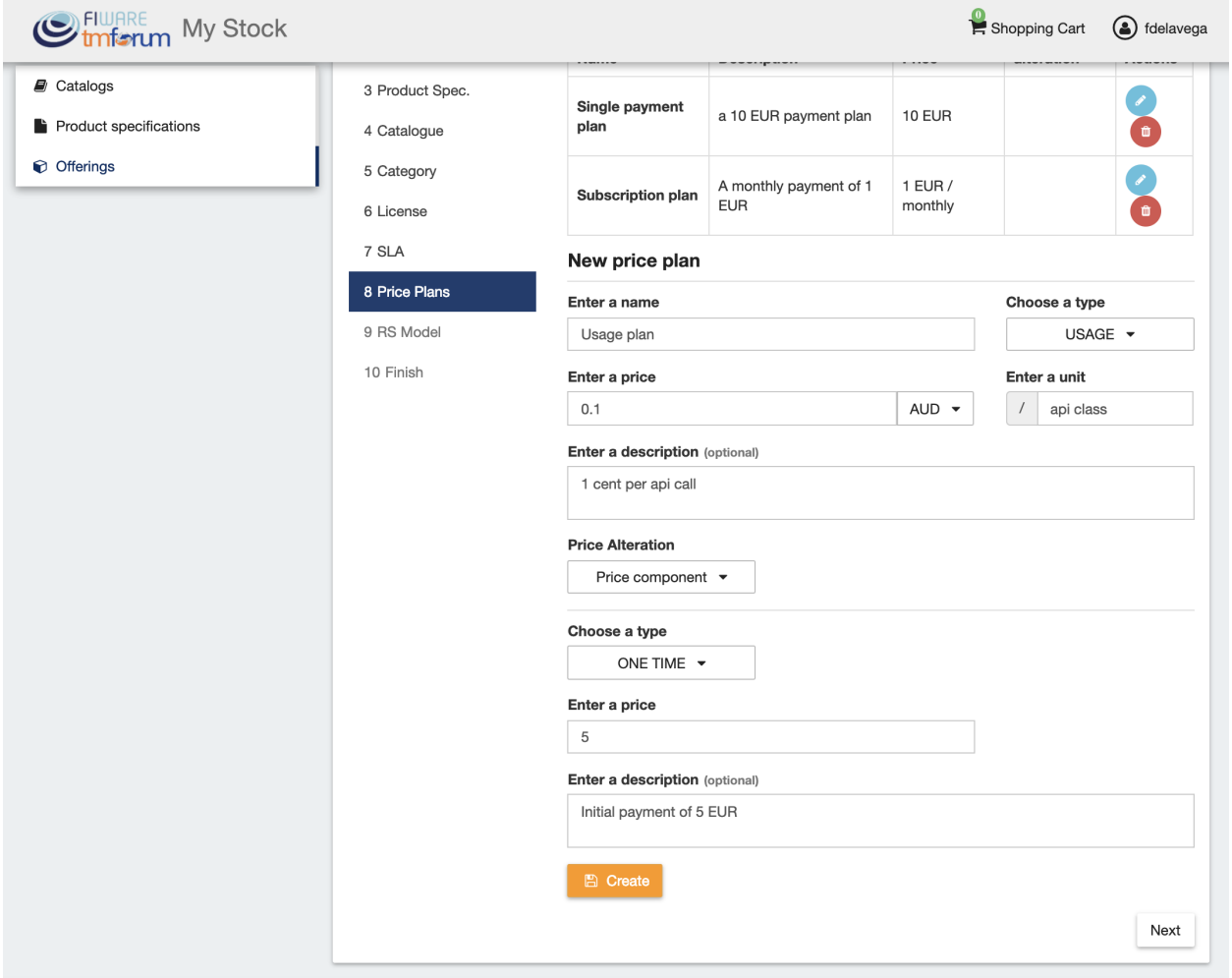

If you choose *recurring*, you have to provide the price, the currency, and the period between charges.

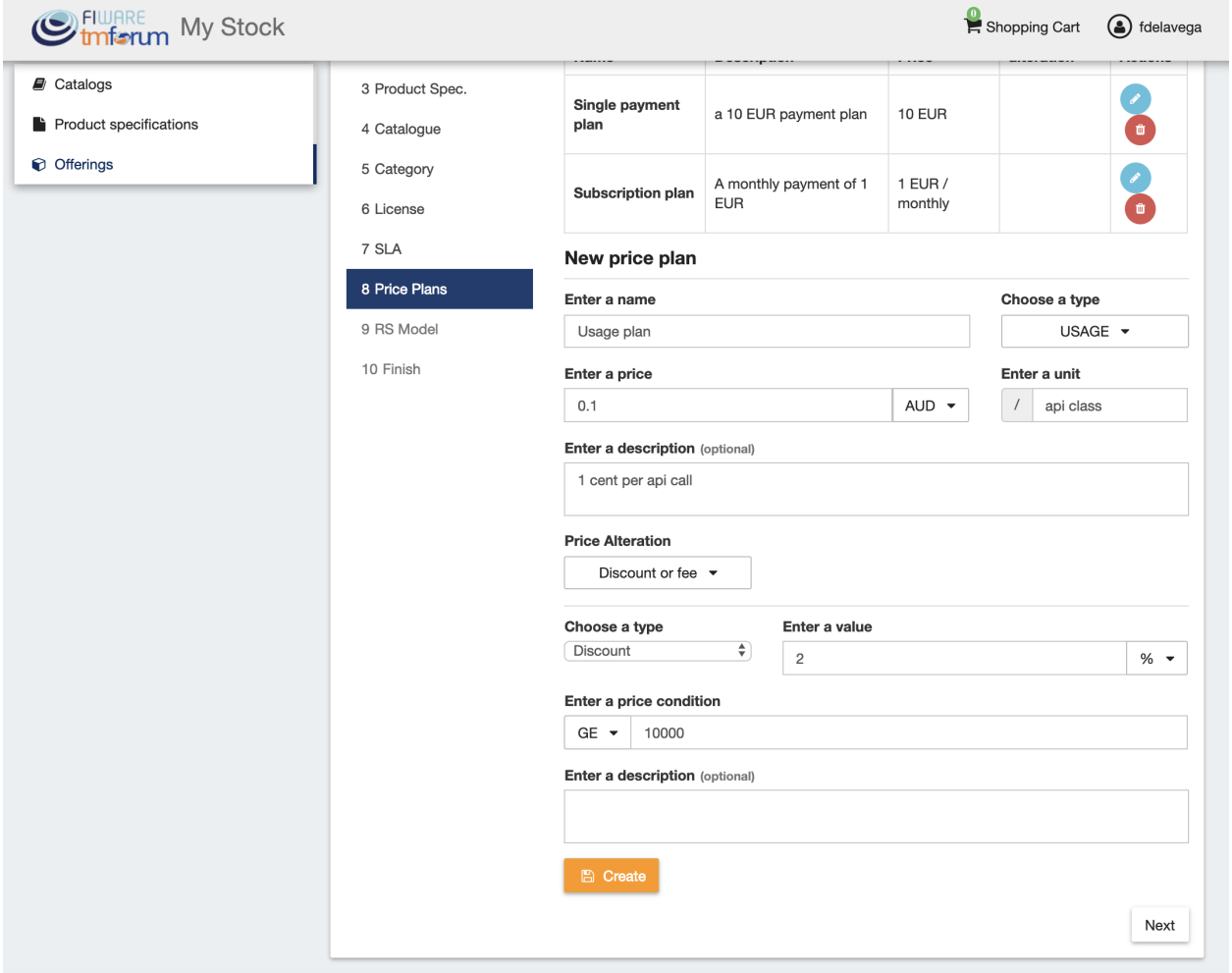



If you choose usage, you have to provide the unit to be accounted, the currency, and the price per unit

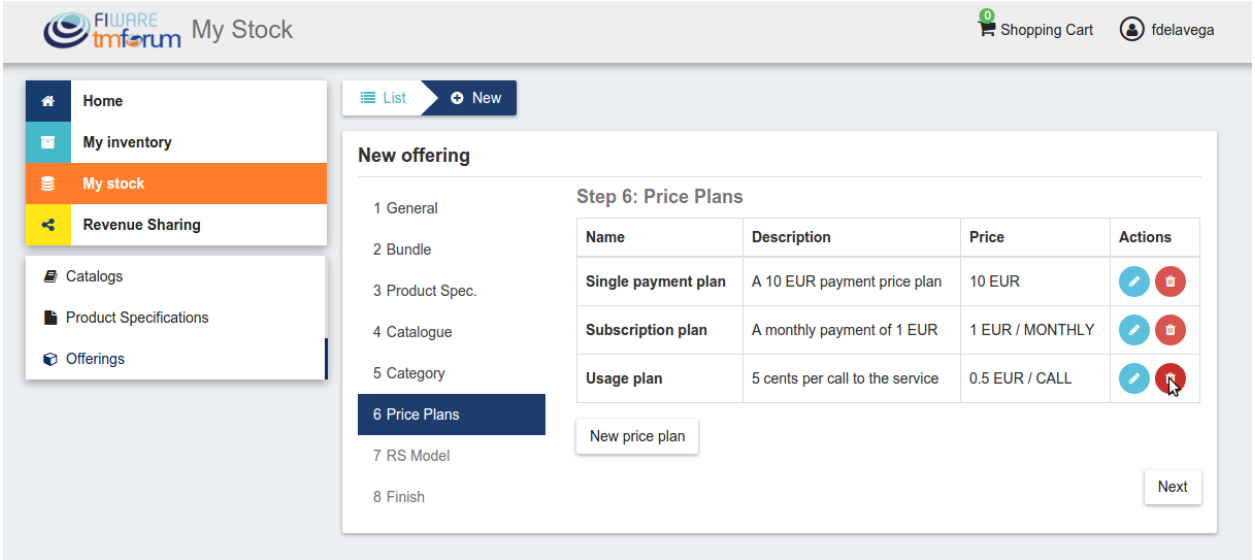



In addition to the basic pricing models it is possible to include price alterations using the *Price Alteration* section. In this regard, it is possible to provide two types of alterations: (1) Price components, enable to extend the model with a complementary pricing (e.g an initial or recurring fixed payment in a usage model). (2) fees and discounts, which are applied to the original model when some condition is satisfied (e.g a 2% discount when more that 10k calls has been made)

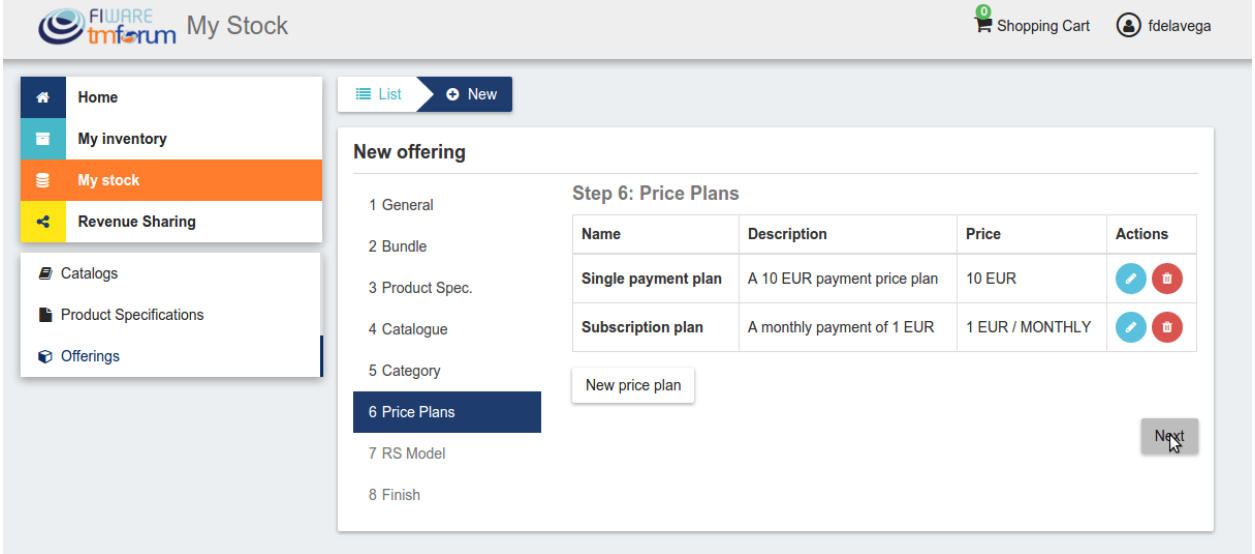

FIWARE tmforum My Stock
Shopping Cart
fdelavega
Catalogs
Product specifications
Offerings
3 Product Spec.
4 Catalogue
5 Category
6 License
7 SLA
8 Price Plans
9 RS Model
10 Finish
Single payment plan
a 10 EUR payment plan
10 EUR
Subscription plan
A monthly payment of 1 EUR
1 EUR / monthly
New price plan
Enter a name
Usage plan
Choose a type
USAGE
Enter a price
0.1
AUD
Enter a unit
/
api class
Enter a description (optional)
1 cent per api call
Price Alteration
Price component
Choose a type
ONE TIME
Enter a price
5
Enter a description (optional)
Initial payment of 5 EUR
Create
Next



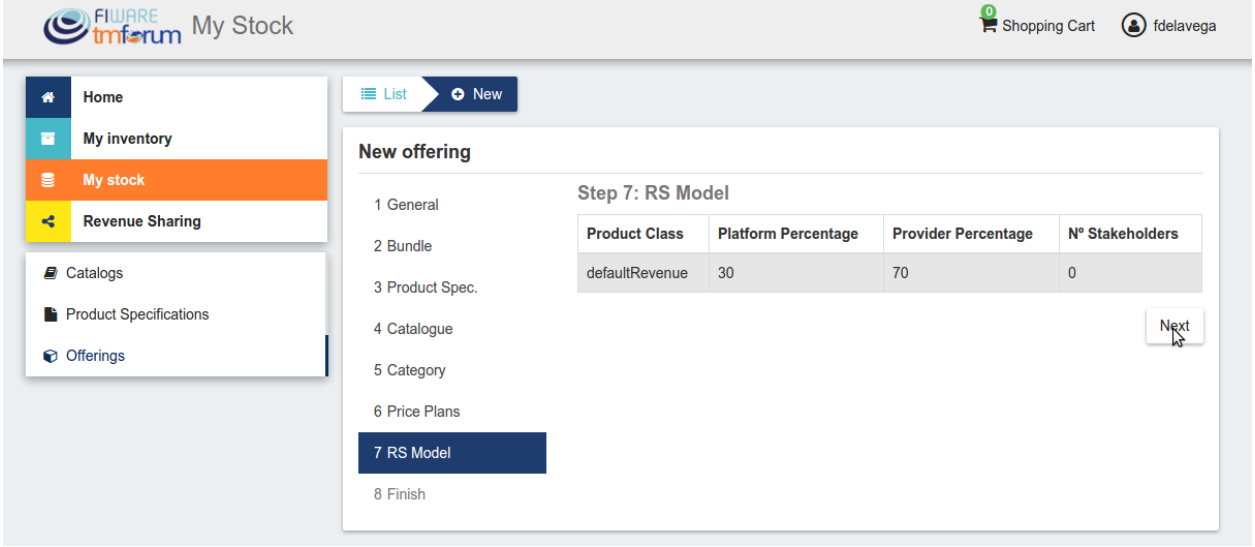

You can update or remove plans by clicking on the corresponding action button.

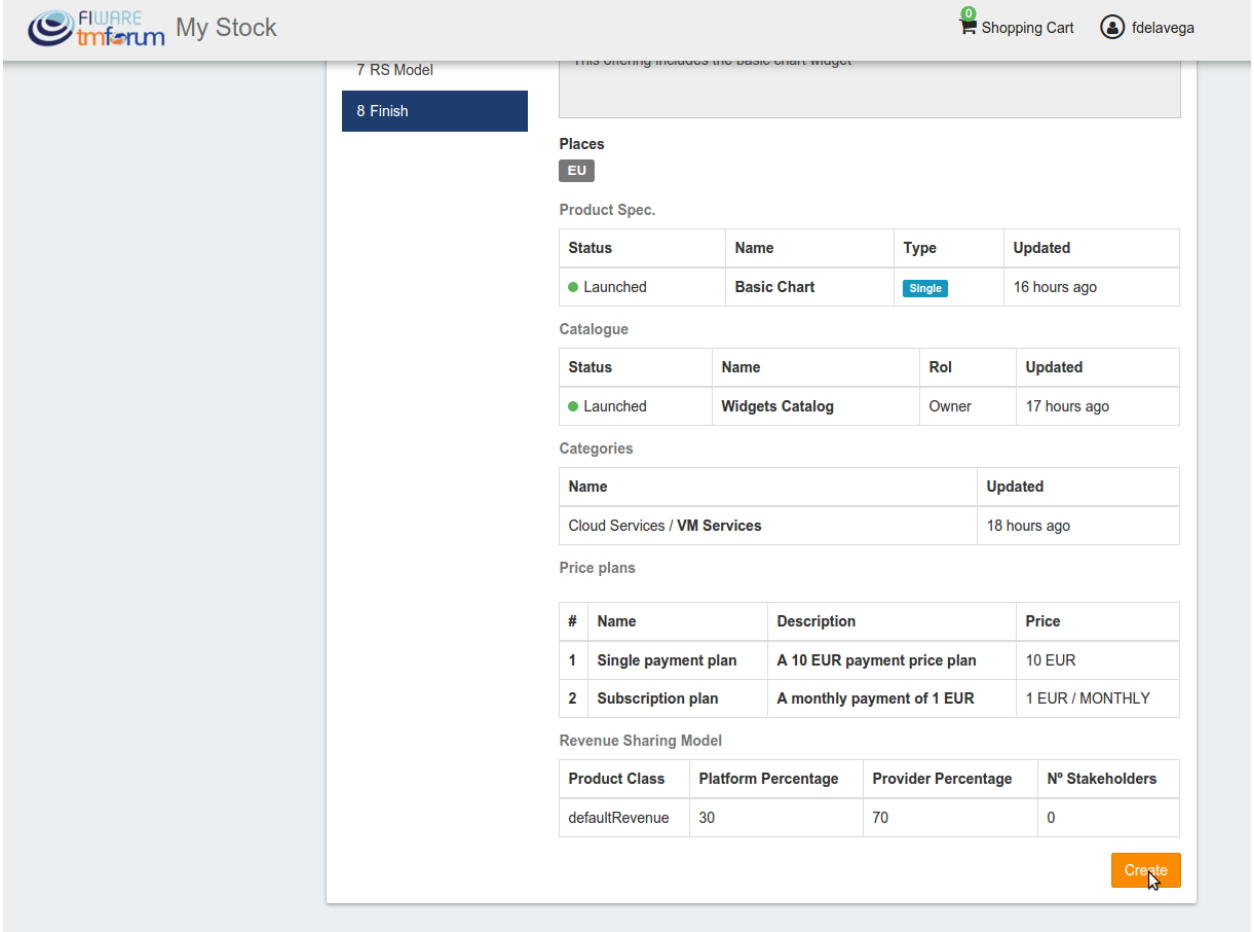



Once you have created you pricing model click on *Next*

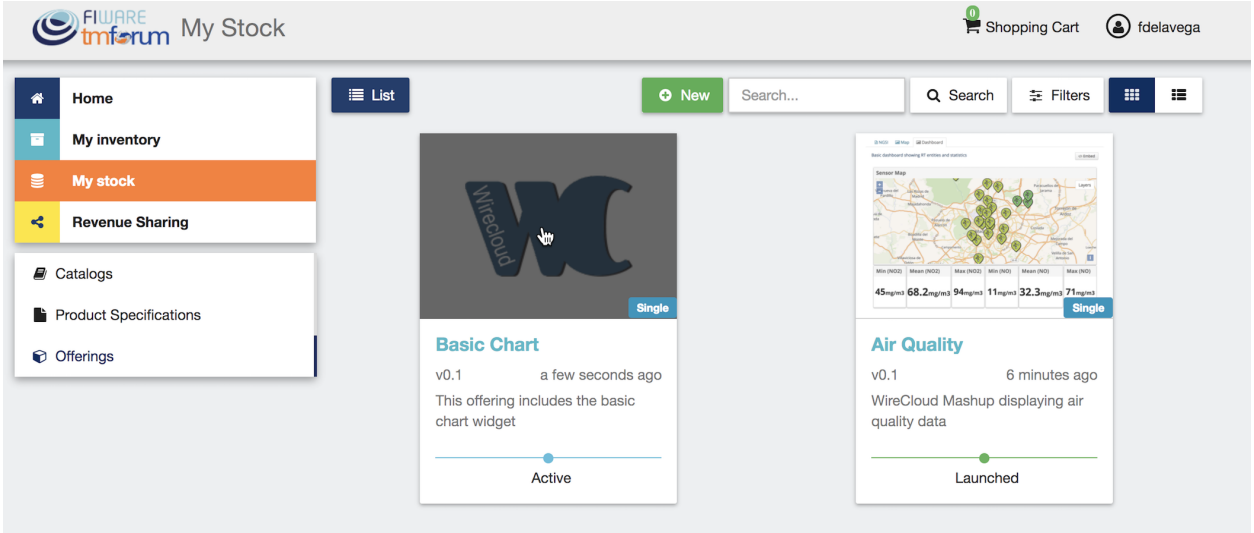



In the last step of the process, you have to choose the revenue sharing model to be applied to you offering between the available ones. Once done, click on *Next* and then on *Create*.

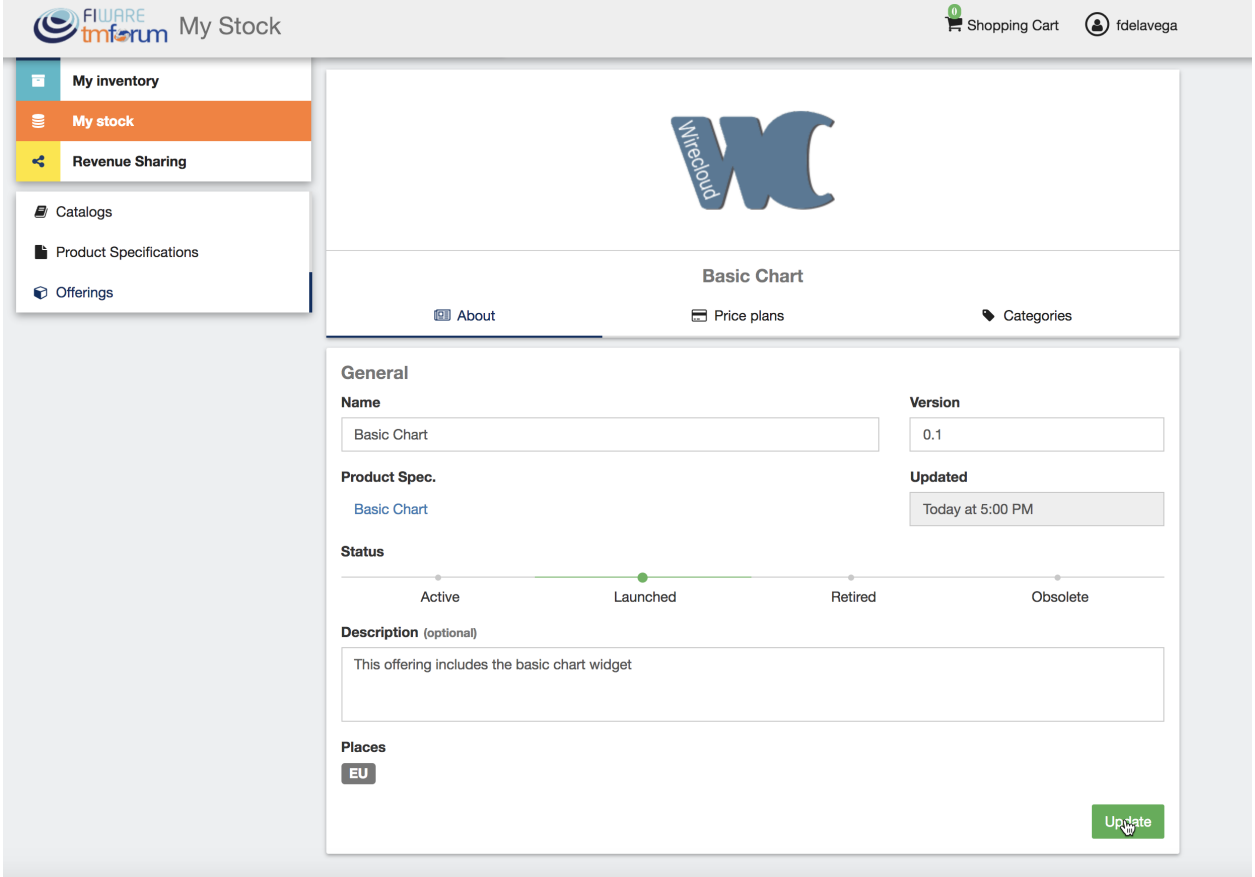




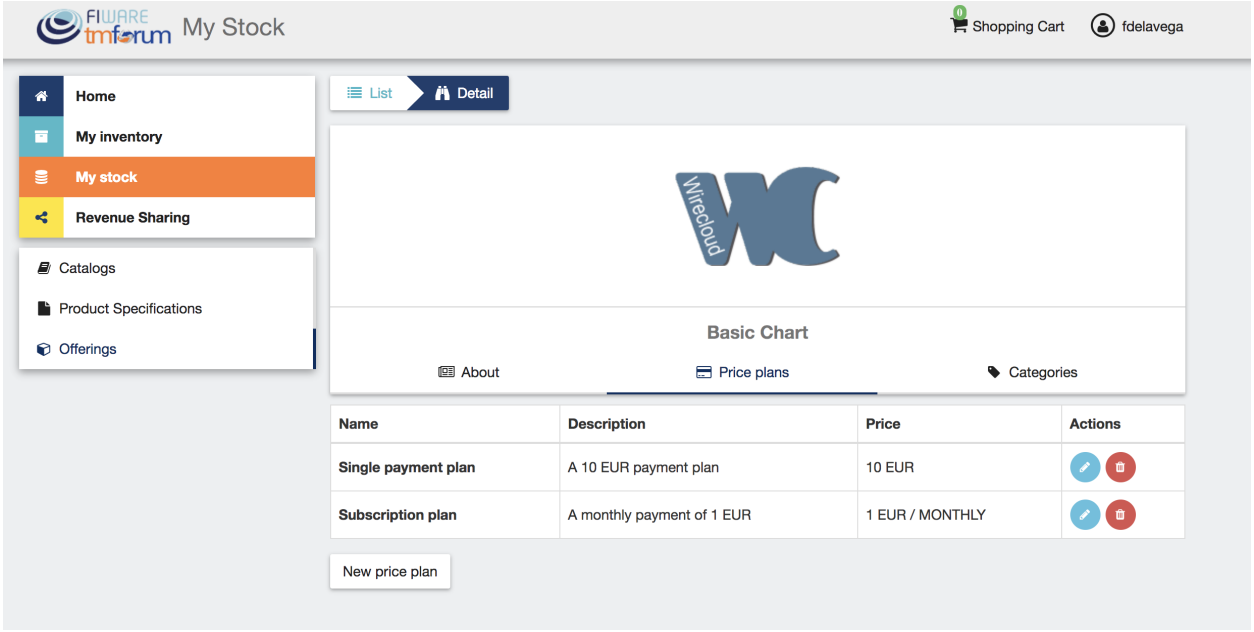

Sellers can also edit their offerings. To do that click on the offering to be updated.

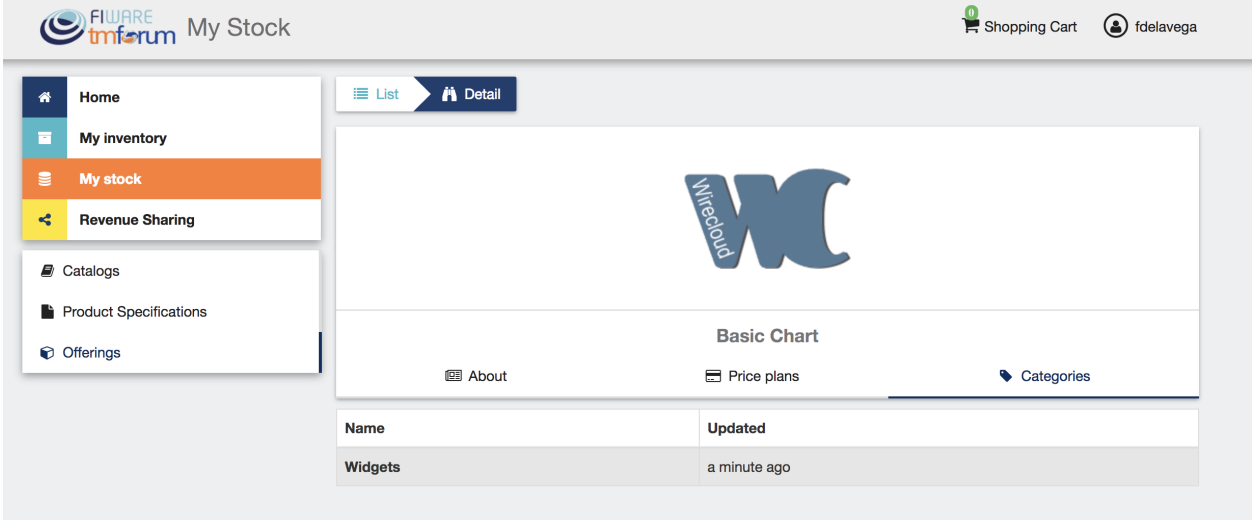



In the displayed form, change the fields you want to edit and click on *Update*. Note that for start selling you offering you have to update its status to *Launched*




It is also possible to update the *Price Plans* and *Categories* of the offering by accessing to the related tab.

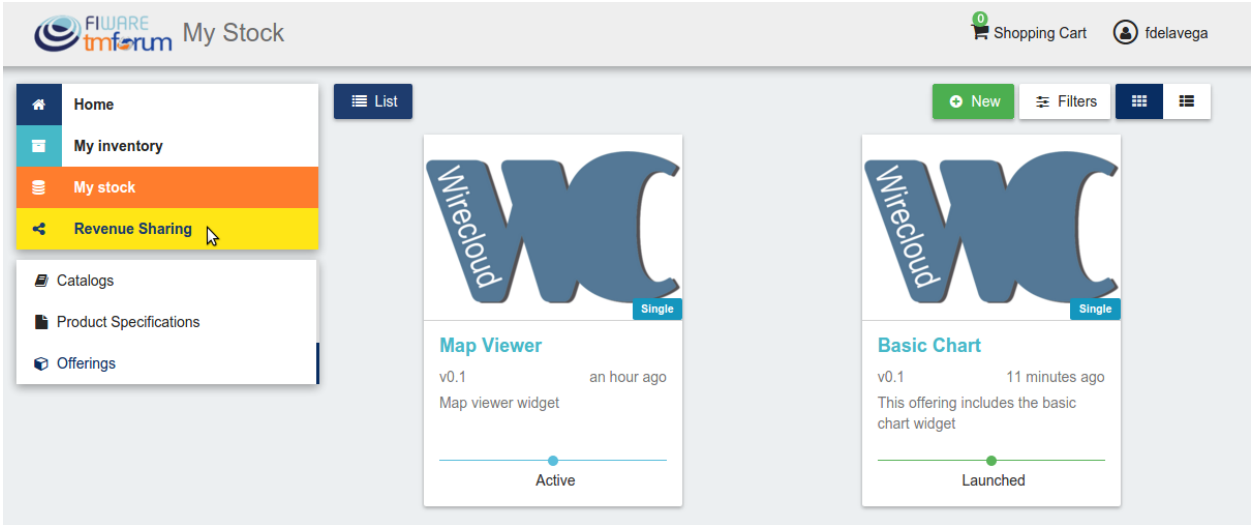




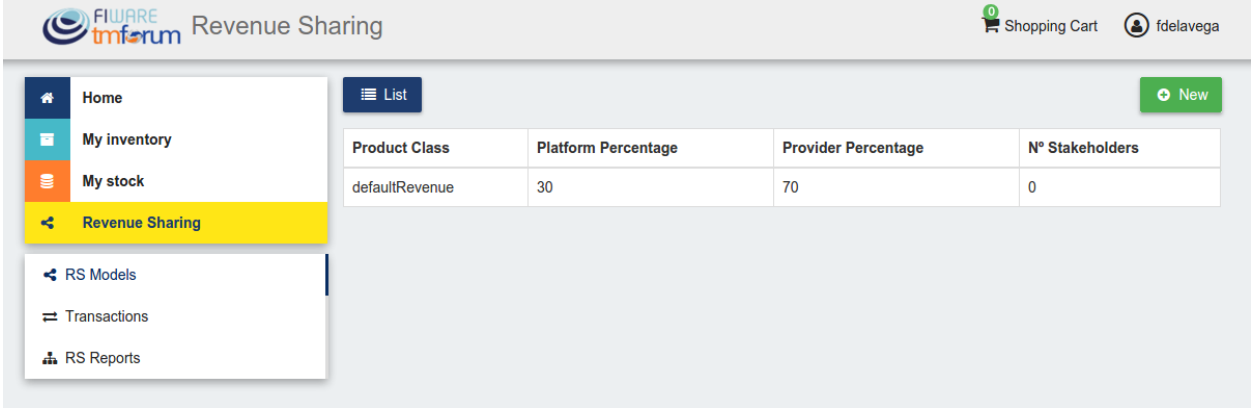

## Manage Revenue Sharing Models

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.

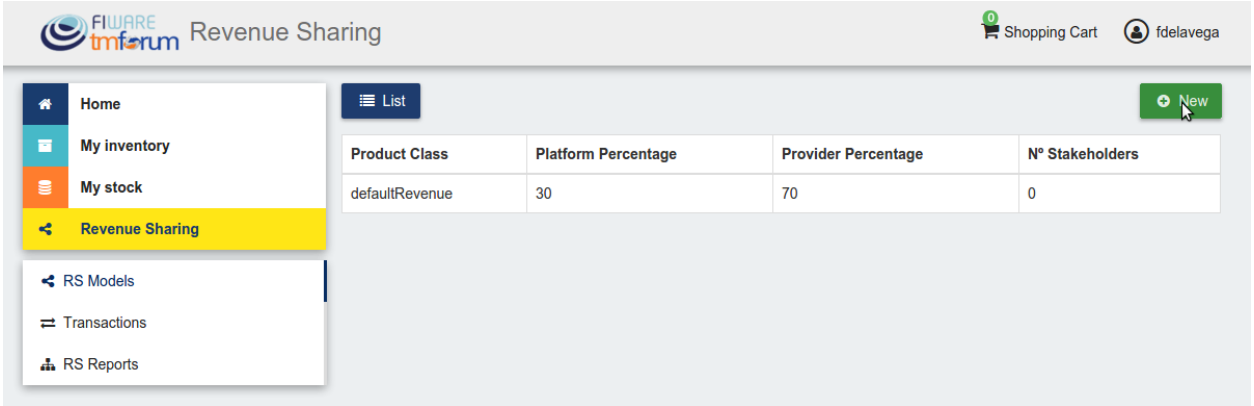
To manage RS models go to the *Revenue Sharing* section.



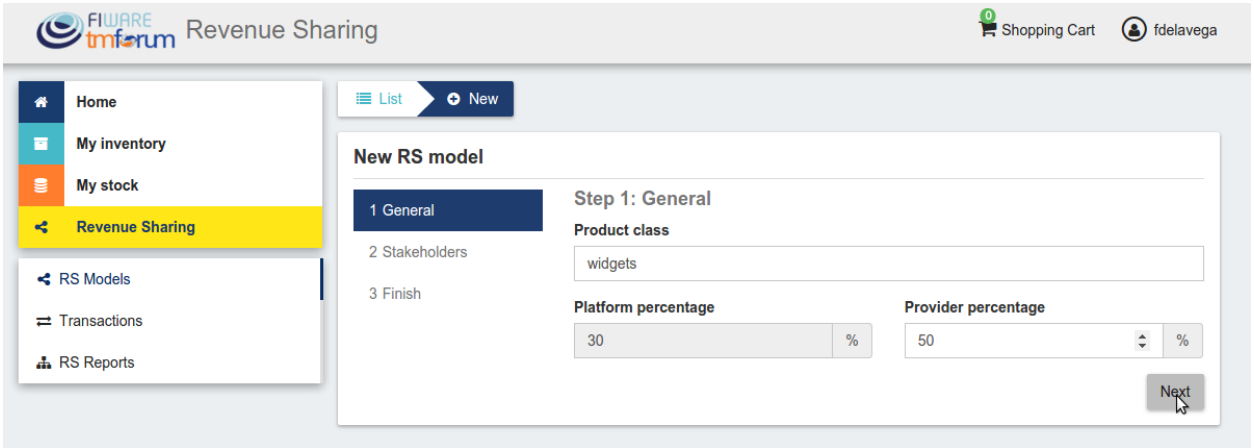

In this view, you can see the revenue sharing models you have available. By default it will appear the default RS model which establishes the revenue distribution between you and the Business API Ecosystem instance owner.



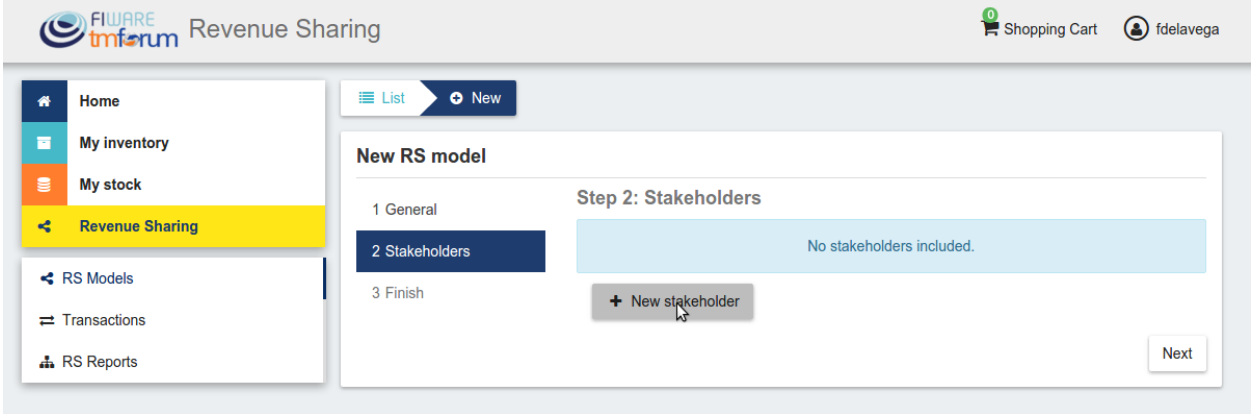

You can create a new RS model clicking on *New*



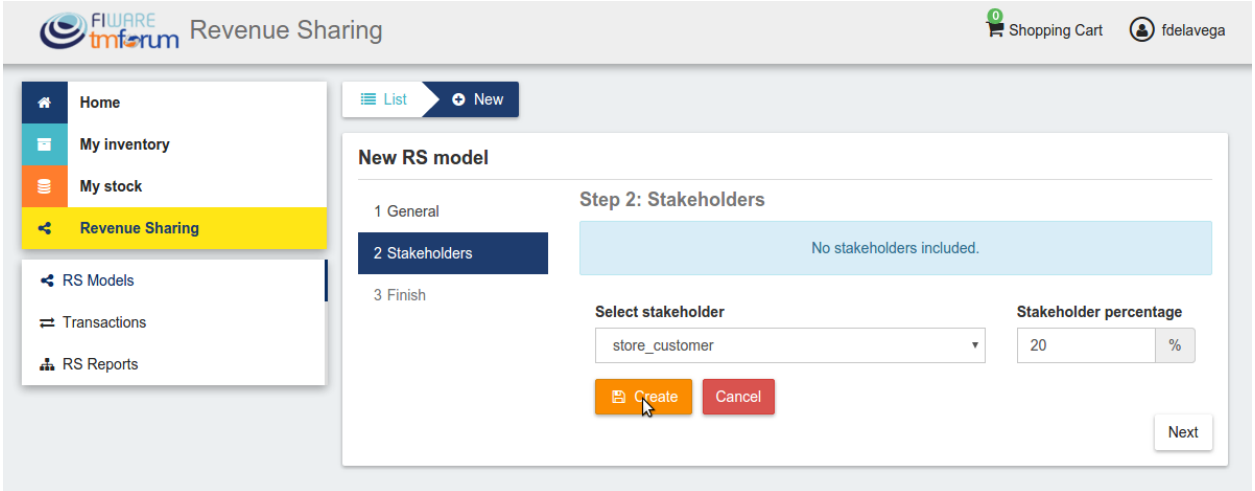

In the first step of the process you have to provide a product class, which identifies the RS model, and the percentage you want to receive. The platform percentage is fixed and cannot be modified. Once provided click on *Next*




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

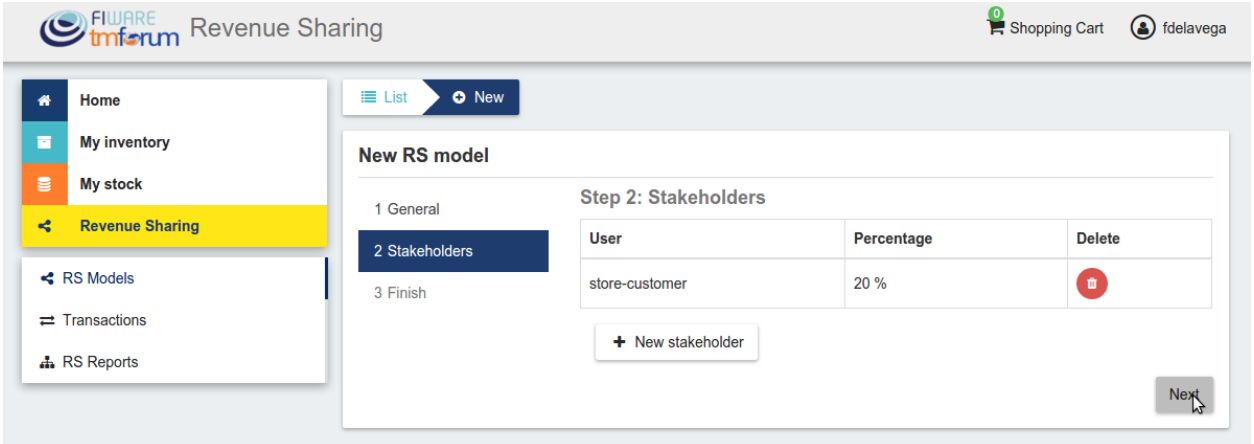


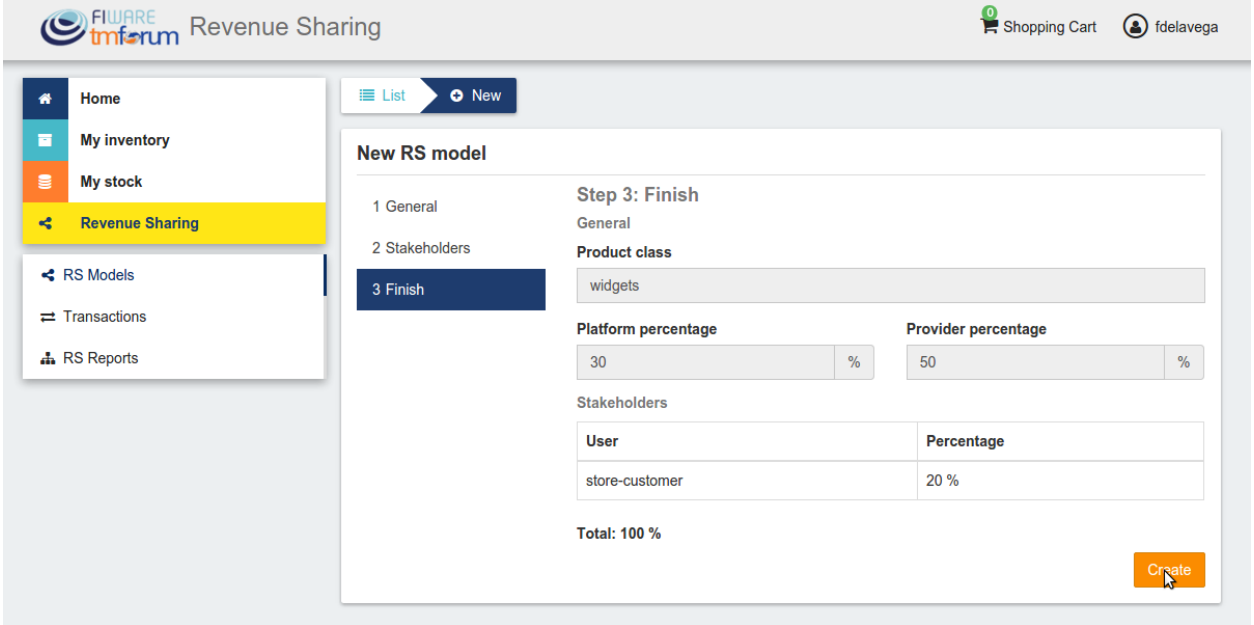

Then, select the Stakeholder between the available users, and provide its percentage. Finally, save it clicking on *Create*



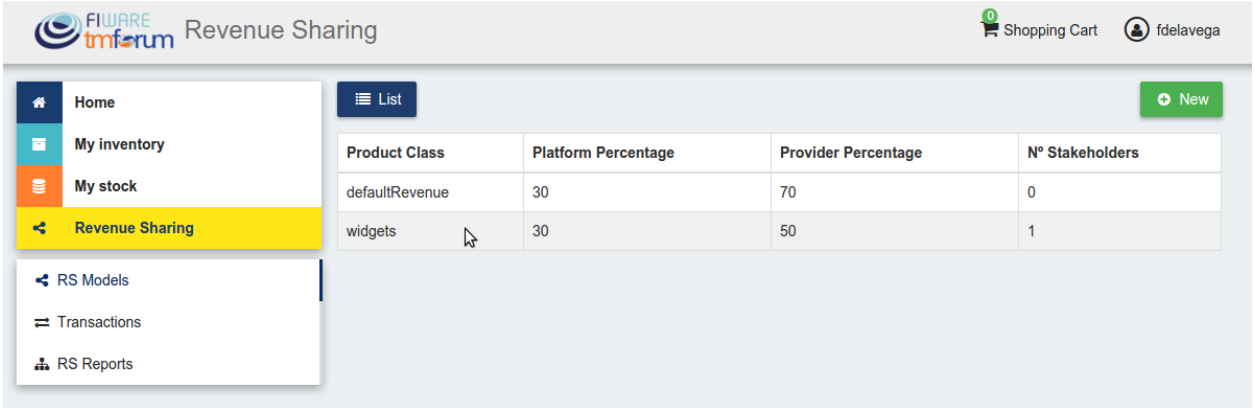

**Note:** The total percentage (provider + platform + stakeholders) must be equal to 100

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

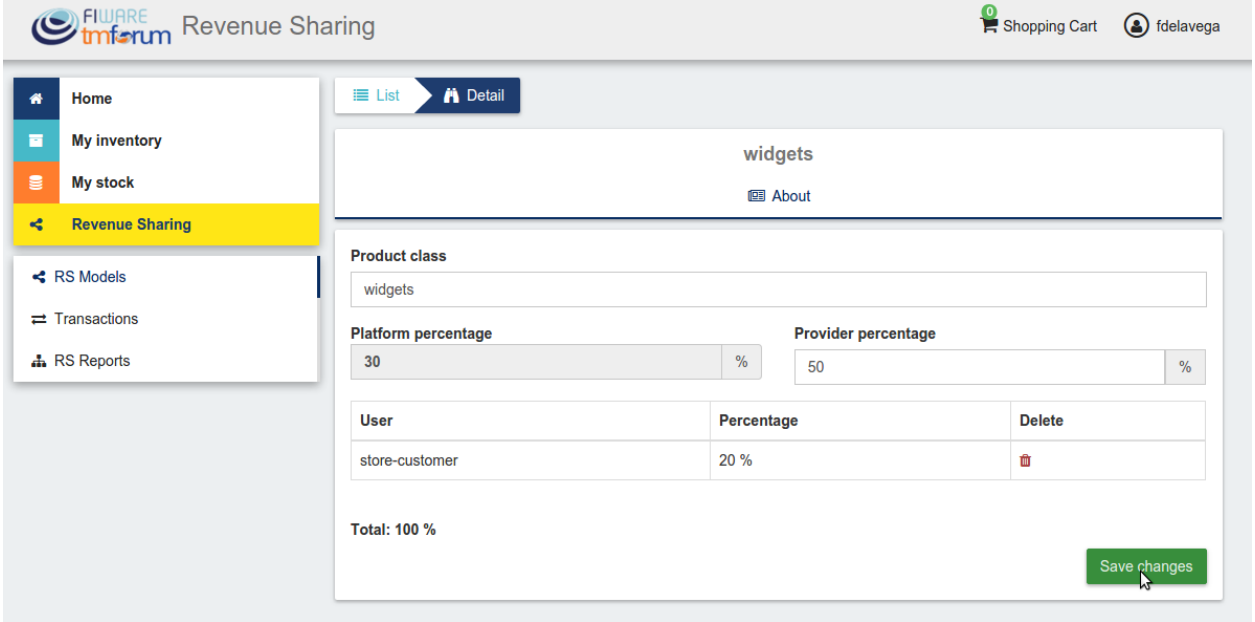





Sellers can also update their RS model. To do that click on the RS model to be updated.

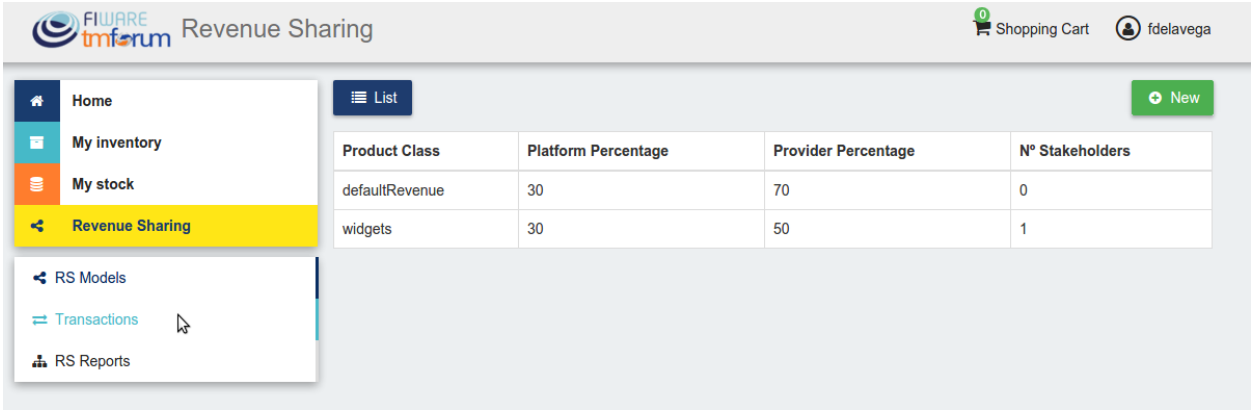



Then, update the required fields (including the stakeholders if you want), and click on *Save Changes*

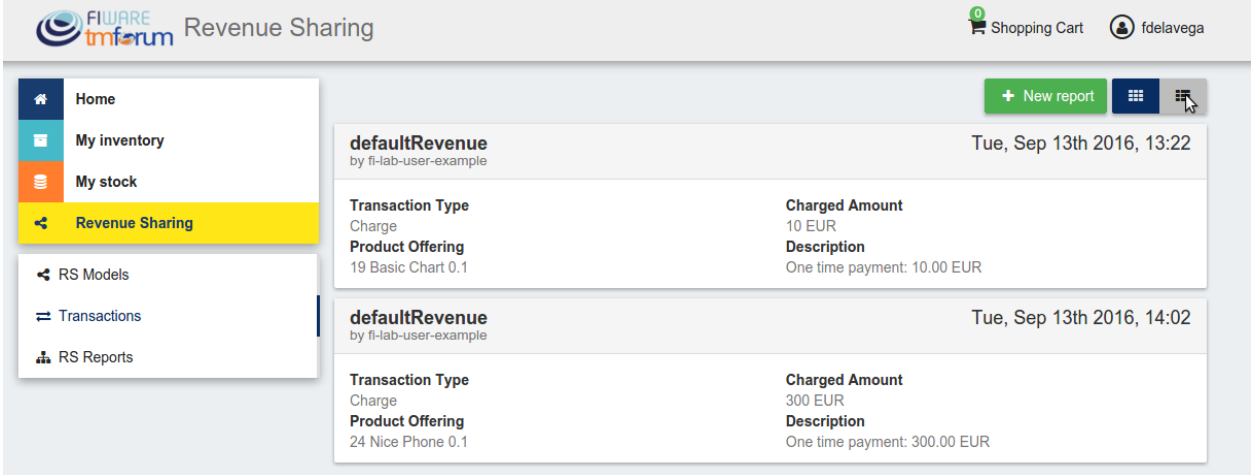


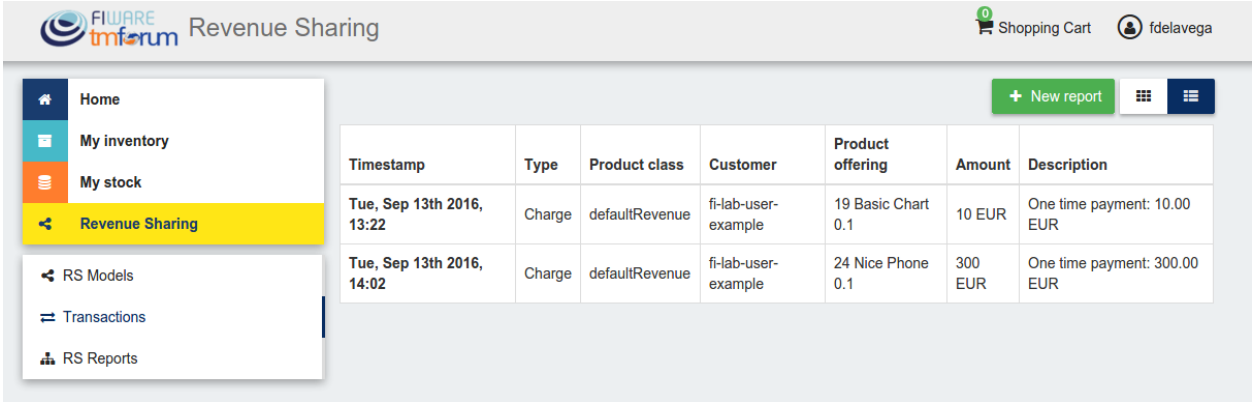

## Manage Transactions

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*



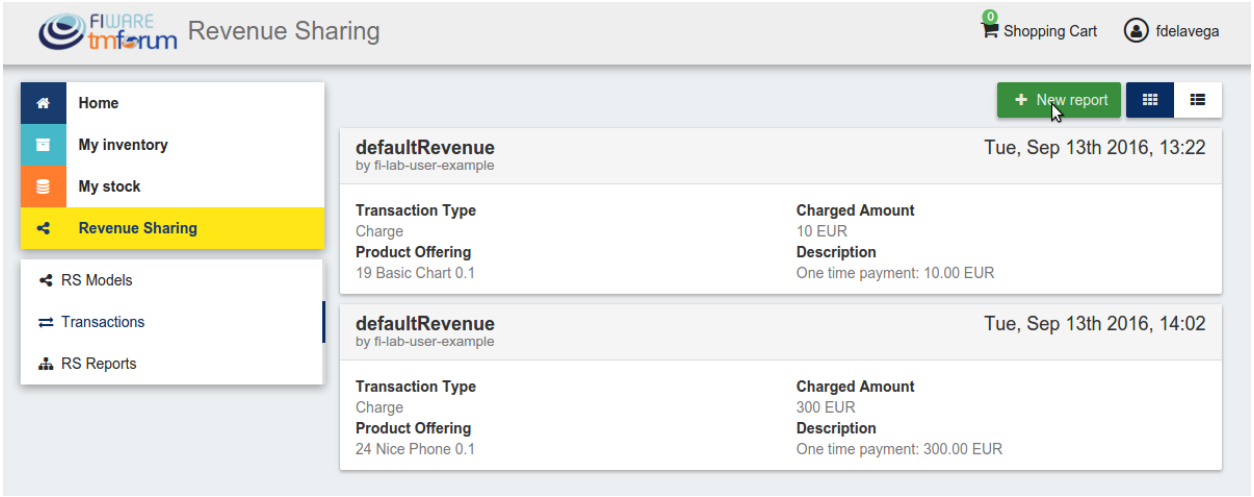

In the displayed view, you can see the transactions pending to be paid to you and your stakeholders. It is also possible to display the transactions in tabular way



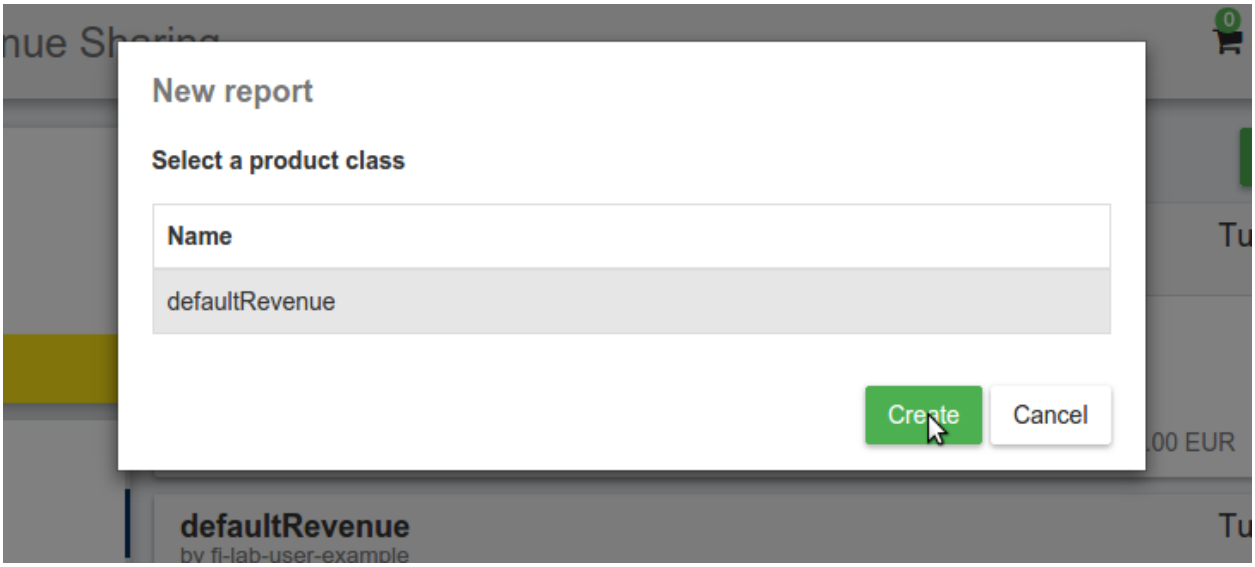




These transactions are aggregated and paid by the Business API Ecosystem periodically once a month. Nevertheless, if you need to be paid, you can force the revenue sharing calculus and payment of your pending transactions by manually generating a revenue sharing report.

To create a new report click on *New Report*

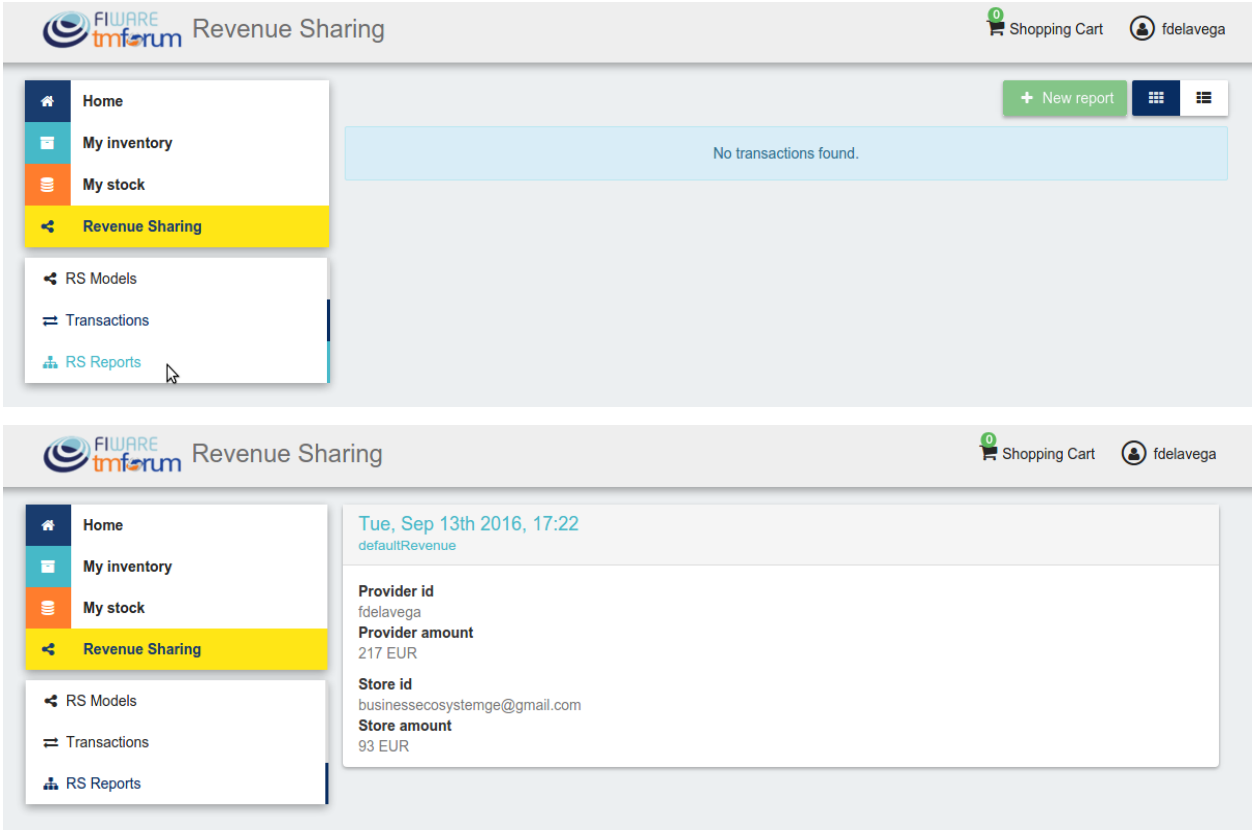



In the displayed modal, choose the product classes to be calculated and click on *Create*




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*

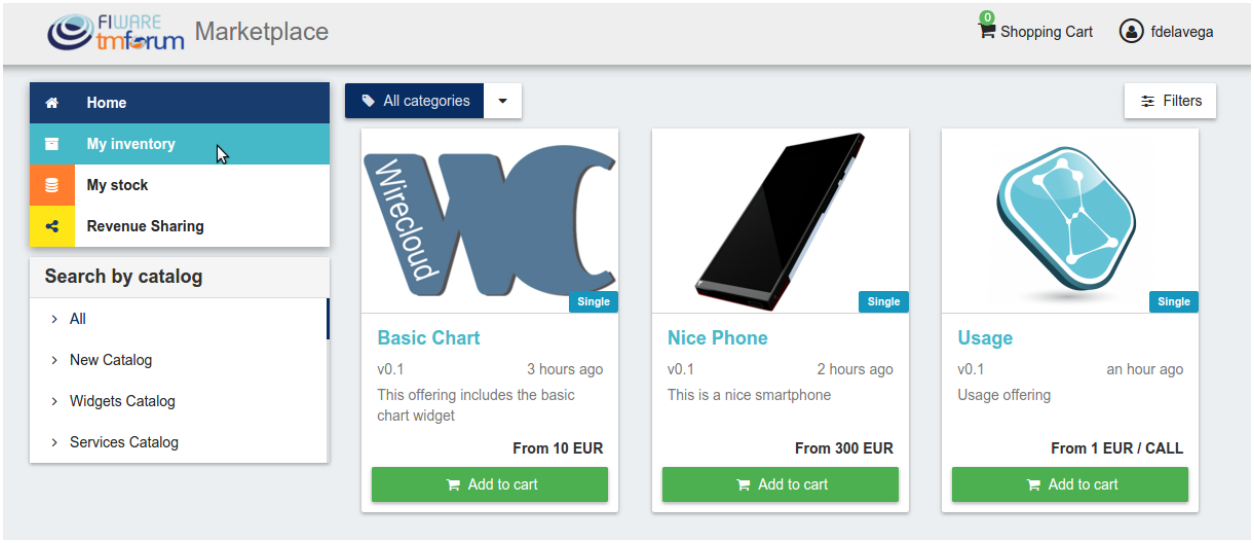


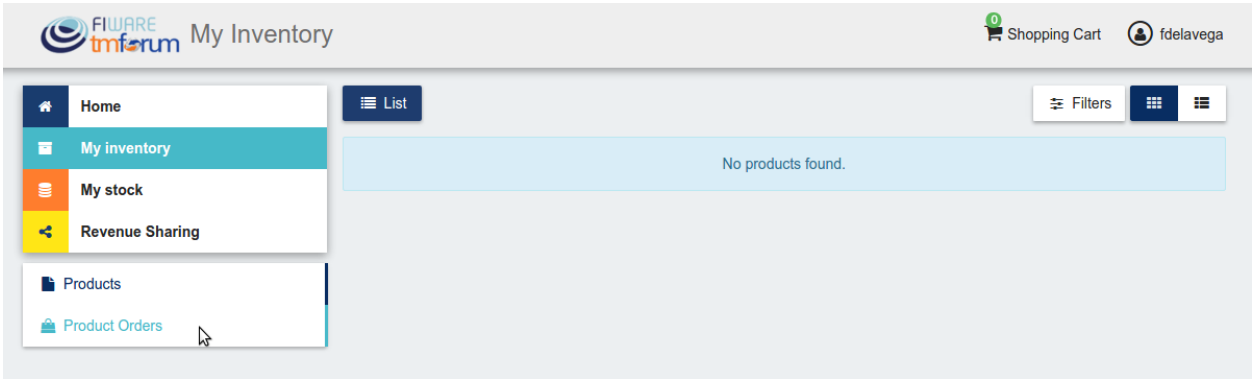

**Note:** Sellers would need to have a PayPal account associated to the email of their FIWARE IdM account in order to be paid for their products

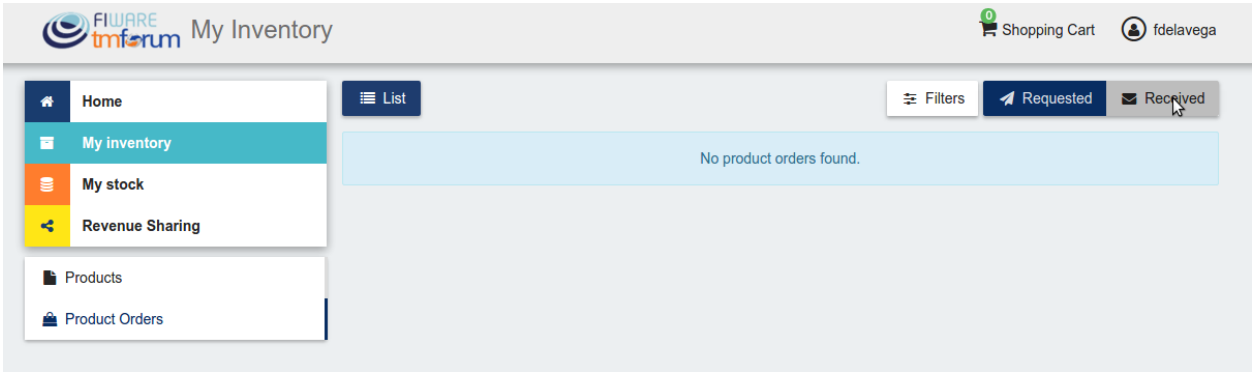
### Manage Received Orders

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.

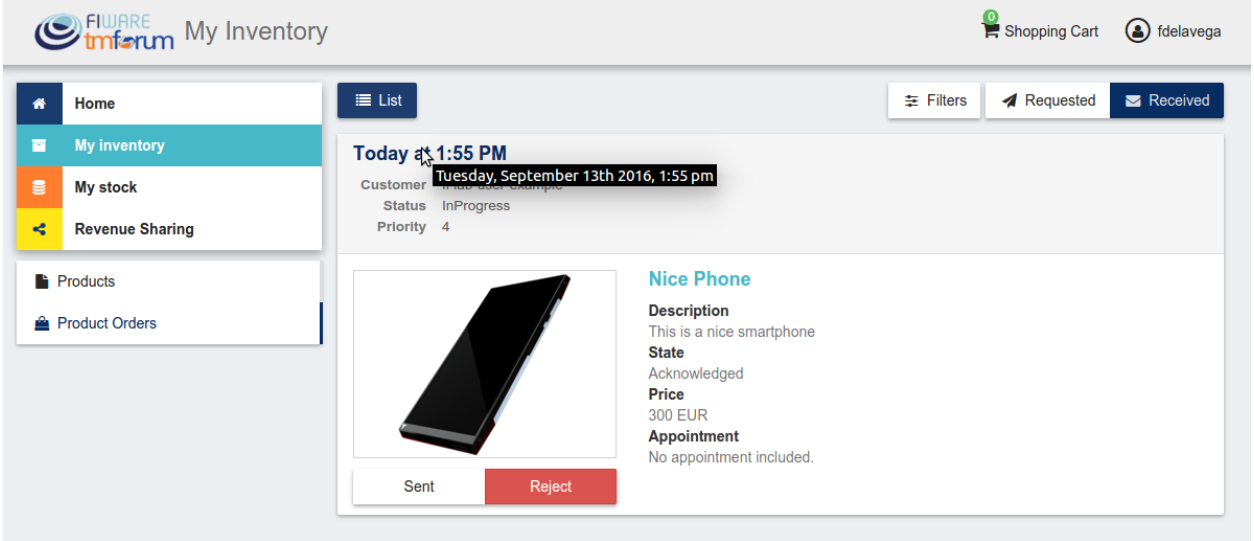
To view your received orders go to *My inventory* section, click on *Product orders*, and open the *Received* section.



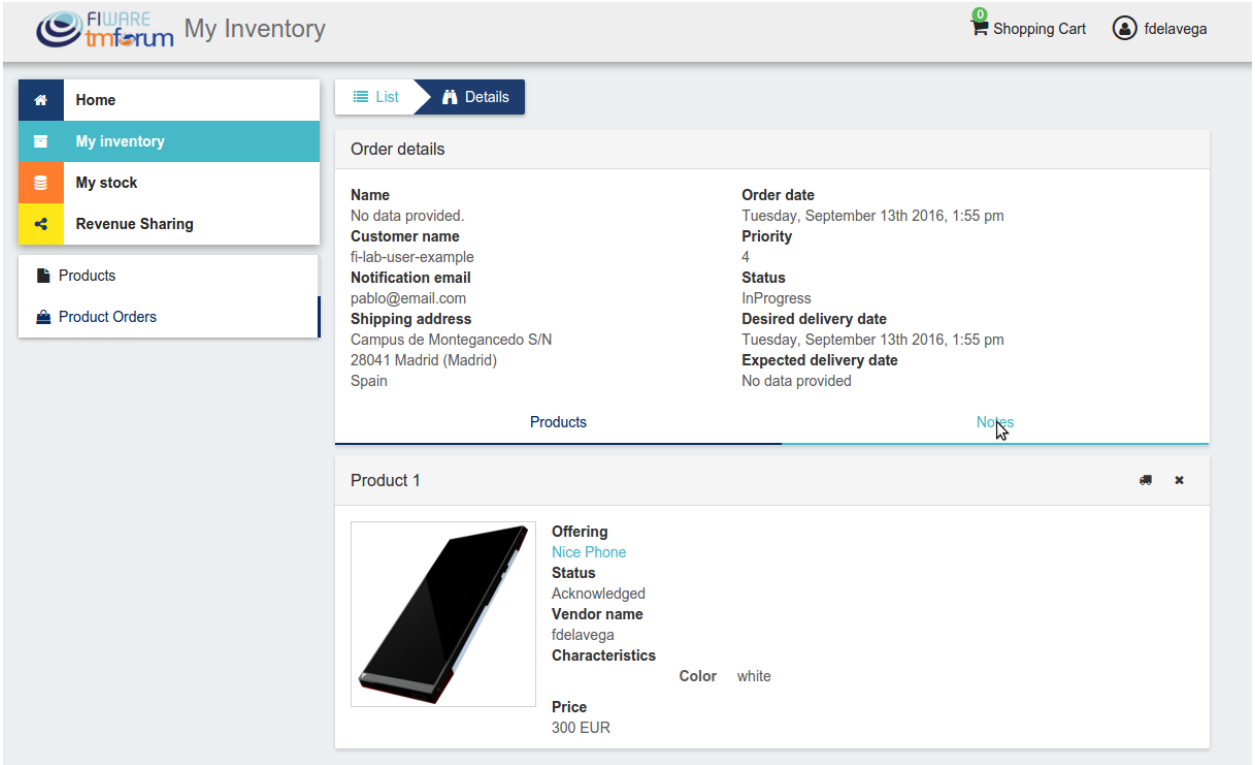




You can view the details of a received order clicking on the order date

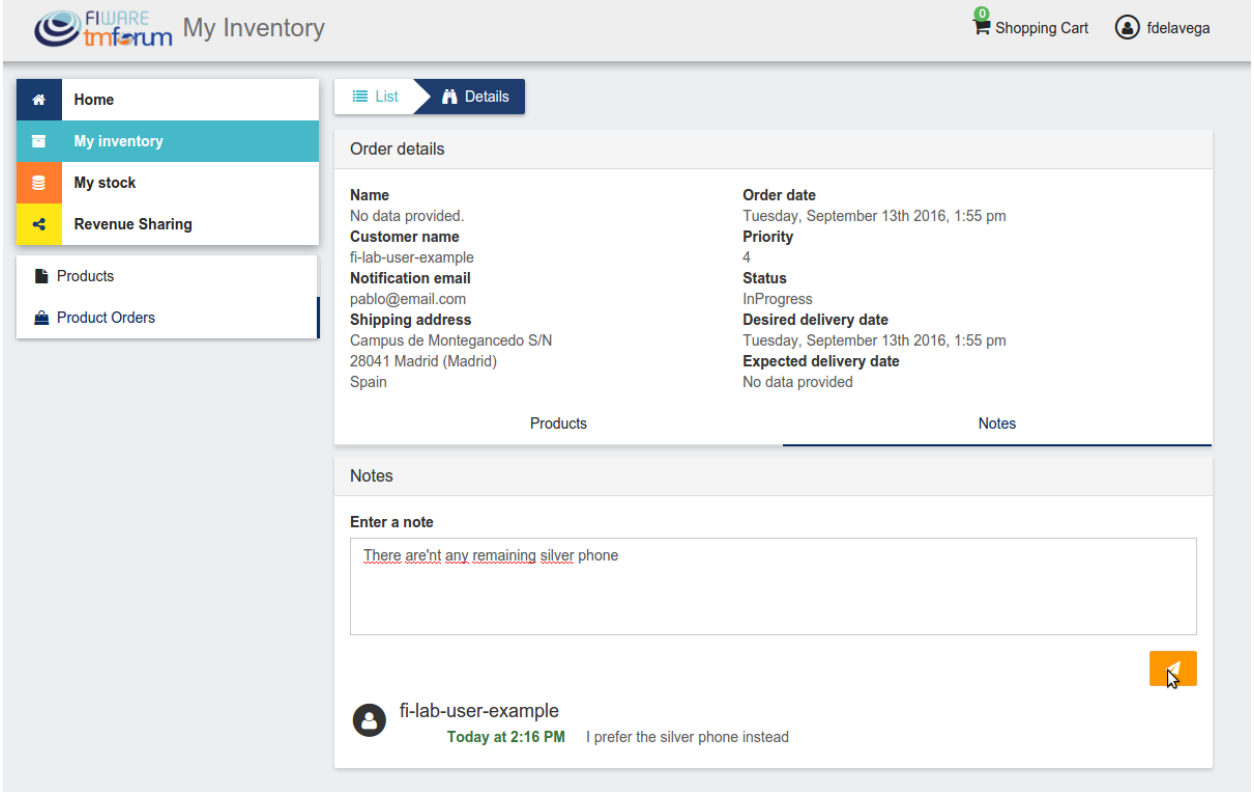



In the displayed view you can review the details of the order and the details of your products acquired by the customer, including the chosen characteristics.

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

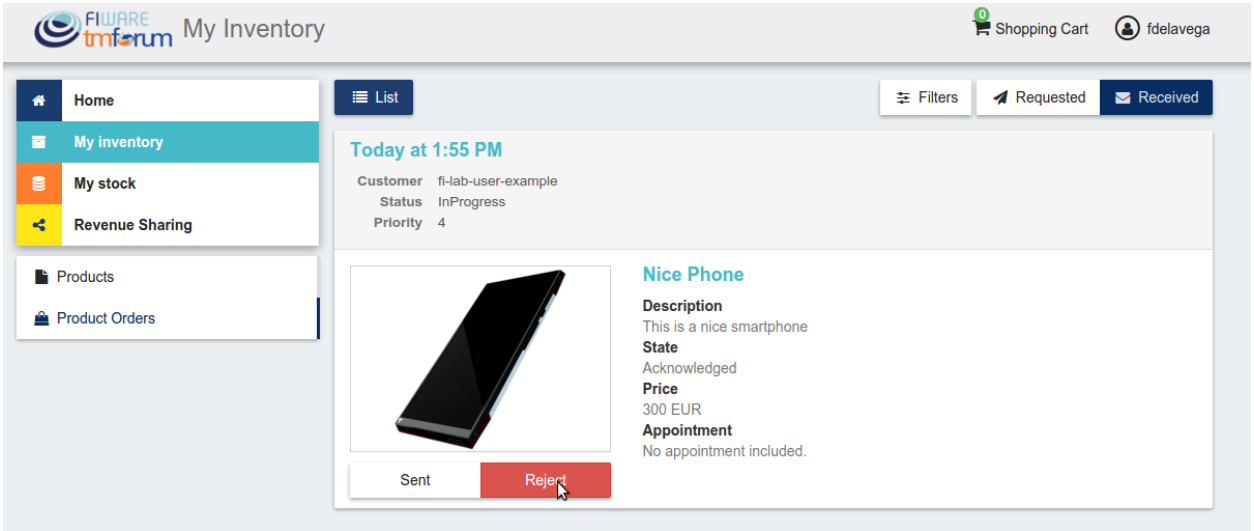


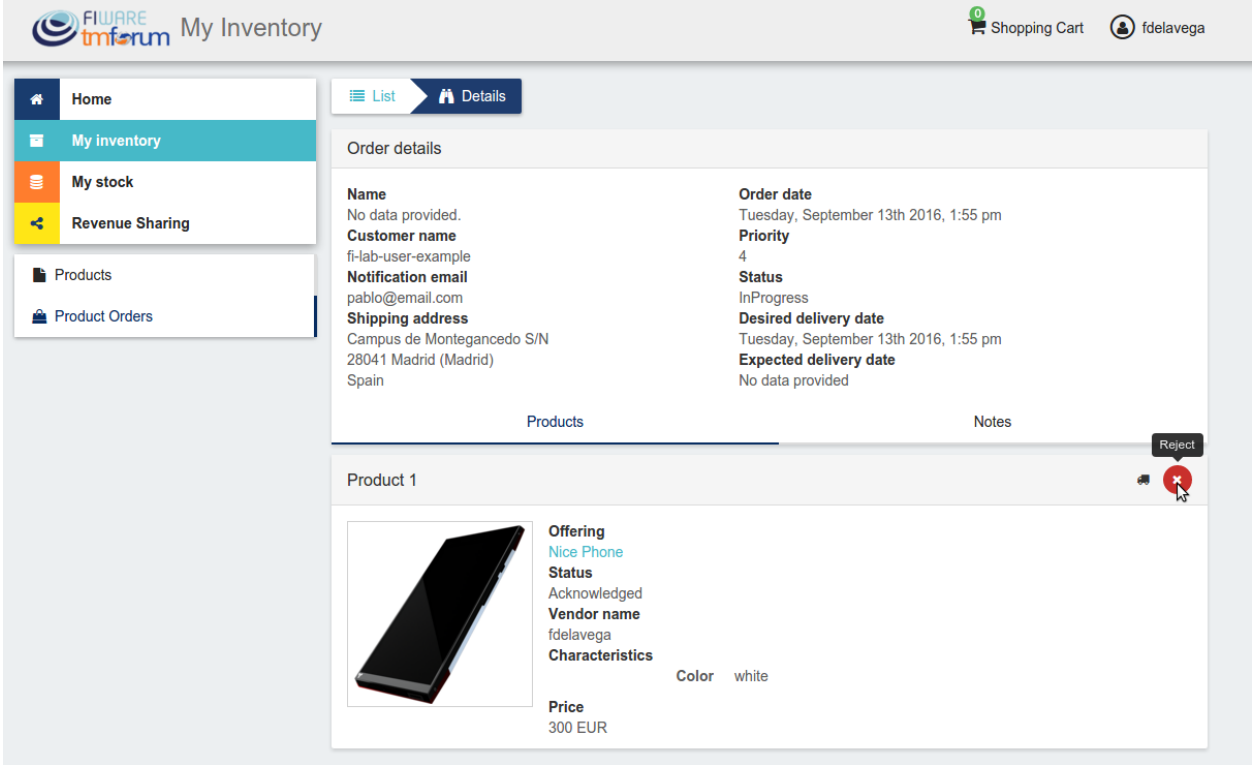

You can also give a reply to customer notes including it in the text area and clicking on the send button



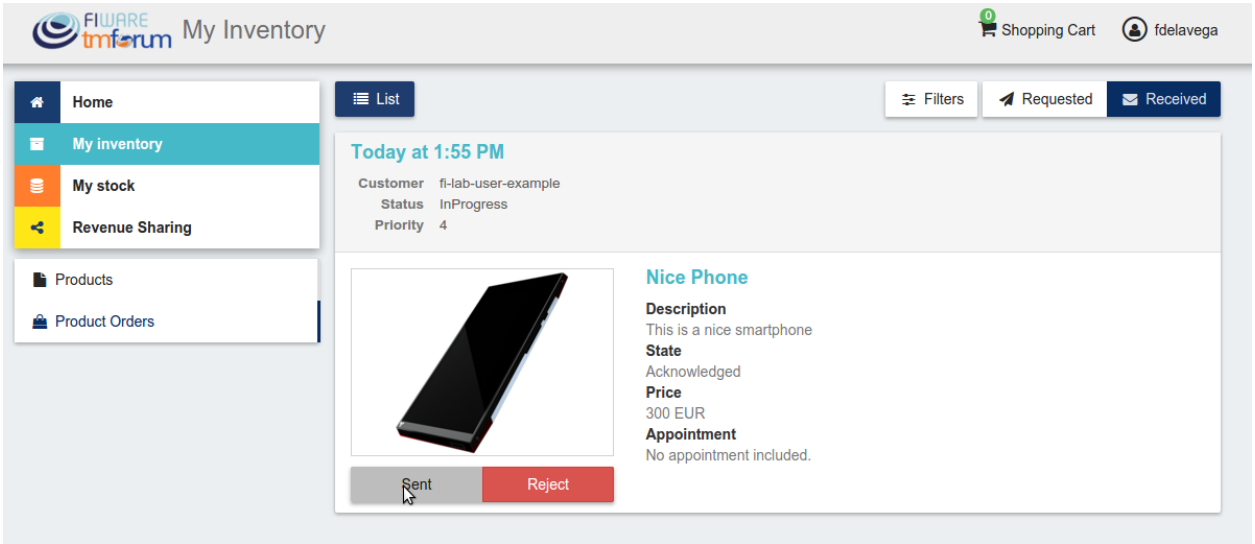

If the acquired product is not digital, the order needs to be processed manually by the seller, in the sense that the seller will have to send the acquired product to the customer. To deal with this situation, the order details view allows sellers to manually change the status of the order.

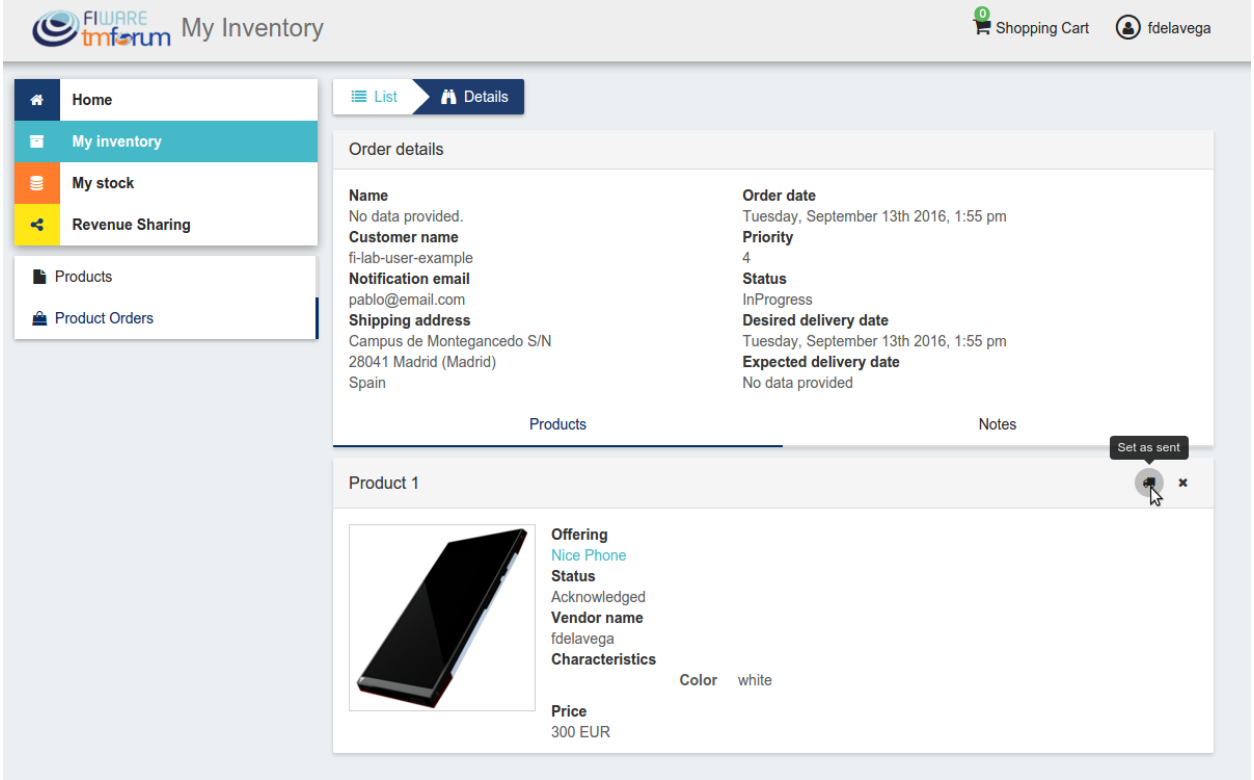
To reject a received order you have to click in the *Reject* button located in the search or in the details view of the order.



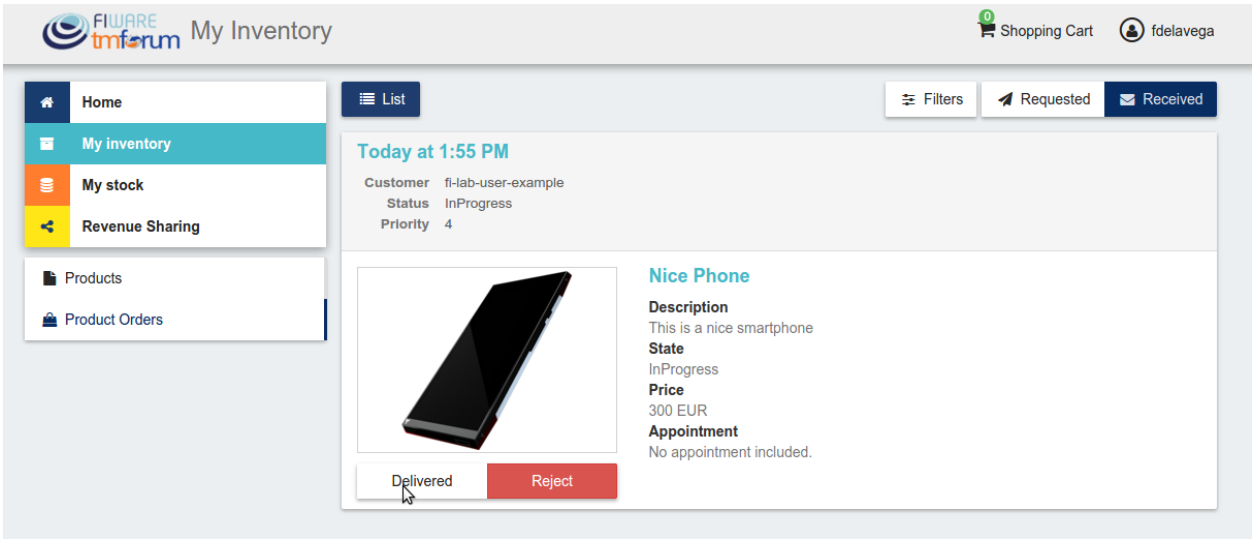


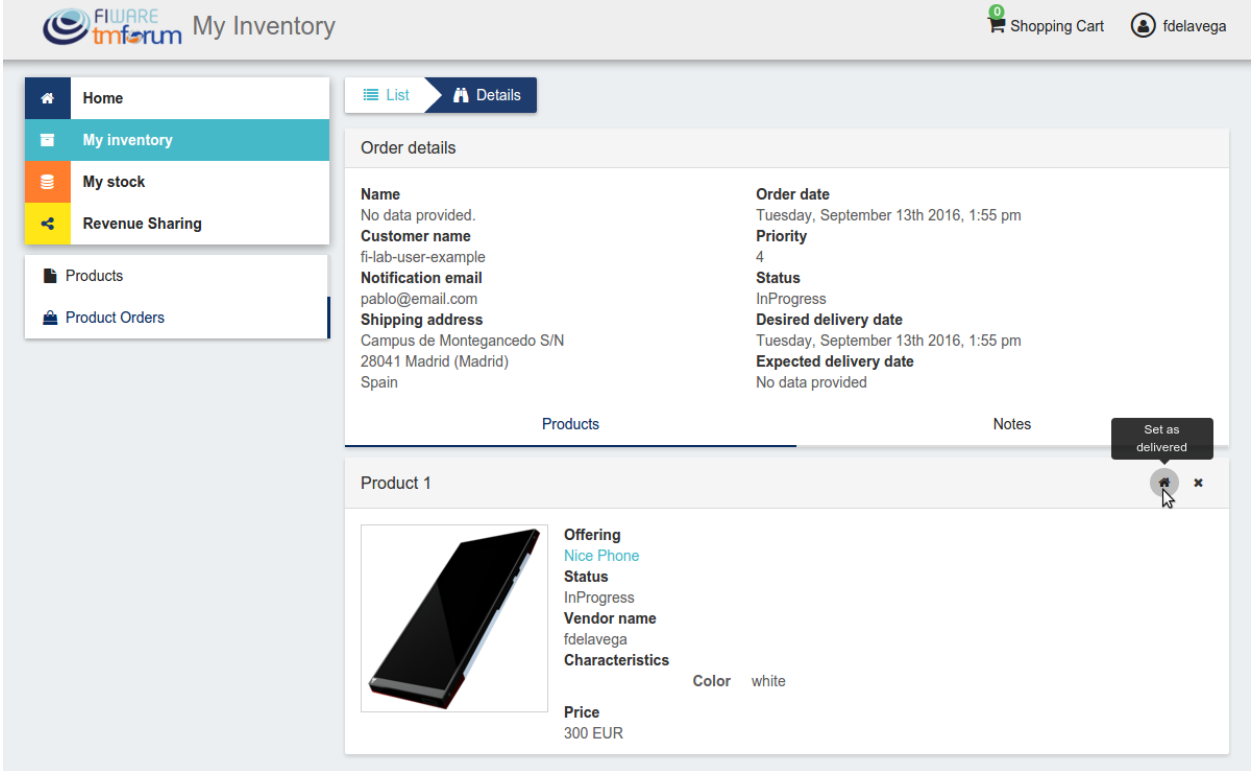

In case you accept the order and send the product to the customer, you have to put it as *inProgress* clicking on the *Sent* button






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



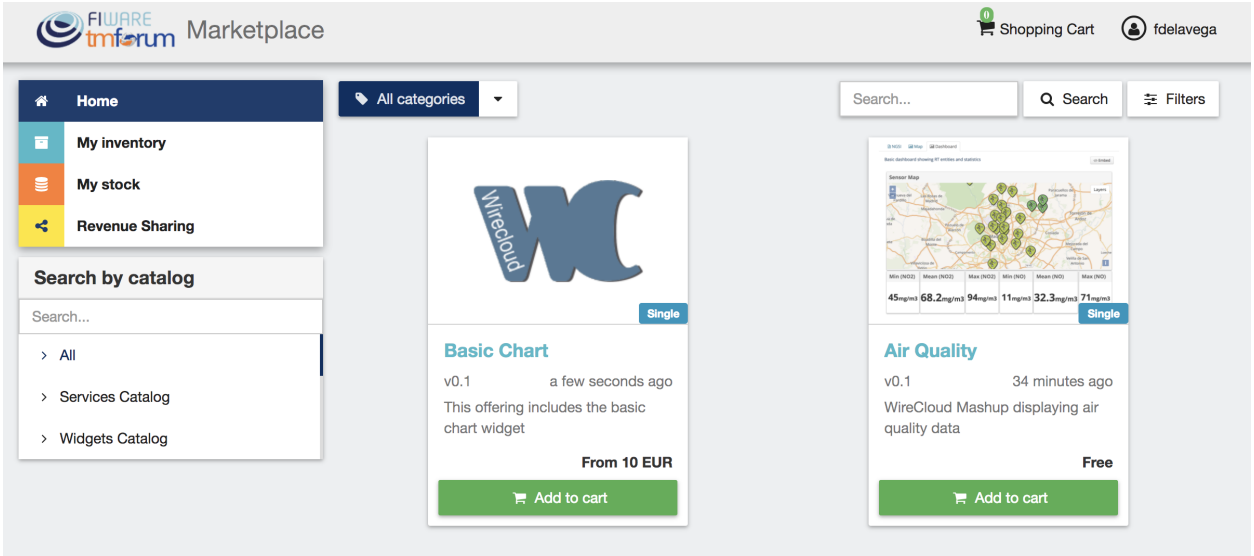



## 1.3.5 Customer

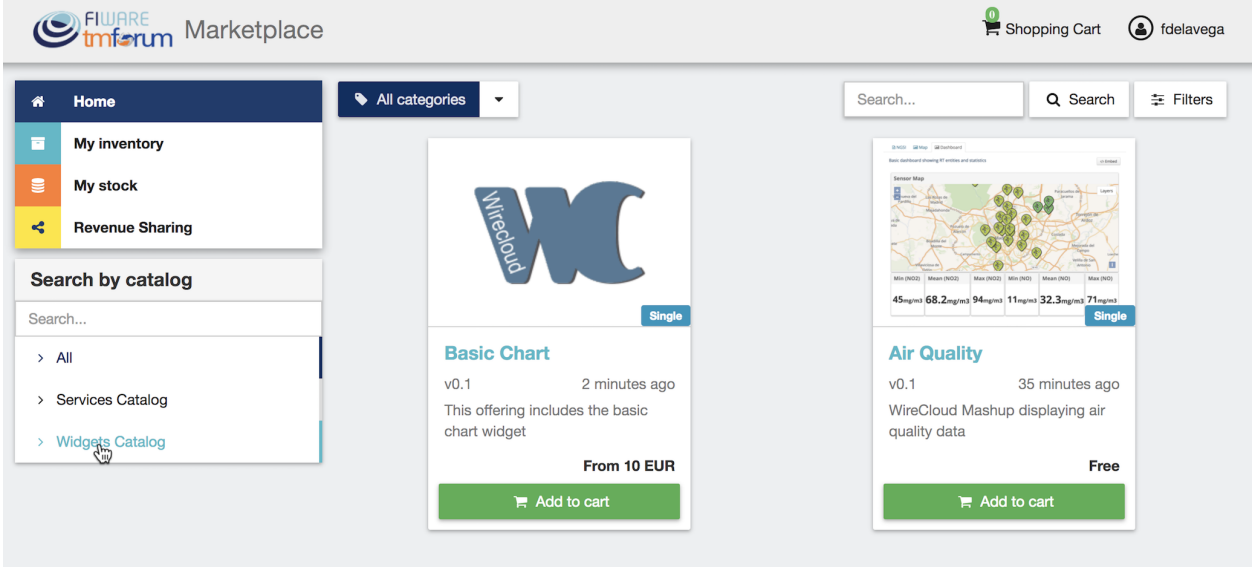
All of the users of the system have by default the *Customer* role. Customers are able to create orders for acquiring offerings.

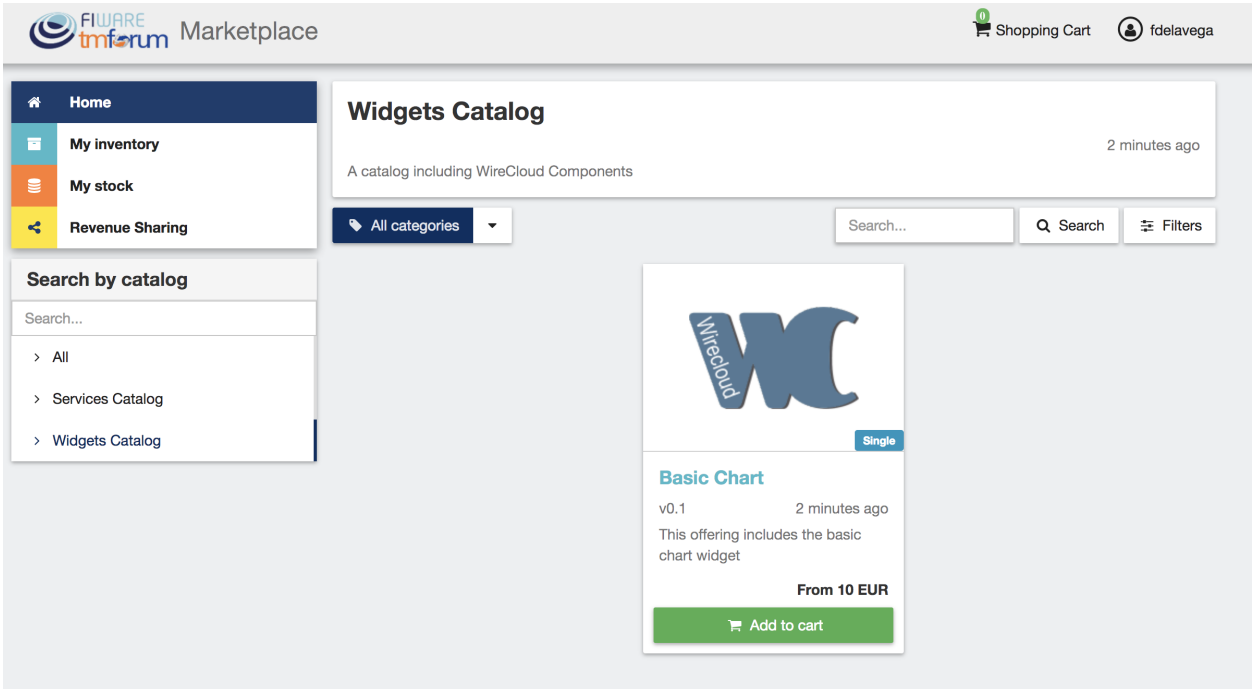
### List Available Offerings

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

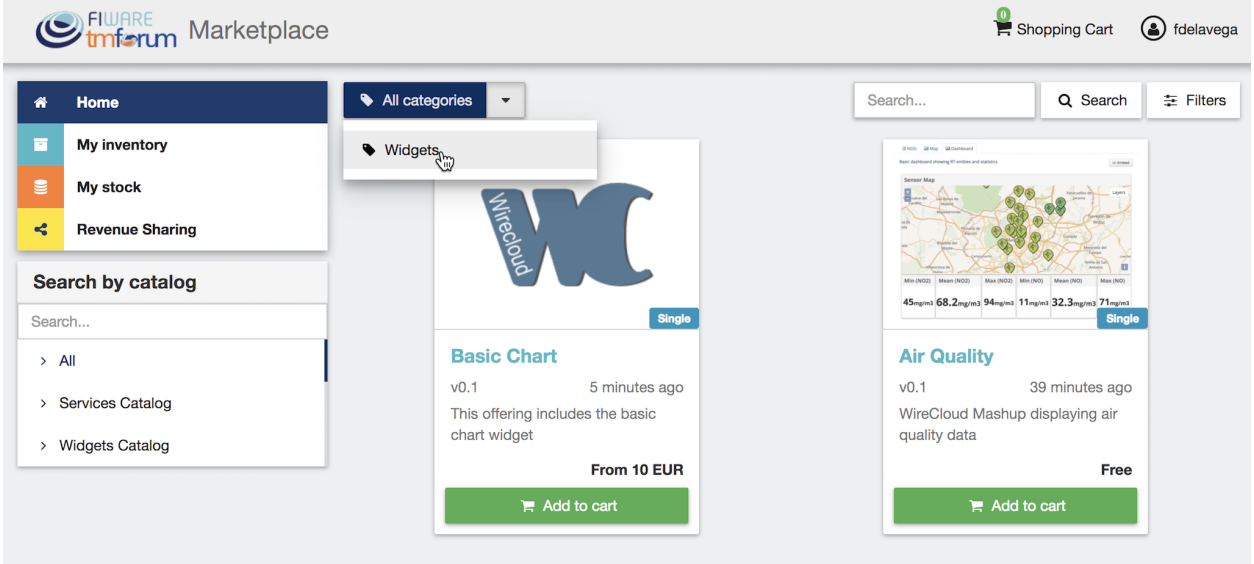


Additionally, customers can select a specific catalog of offerings by clicking on it.

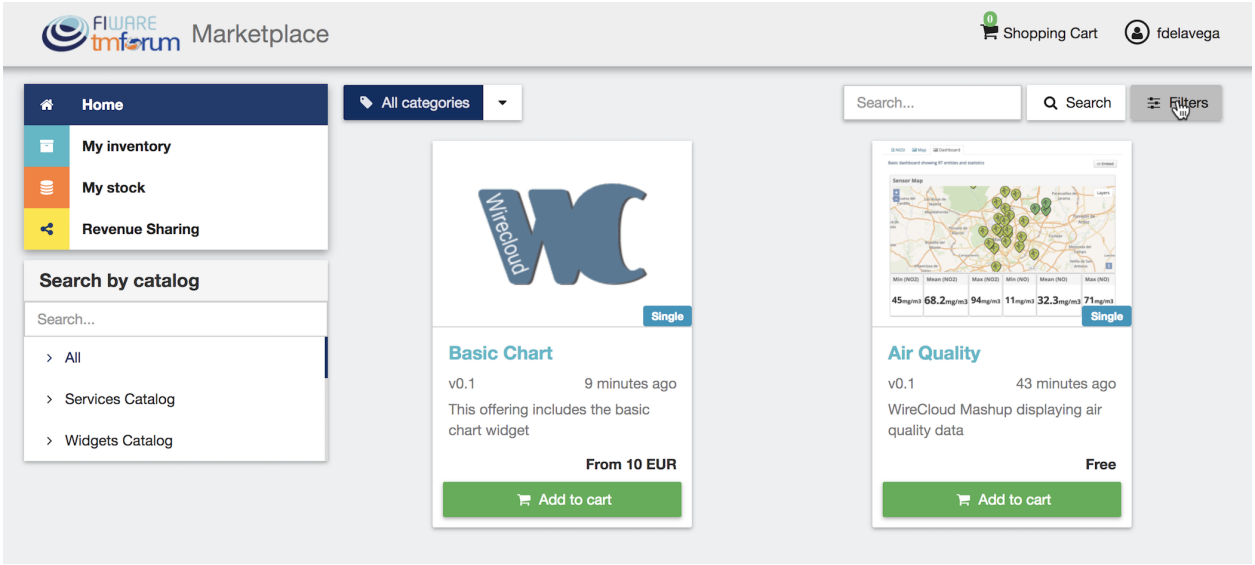


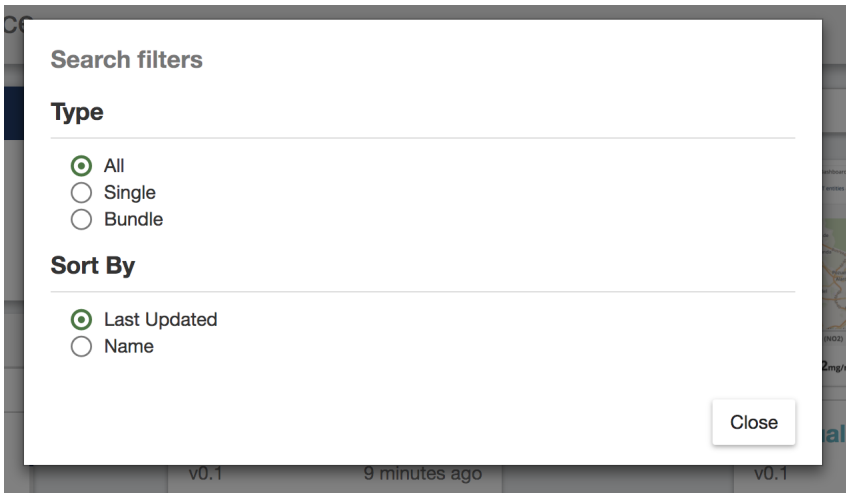




Moreover, customers can filter the shown offerings by category using the categories dropdown and choosing the wanted one.

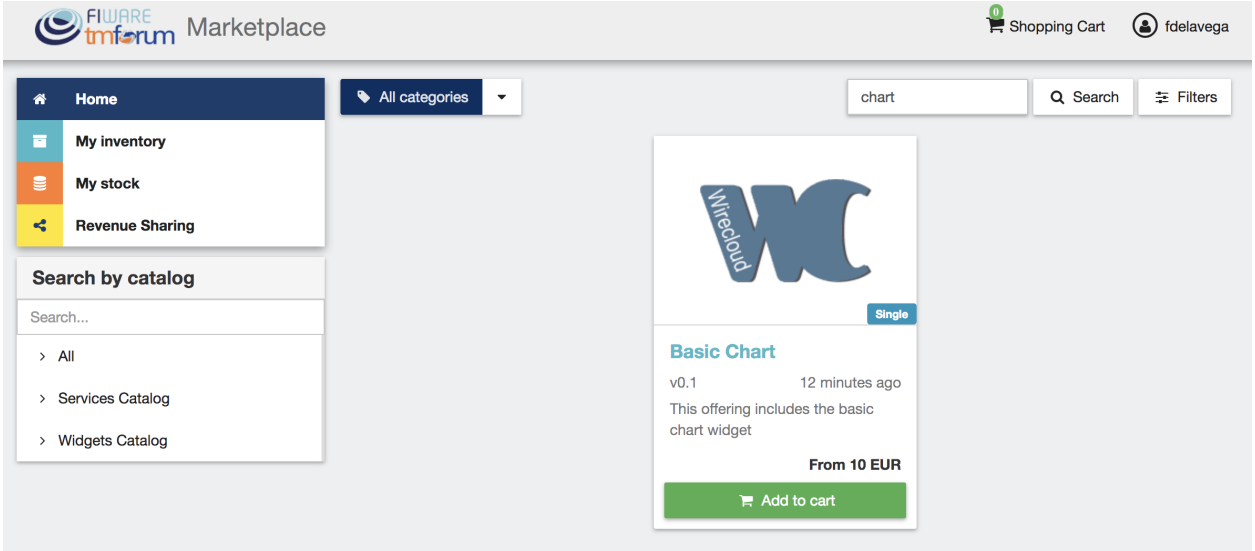



Customers can also filter bundle or single offerings using the *Filters* modal as well as choosing its sorting.

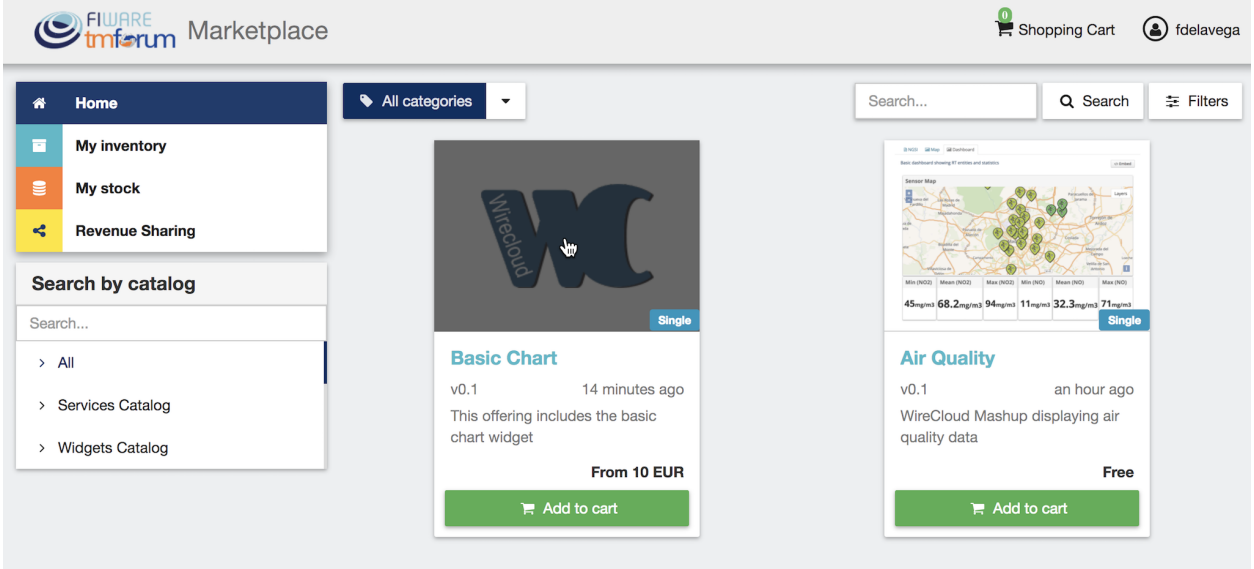


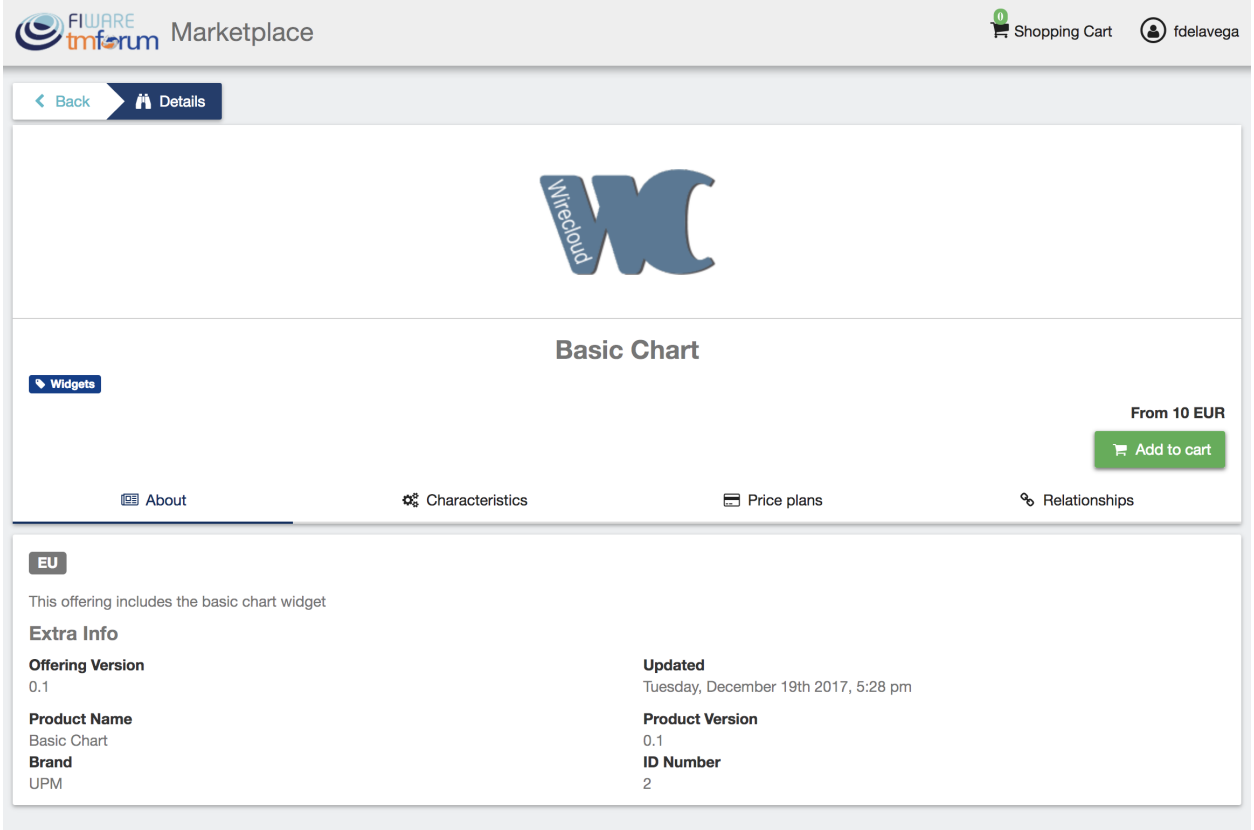
Finally, customers can search offerings by keyword using the provided search bar

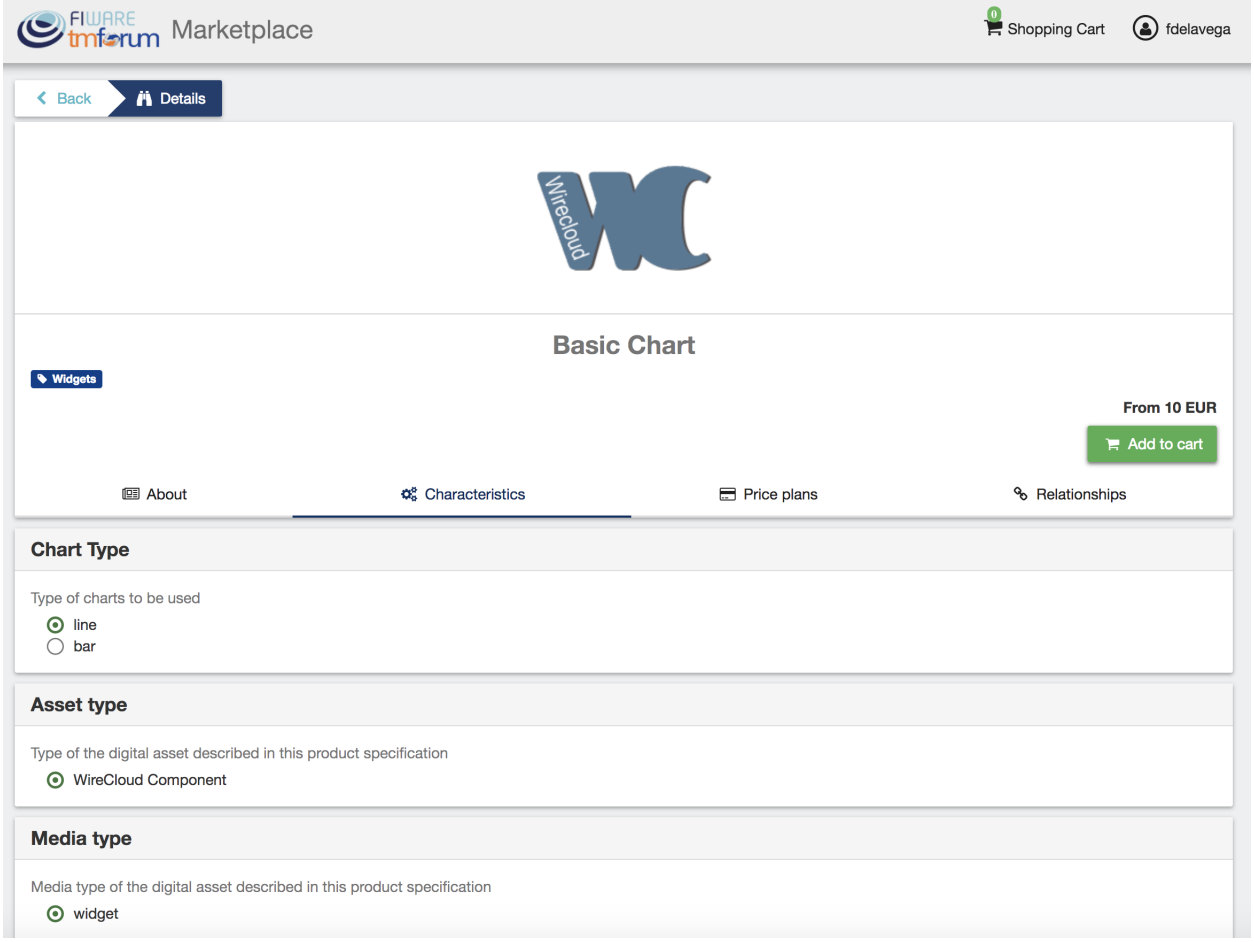


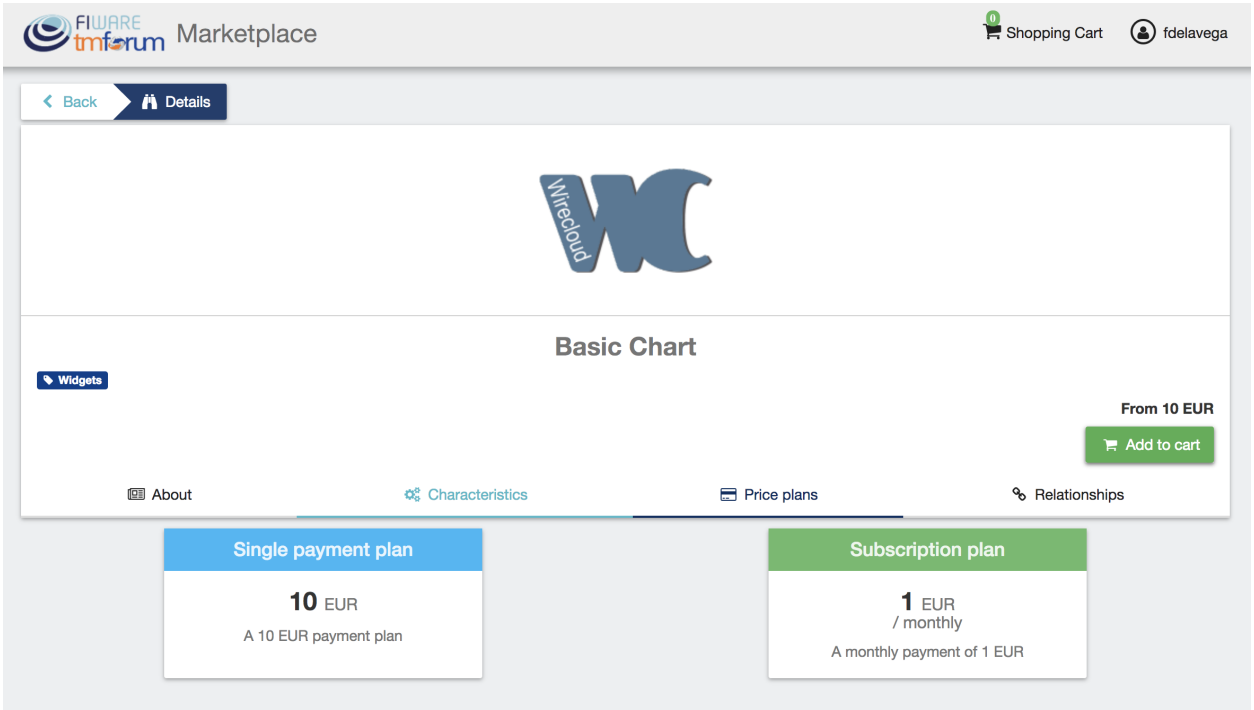

Customers can open the details of an offering by clicking on it




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
0
Shopping Cart
fdelavega
Back
Details
Wirecloud
Basic Chart
Widgets
From 10 EUR
Add to cart
About
Characteristics
Price plans
Relationships
EU
This offering includes the basic chart widget
Extra Info
Offering Version
0.1
Updated
Tuesday, December 19th 2017, 5:28 pm
Product Name
Basic Chart
Product Version
0.1
Brand
UPM
ID Number
2

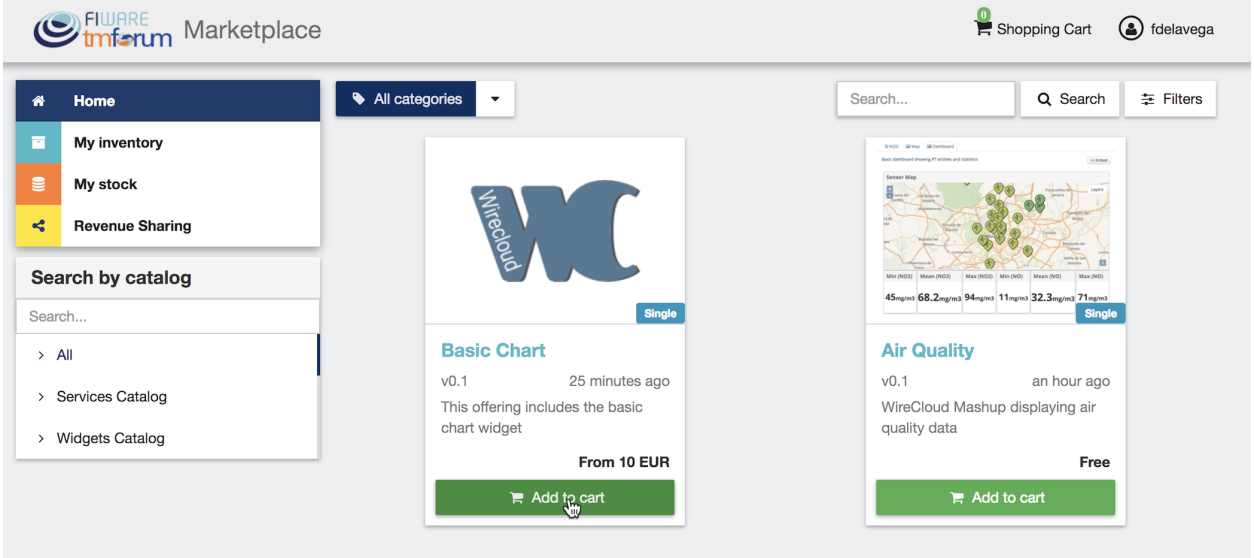

FIWARE tmforum Marketplace
Shopping Cart
fdelavega
Back
Details
Wirecloud
Basic Chart
Widgets
From 10 EUR
Add to cart
About
Characteristics
Price plans
Relationships
Chart Type
Type of charts to be used
line
bar
Asset type
Type of the digital asset described in this product specification
WireCloud Component
Media type
Media type of the digital asset described in this product specification
widget

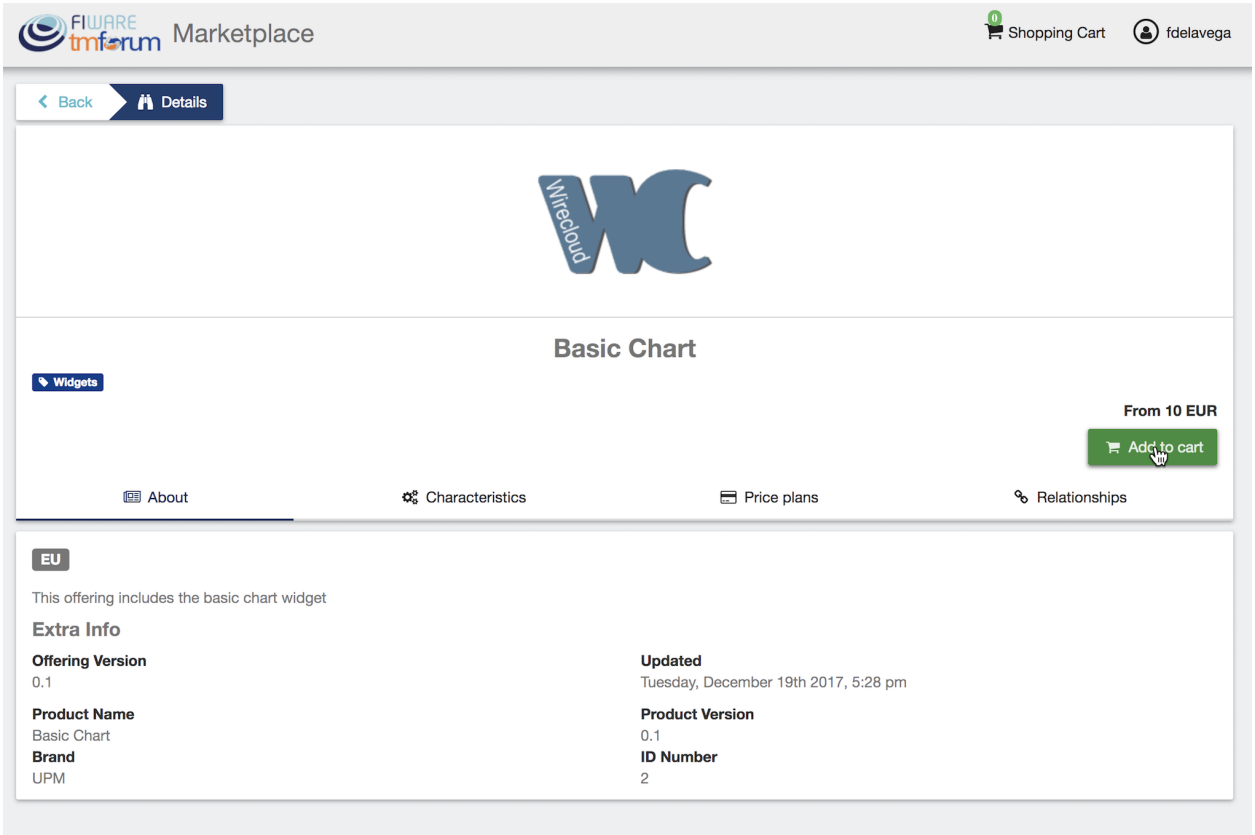

FIWARE tmforum Marketplace
Shopping Cart
fdelavega
Back
Details
Wirecloud
Basic Chart
Widgets
From 10 EUR
Add to cart
About
Characteristics
Price plans
Relationships
Single payment plan
10 EUR
A 10 EUR payment plan
Subscription plan
1 EUR
/ monthly
A monthly payment of 1 EUR

### Create Order

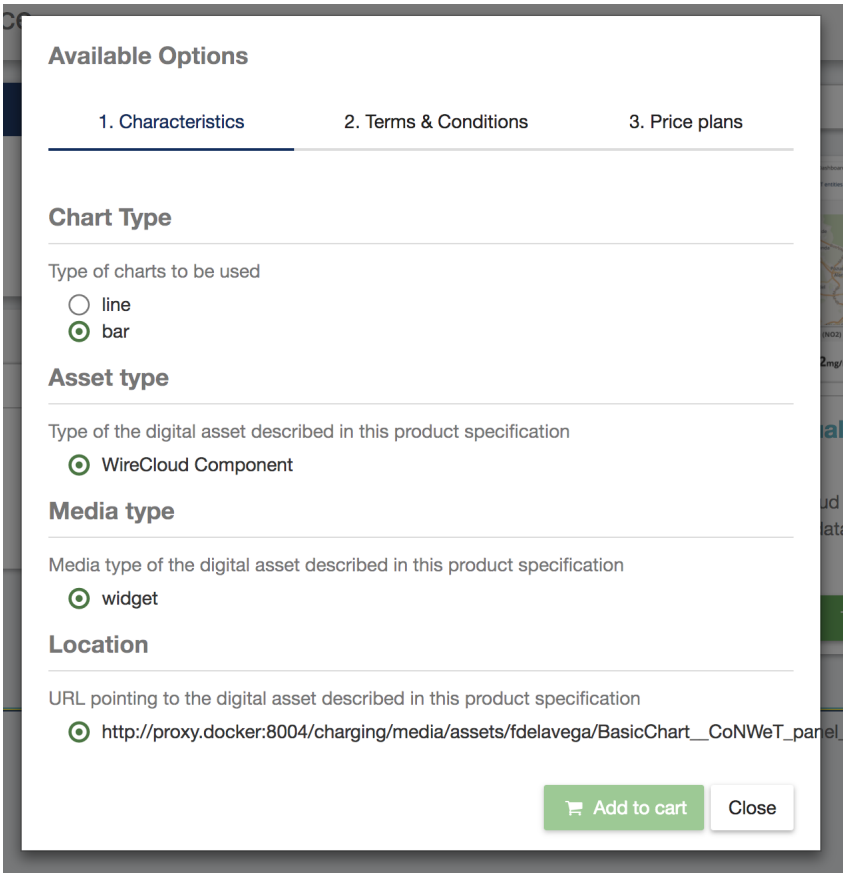
Customers can create orders for acquiring offerings. The different offerings to be included in an order are managed using the *Shopping Cart*.

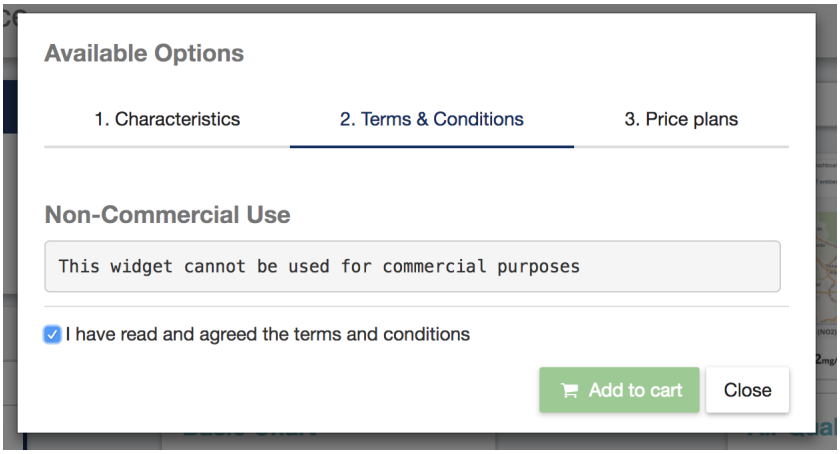
To include an offering in the shopping cart there are two possibilities. You can click on the *Add to Cart* button located in the offering panel when searching, or you can click on the *Add to Cart* button located in the offering details view.

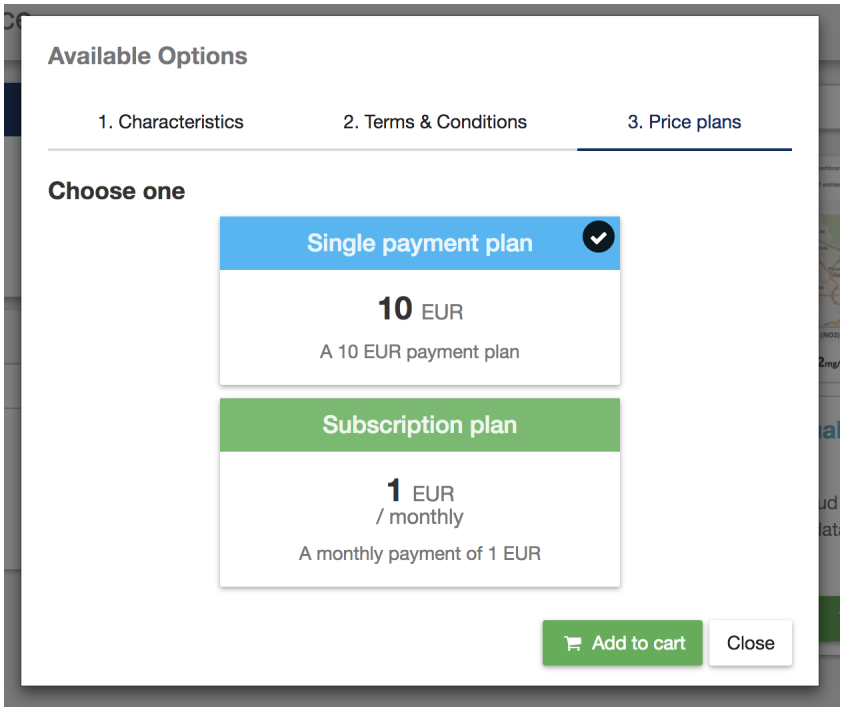


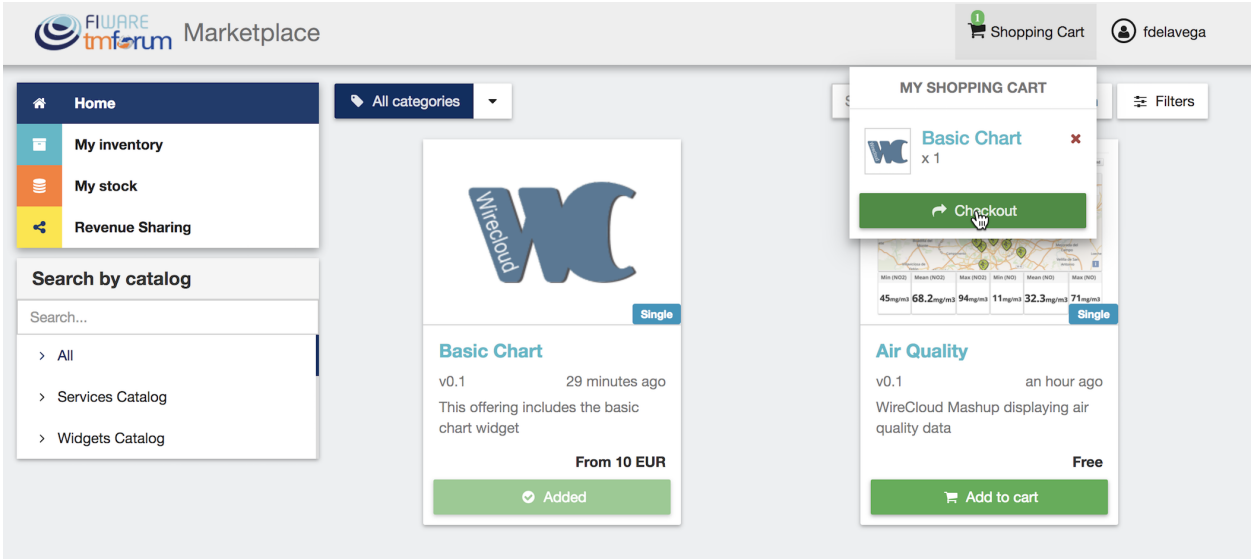

If the offering has configurable characteristics, multiple price plans or terms and conditions, a modal will be displayed where you can select your preferred options


Available Options
1. Characteristics
2. Terms & Conditions
3. Price plans
Chart Type
Type of charts to be used
line
bar
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
Location
URL pointing to the digital asset described in this product specification
http://proxy.docker:8004/charging/media/assets/fdelavega/BasicChart__CoNWeT_panel_
Add to cart
Close

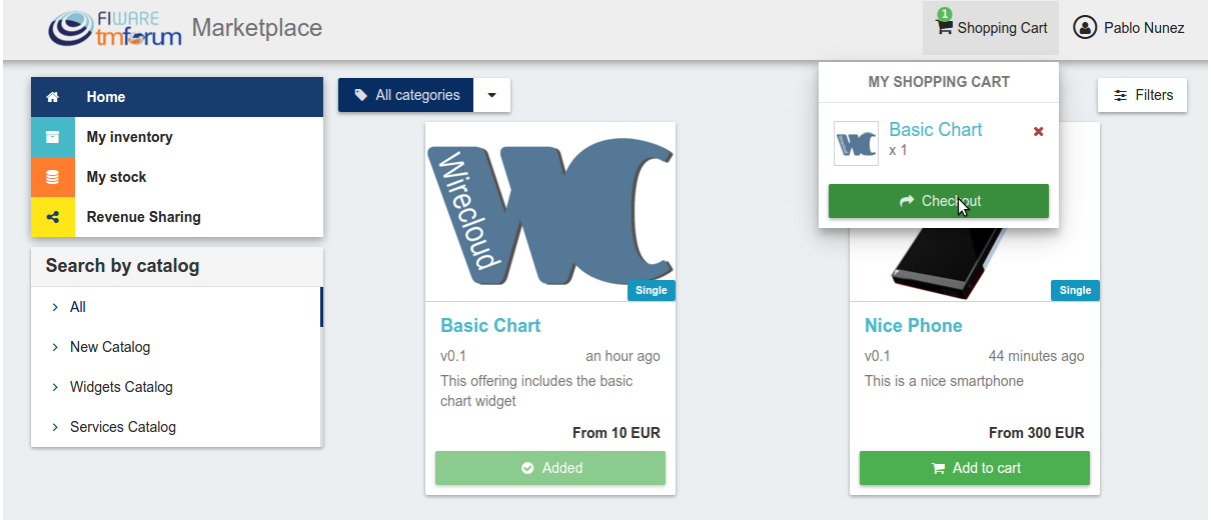

Available Options
1. Characteristics
2. Terms & Conditions
3. Price plans
Non-Commercial Use
This widget cannot be used for commercial purposes
I have read and agreed the terms and conditions
Add to cart
Close




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



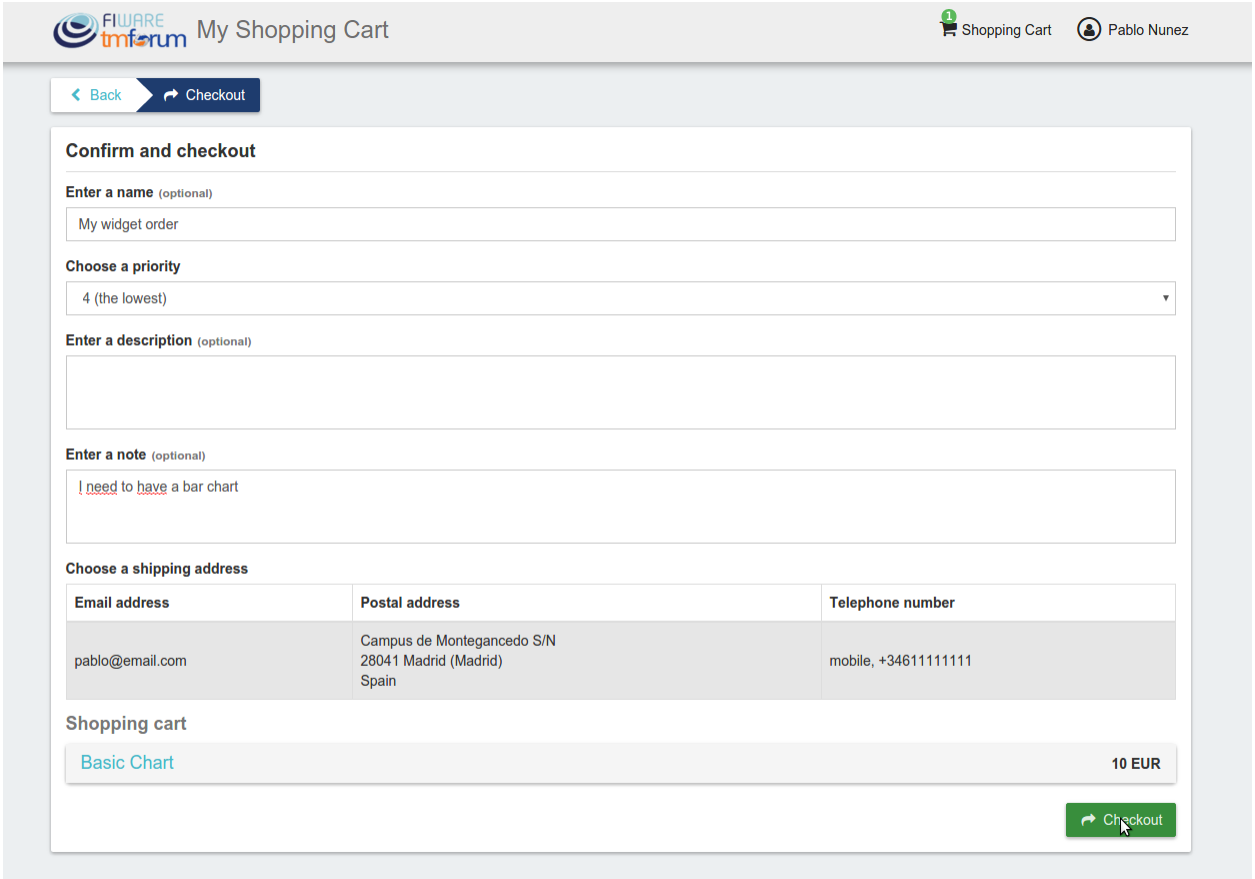

Once you have included all the offerings you want to acquire to the shopping cart, you can create the order clicking on *Shopping Cart*, and then on *Checkout*



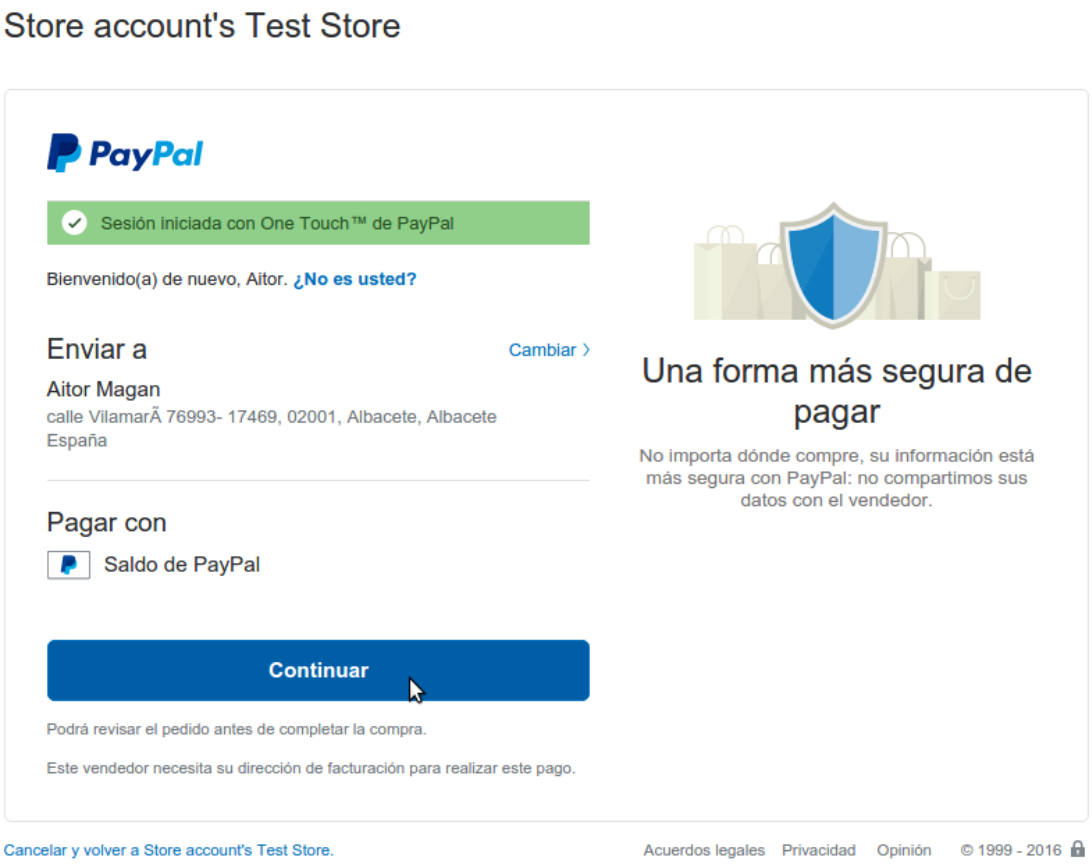

In the displayed form, you can include an optional name, an optional description, or an optional note. Notes can include any additional information you want to provide to the sellers of the acquired offerings.

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

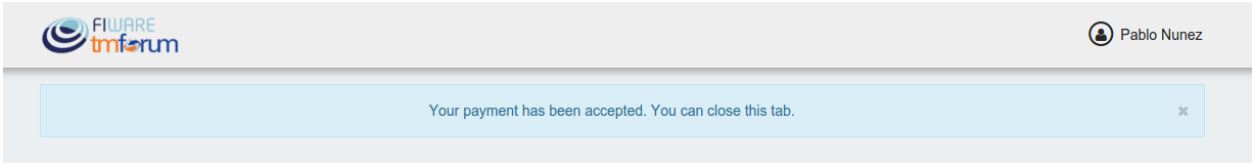
Once you have provided all the required information you can start the order creation clicking on *Checkout*




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

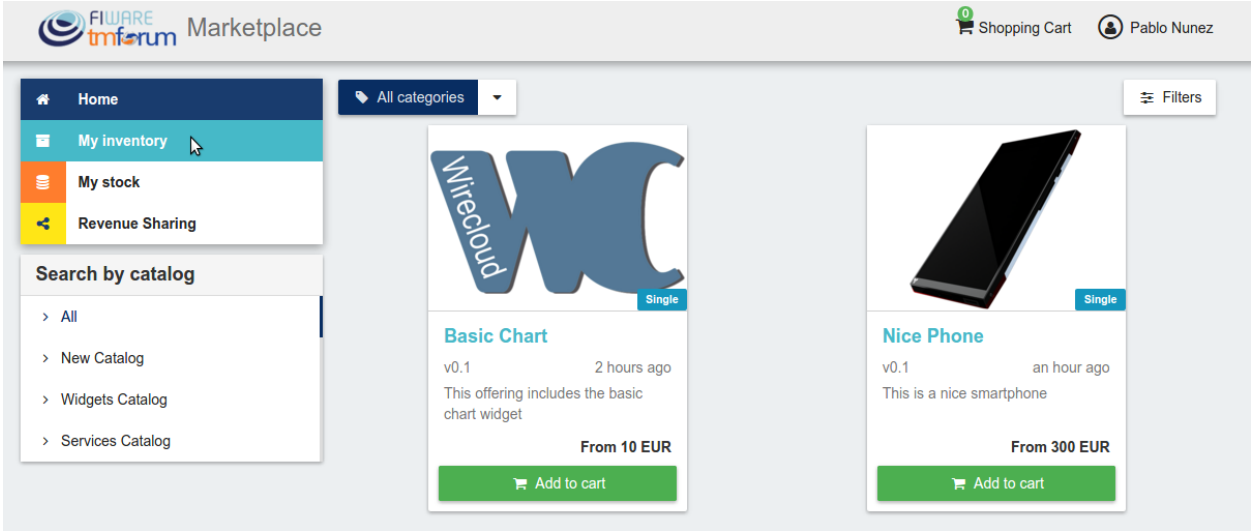



Finally, you will see a confirmation page

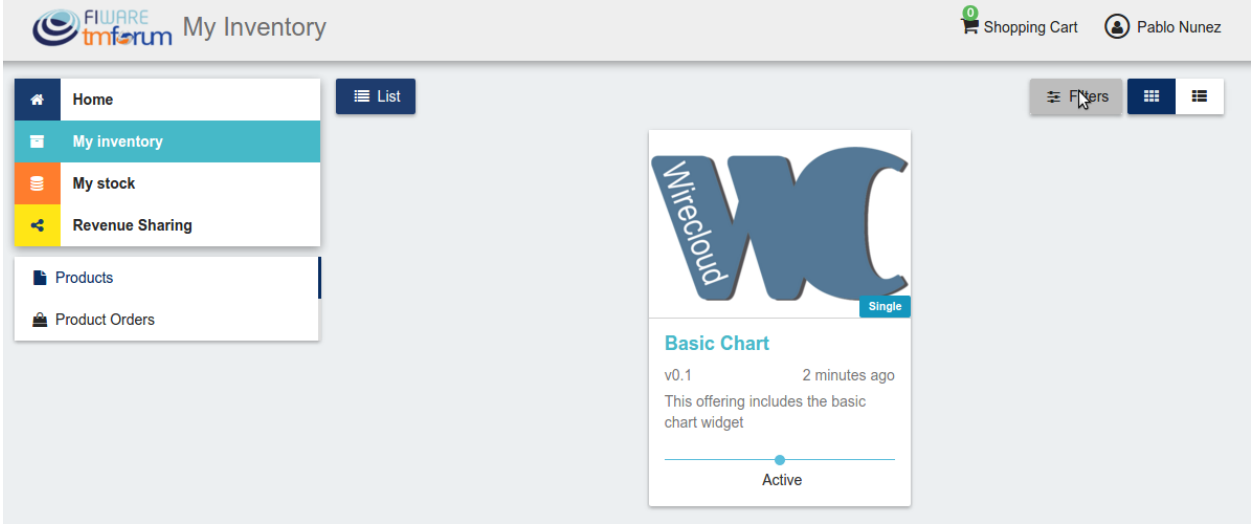


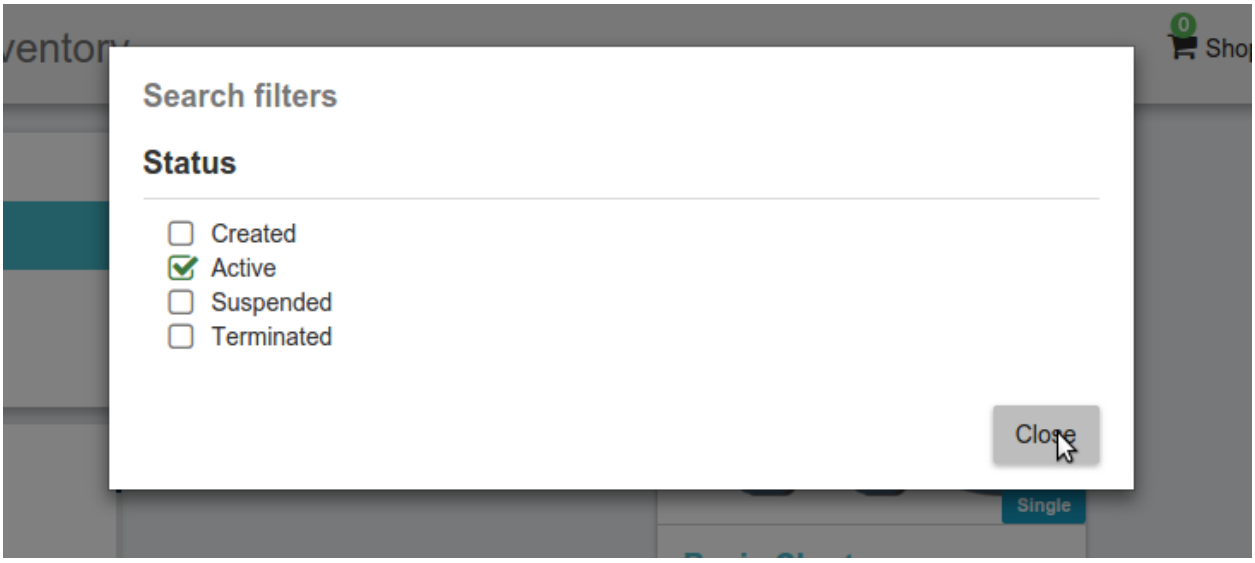
### Manage Acquired Products

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

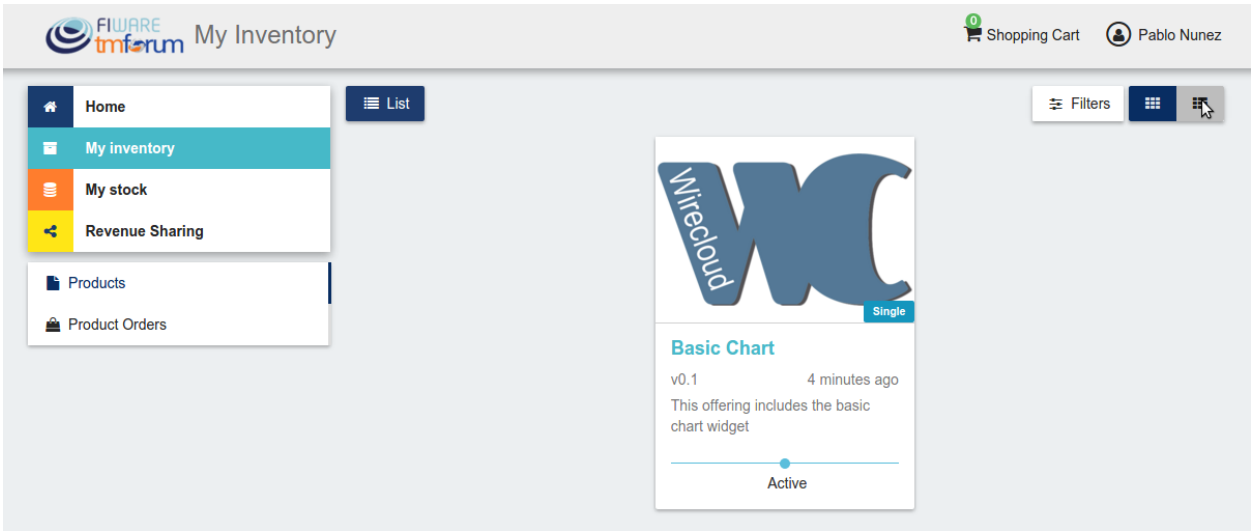


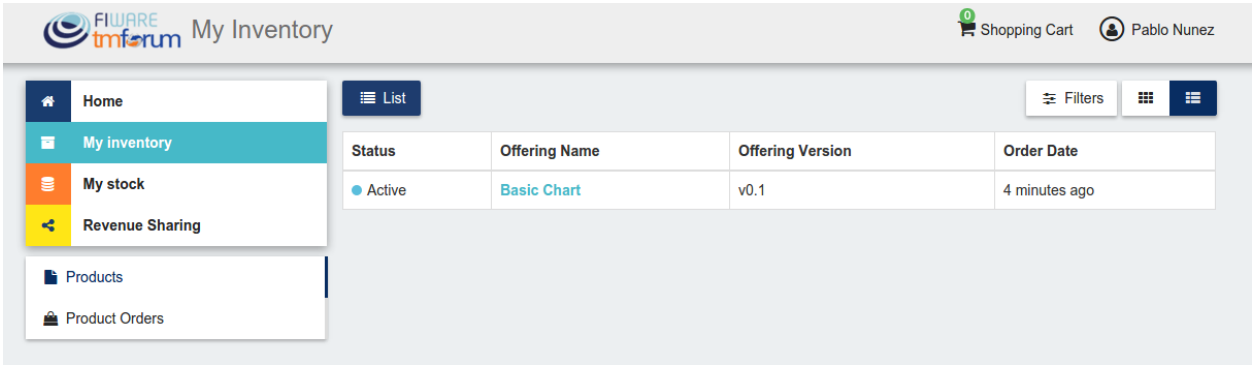

In this view, it is possible to filter you products by its status. To do that click on *Filters*, select the related statuses, and click on *Close*



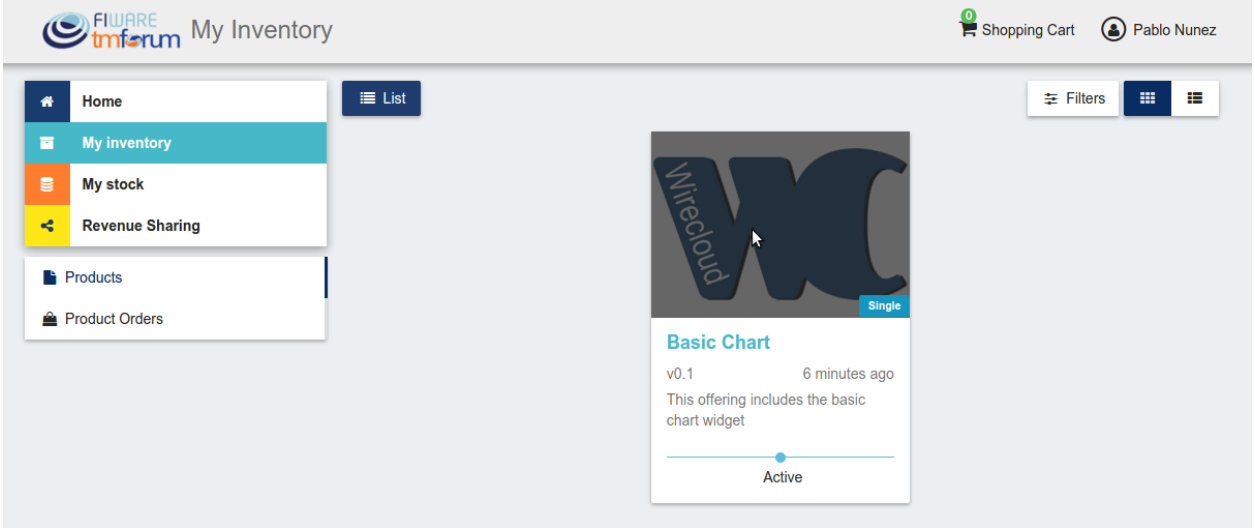



It is also possible to switch between the grid and tabular views using the related buttons

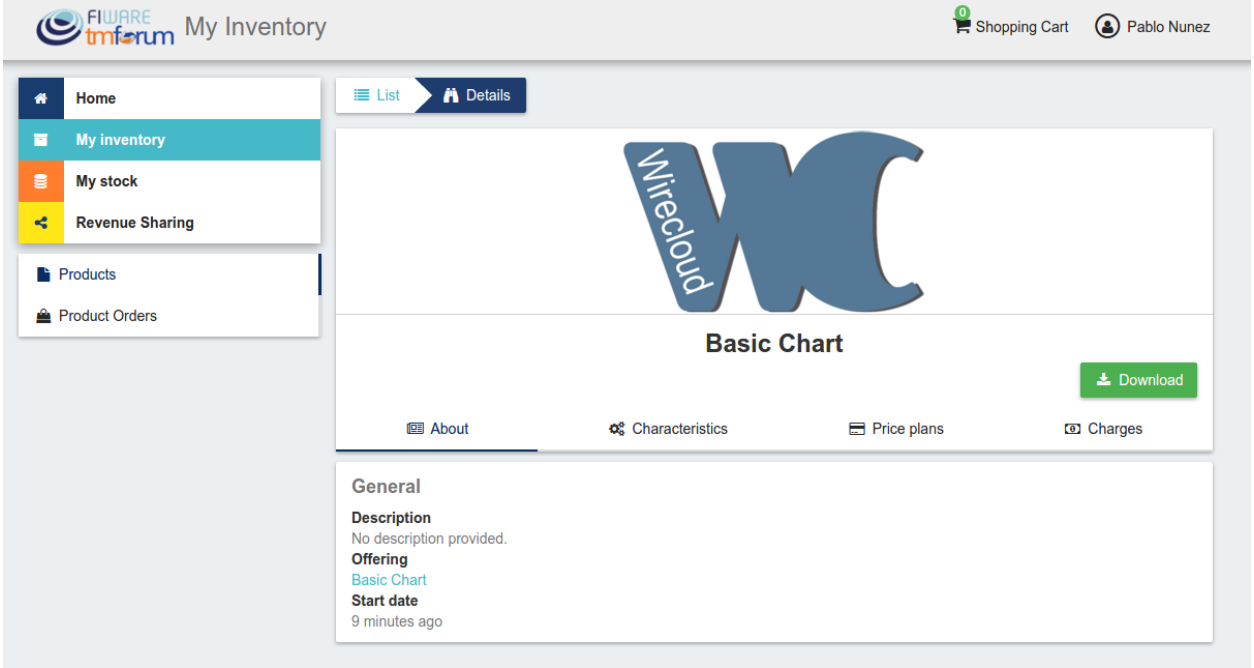


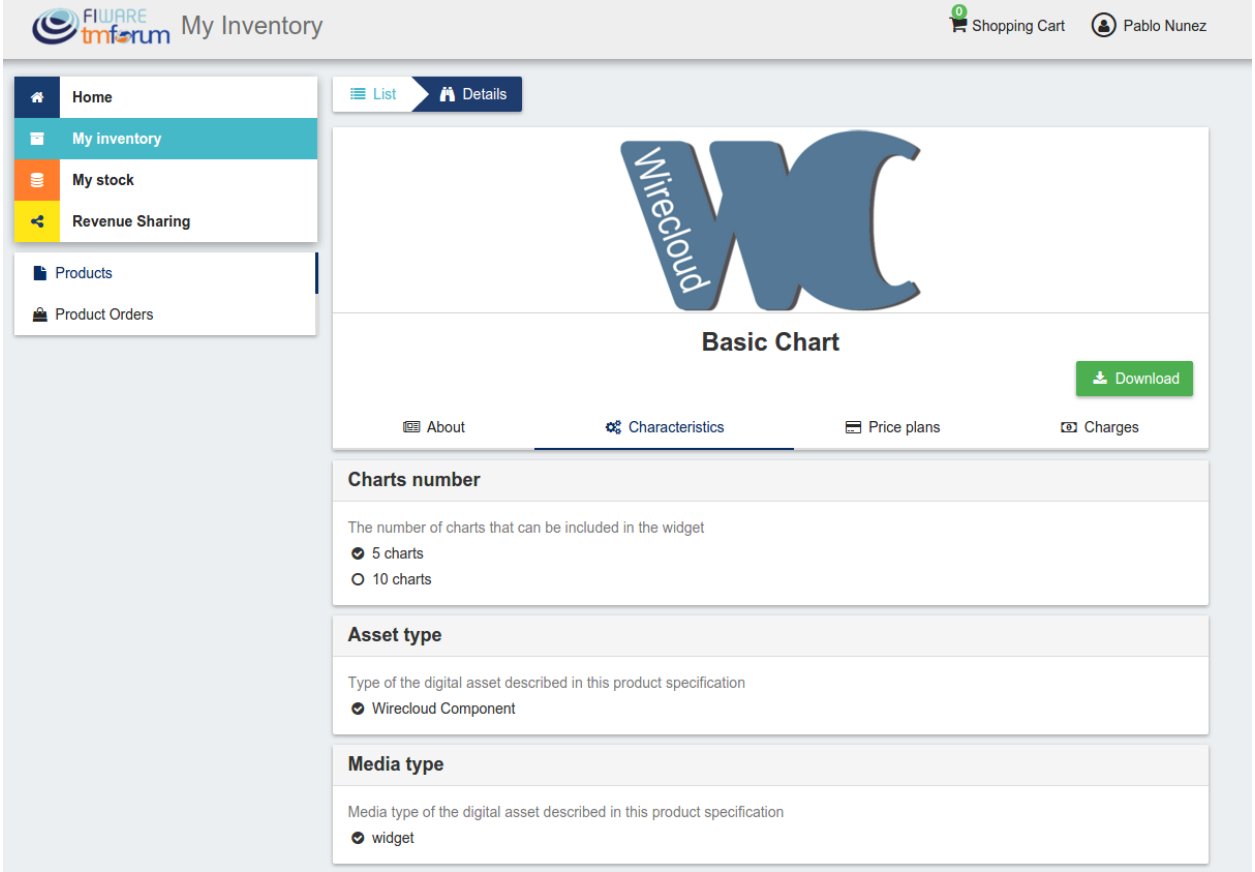



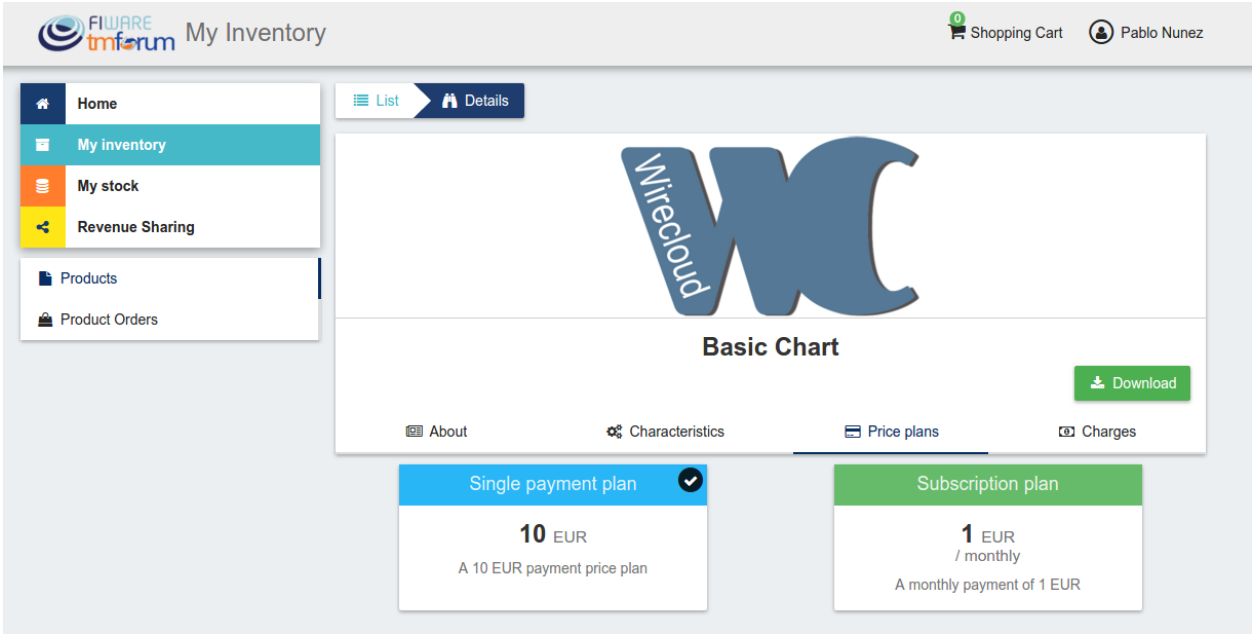

You can manage a specific acquired product clicking on it



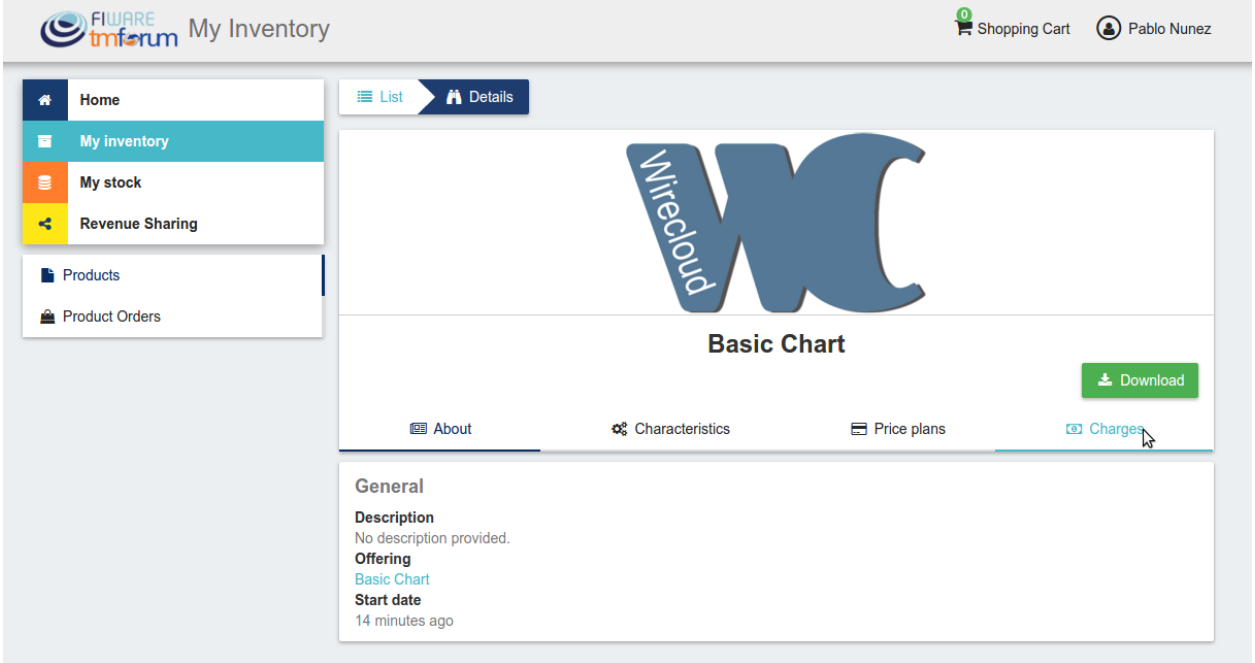

In the displayed view, you can see the general info of the acquired product, and the characteristics and pricing you have selected.



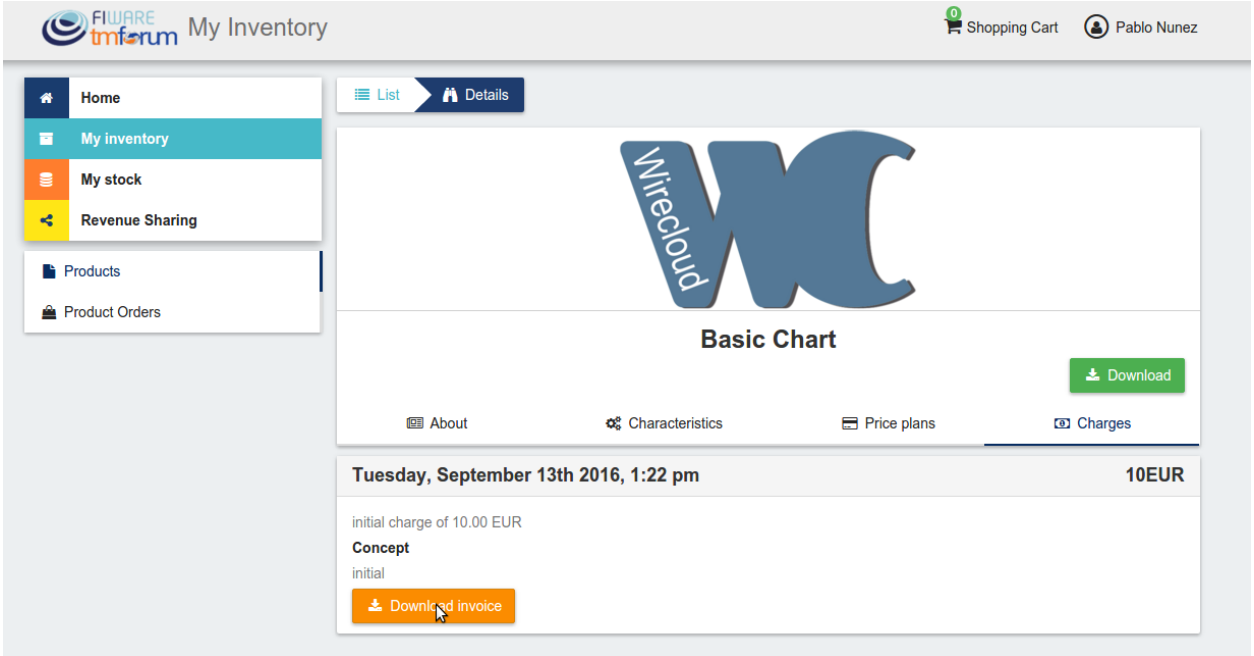


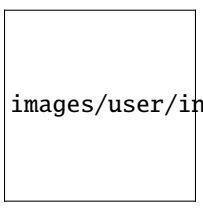

Additionally, you can see your charges related to the product accessing to the *Charges* tab



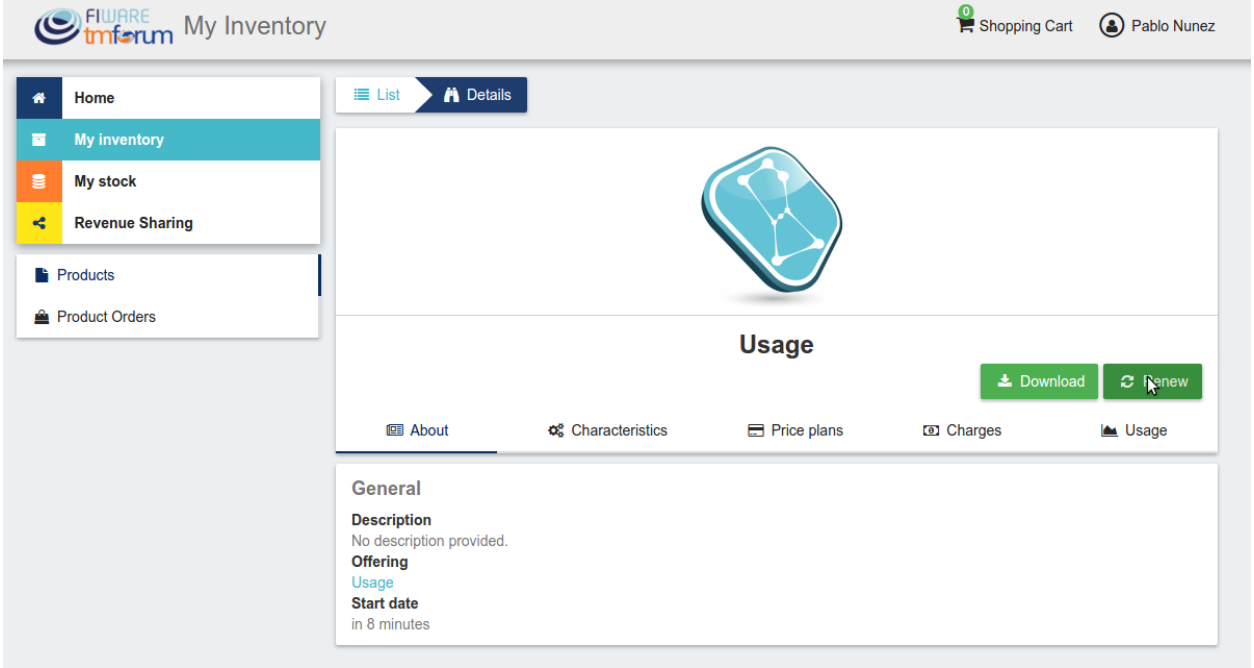

In this tab, you will find detailed information of the different charges and you will be able to download the related invoice clicking on *Download Invoice*




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

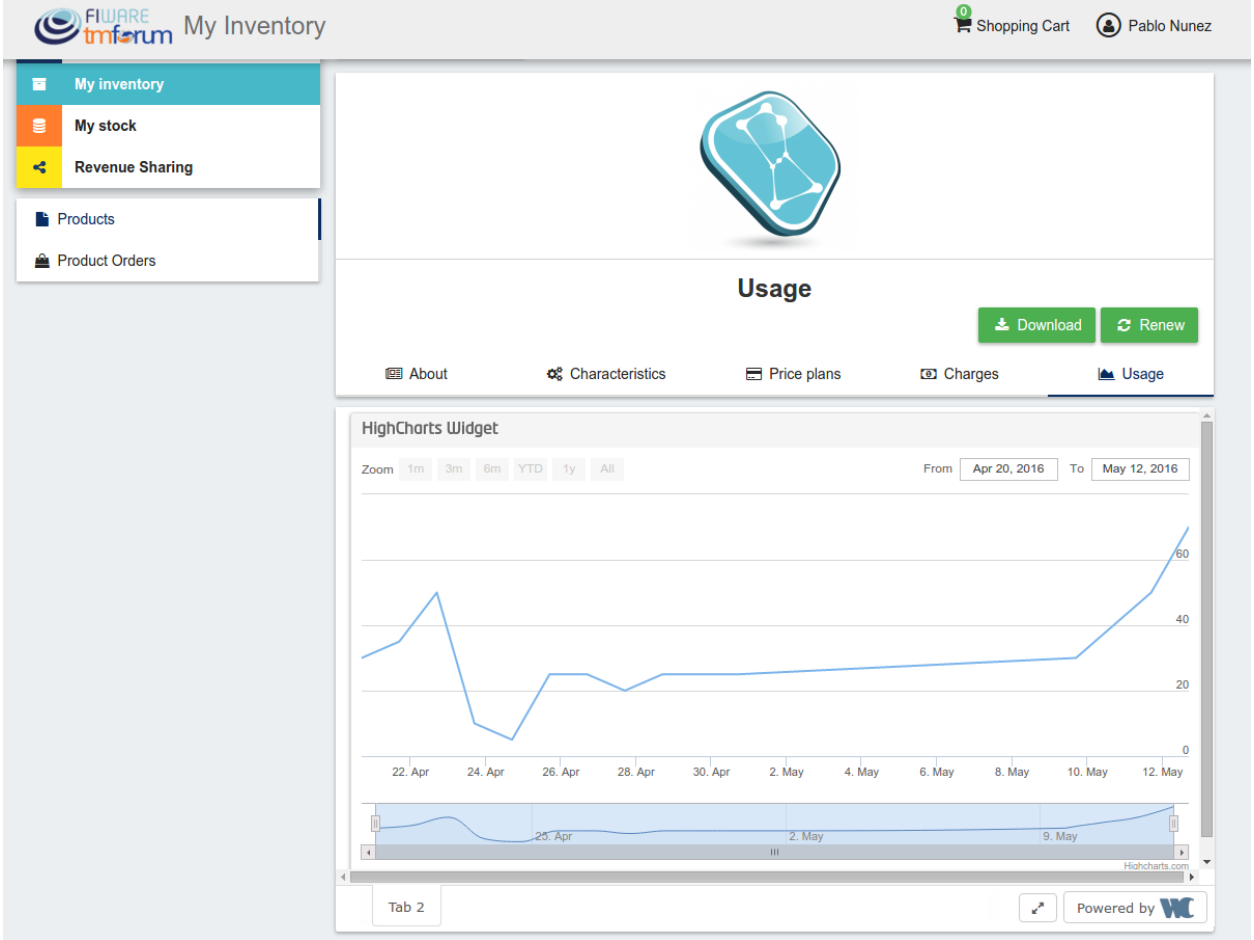


In case the chosen pricing model defines a recurring payment or a usage payment, you will be able to renew your product clicking on *Renew*. After clicking, you will be redirected to PayPal to pay the related amount.



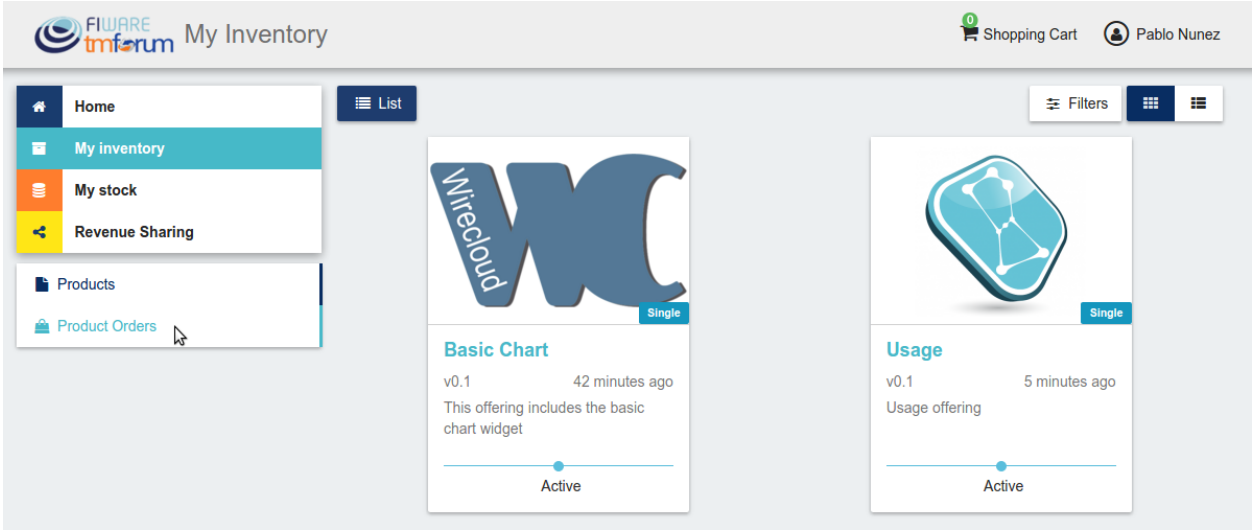

**Note:** If you product has expired and you do not renew it, it will be suspended, which means you will not have access to the acquired service until you pay

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



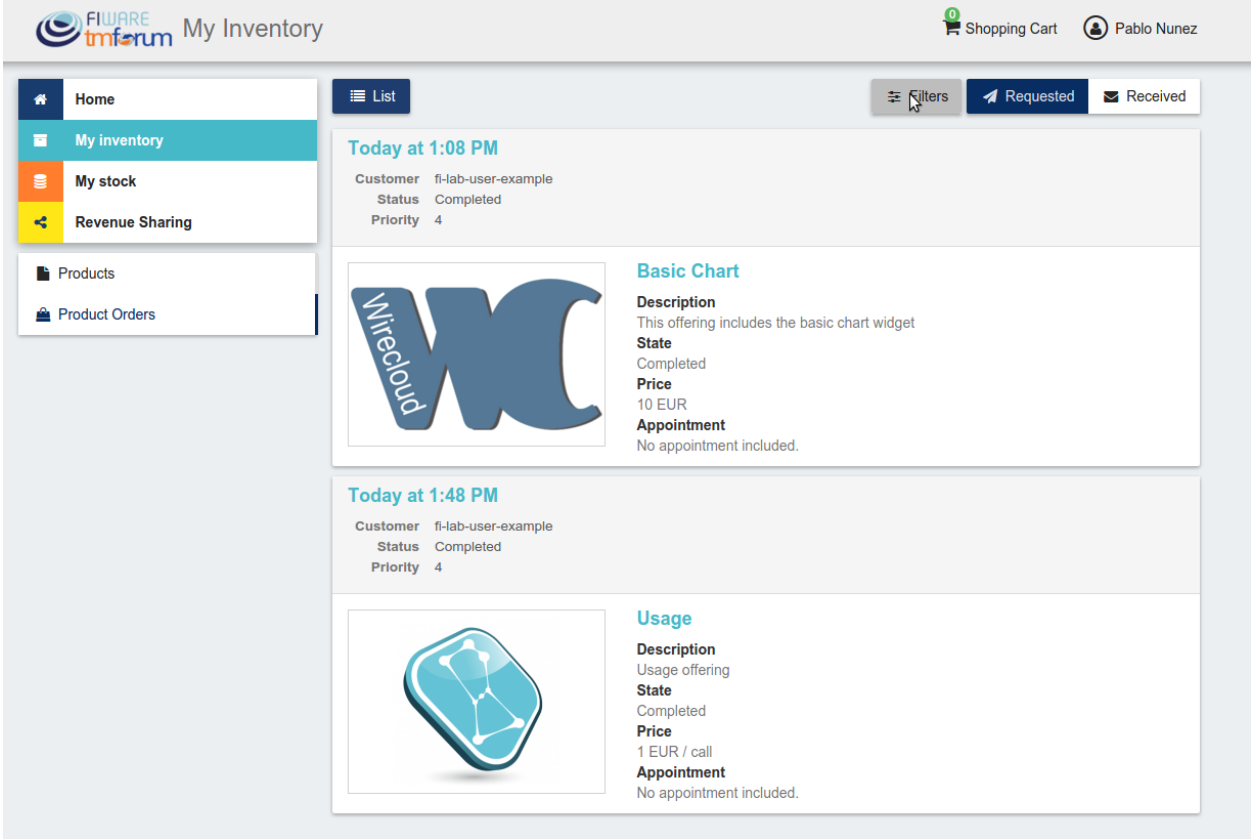

## Manage Requested Orders

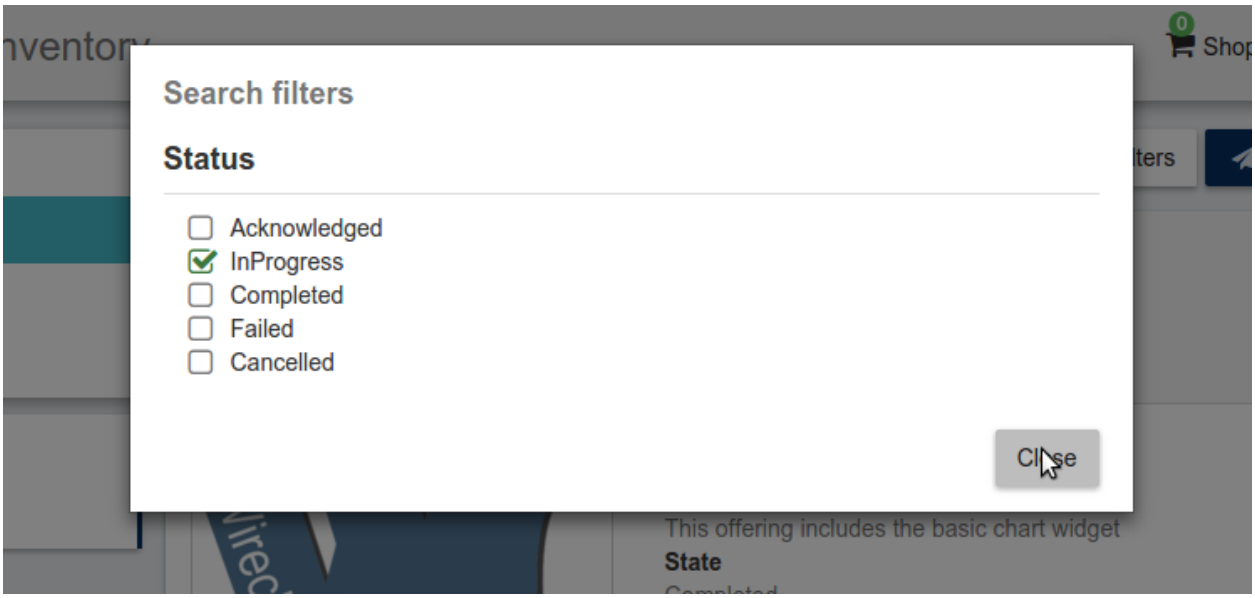
Customers can manage some aspects of the orders they have created. To see your requested orders, go to *My Inventory* and click on *Product Orders*



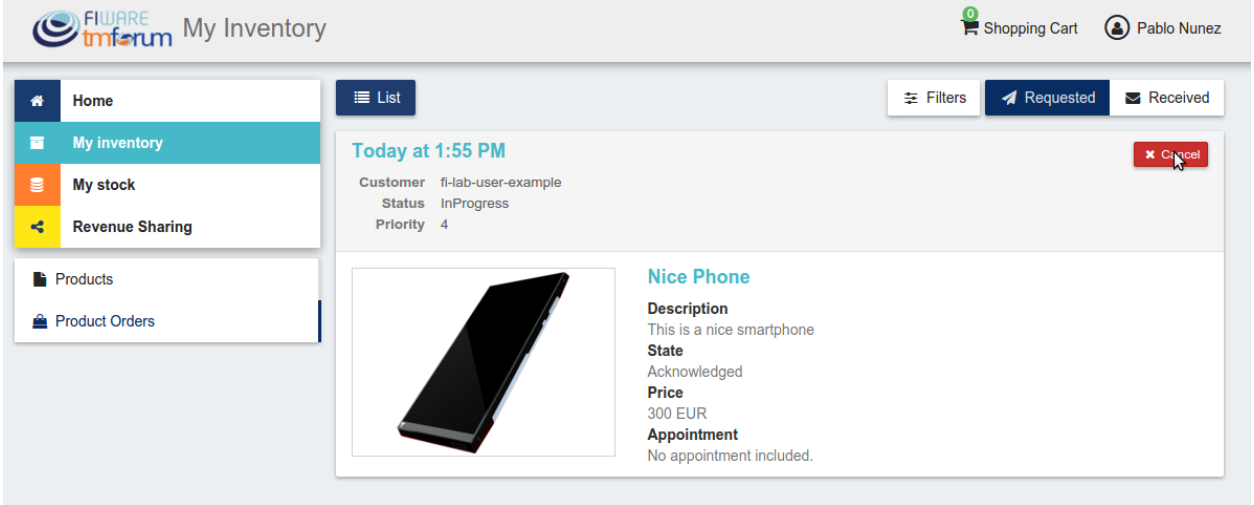

In the displayed view, you can see the orders you have created, which can be filtered by its status. To do that, click on

*Filters*, select the wanted statuses, and click on *Close*

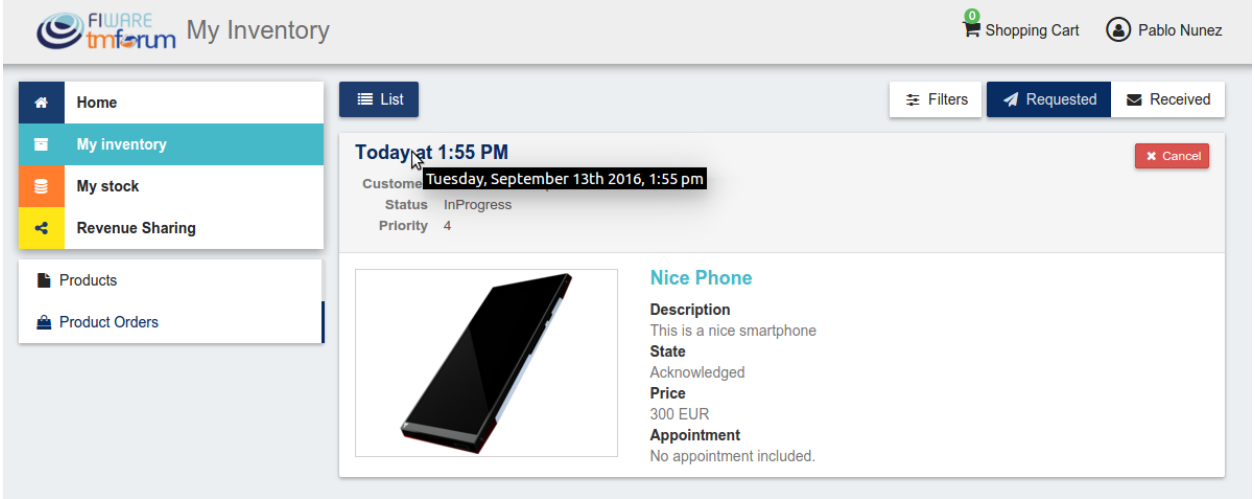


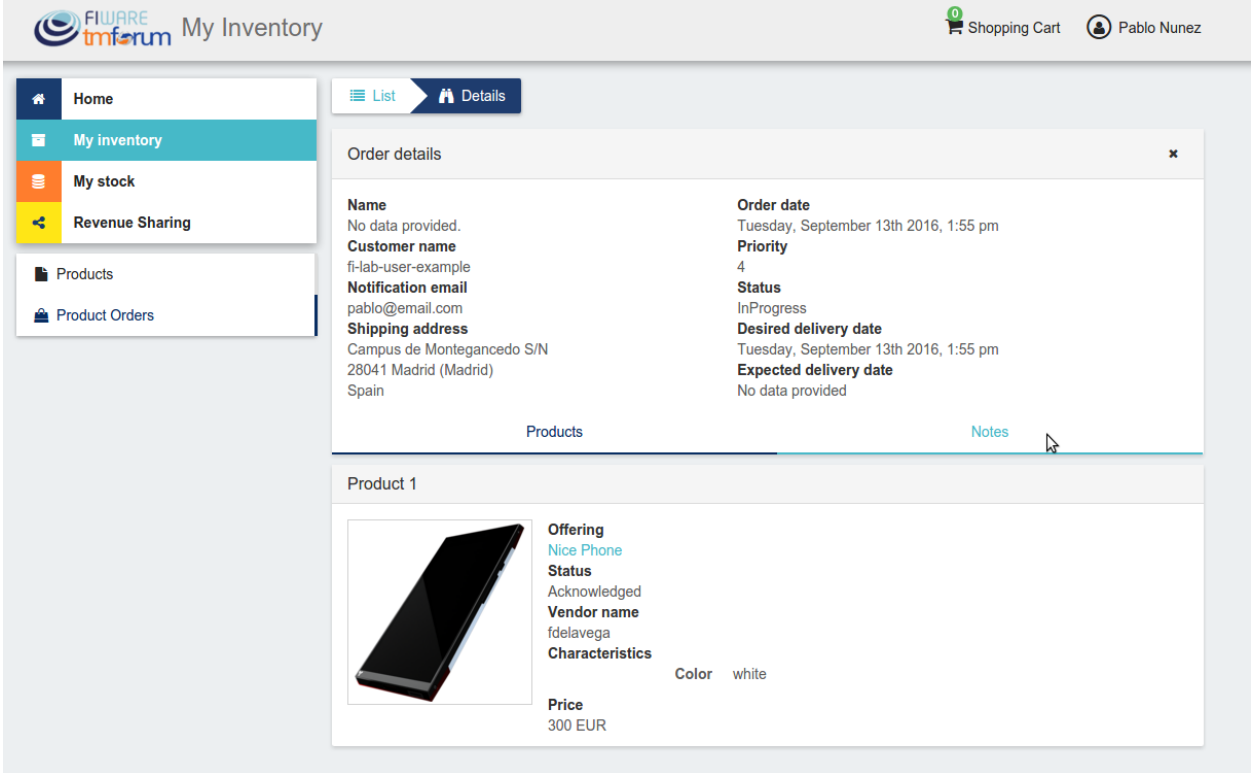

For those orders that include offerings of non digital products, you will be able to cancel them if the seller has not yet started the process. To do that, locate the order to be canceled and click on *Cancel*



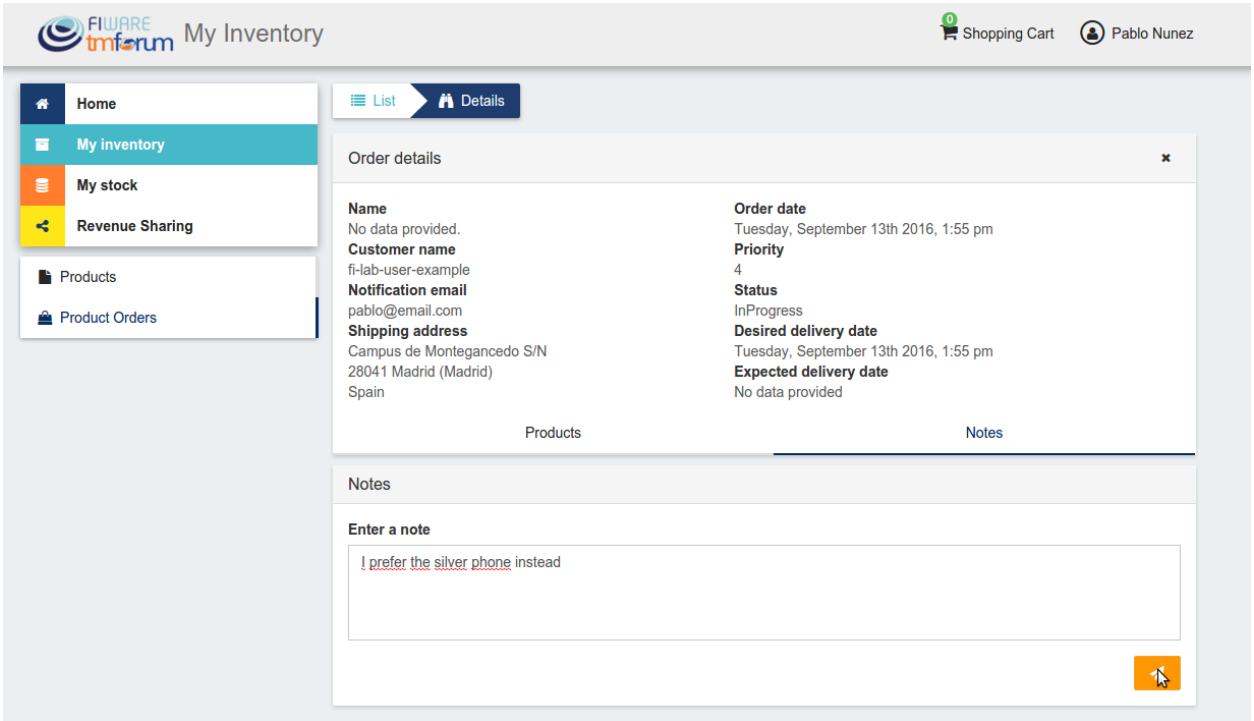

Moreover, you can review the details of the order. To do that click on the date of the order.




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




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



## 1.4 Programmer Guide

### 1.4.1 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
- pull_accounting (optional): This flag is used to indicate that the service defined by the plugin is not pushing accounting information to the usage API of the Business API Ecosystem, but exposing an API that must be queried to retrieve this information.
- form (optional): This field is used to define a custom form that will be displayed for retrieving asset-specific meta data. This field is defined as an object where keys are the name of the metadata property and values define the following information:
  - type: Type of the particular metadata property. Allowed values are *text*, *textarea*, *checkbox* and *select* mapping the form input types to be displayed for retrieving the data.
  - label: Label to be displayed jointly with the form input.
  - default: Default value to be used if no value provided for the property
  - placeholder (text and textarea): Placeholder to be included within the form input
  - options (select): List of valid options when the input is a select. It includes *text* and *value* for each entry.

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"],
    "form": {
        "auth_type": {
            "type": "select",
            "label": "Auth type",
            "options": [{
                "text": "OAuth2",
                "value": "oauth2"
            }, {
                "text": "API Key",
                "value": "key"
```

(continues on next page)

(continued from previous page)

```
            }]
        },
        "token_required": {
            "type": "checkbox",
            "label": "Token required?",
            "default": true
        },
        "auth_server": {
            "type": "text",
            "label": "Auth Server",
            "placeholder": "https://authservice.com/auth"
        }
    }
}
```

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_spec_upgrade(self, asset, asset_t, product_spec):
        pass

    def on_post_product_spec_upgrade(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
```

(continues on next page)

(continued from previous page)

```
    def get_usage_specs(self):
        return []

    def get_pending_accounting(self, asset, contract, order):
        return [], Date()
```

### 1.4.2 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_spec_upgrade**: This method is executed when a digital product is being upgraded (a new version of the asset has been provided). This method can be used in order to validate the new digital asset before saving the upgrade
- **on_post_product_spec_upgrade**: This method is executed when a digital product have been upgraded. This method can be used to send notifications or retrieve new information of the product specification.
- **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
- **get_usage_specs**: This method must be implemented when the flag *pull_accounting* is set to true and must return the list of usage specifications the service is able to monitor. For each usage specification a *name* and a *description* must be provided (e.g name: API Call, description: Number of calls made to...)

- **get_pending_accounting**: This method must be implemented when the flag *pull_accounting* is set to true. This method must implement the client able to access to the service the plugin is defining in order to retrieve pending accounting information for a giving contract. It must return the list of pending accounting including:
  - *date*: Timestamp of the accounting record
  - *unit*: Monitored unit
  - *value*: Actual usage made by the customer

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_spec_upgrade

- **asset**: Asset object that have been upgraded
- **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 upgrade. (The structure of this JSON object can be found in the Open Api documentation)

#### on_post_product_spec_upgrade

- **asset**: Asset object that have been upgraded
- **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 upgrade. (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_suspension

- **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

### get_pending_accounting

- **asset**: Asset object whose usage information has to be retrieved
- **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 contaning 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 Conctract objects, one for earch 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

### 1.4.3 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
```

## 1.5 Plugins Guide

This plugins guide covers the available plugins (defining digital asset types) for the Business API Ecosystem v7.8.0

### 1.5.1 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:

- Basic File: Allows the creation of products by providing files as digital assets. No validations or processing is done
- Basic URL: Allows the creation of products by providing URLs as digital assets. No validations or processing is done
- CKAN Dataset : Allows the monetization of CKAN datasets
- CKAN API Dataset Allows the monetization of CKAN datasets whose resources are served by an external APIs (e.g NGSI Queries) secured with API Umbrella.
- Umbrella Service Allows the monetization of services secured by API Umbrella with FIWARE IDM users and roles.
- 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

### 1.5.2 Available Plugins

#### Basic File and Basic URL

The *Basic File* and *Basic URL* plugins are available at GitHub These plugins are intended to enable the creation of digital products in the Business API Ecosystem without the need of specifying a particular type or validation process. In this regard, these plugins allow the publication of any file or any URL as digital asset respectively, and can be used for the creation of simple file catalogs or for testing the Business API Ecosystem.

These plugins do not implement any event handler.

#### CKAN Dataset and CKAN API Dataset

The *CKAN Dataset* and *CKAN API Dataset* plugins are available in GitHub. These plugins define an asset type intended to manage and monetize datasets offered in a CKAN instance. In particular, these plugins are 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.

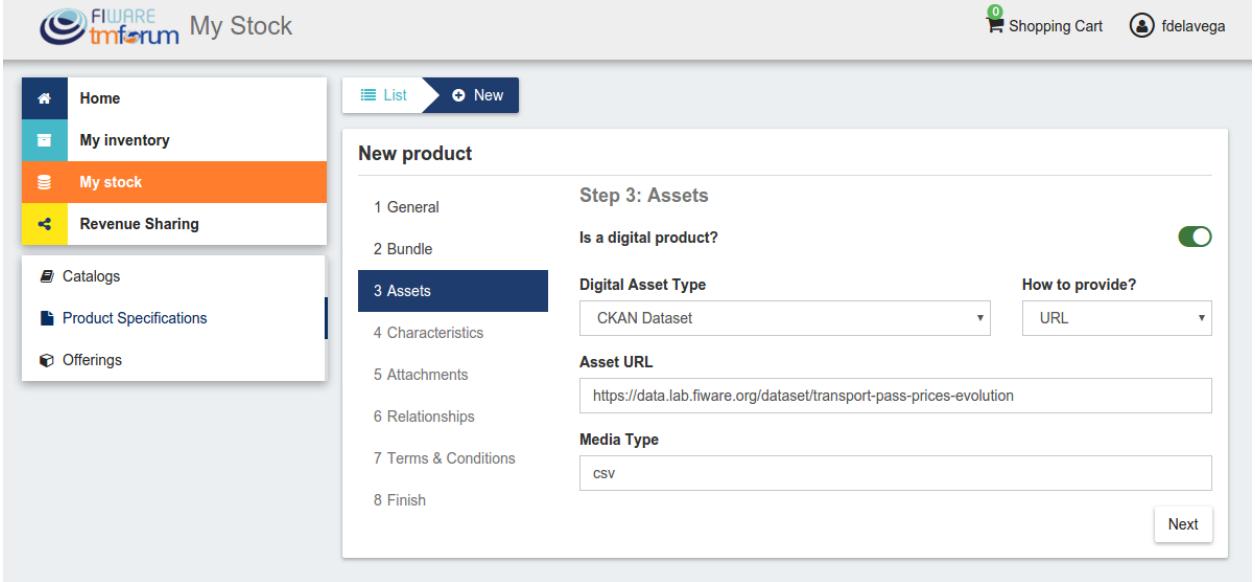
The difference between both plugins is the type of data included as a resource in the CKAN dataset. In particular, *CKAN API Dataset* expects the data to be served by an external API secured with the FIWARE security framework. In this regard, the *CKAN API Dataset* also validates the permissions of the seller in the data service and grants customers access to it using the FIWARE IdM roles and permissions.

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* or *CKAN API Dataset* plugin must be installed in the Business API Ecosystem, since the aforementioned CKAN extension uses the *CKAN Dataset* or *CKAN API Dataset* asset type (depending on the dataset resource) for creating product specifications.

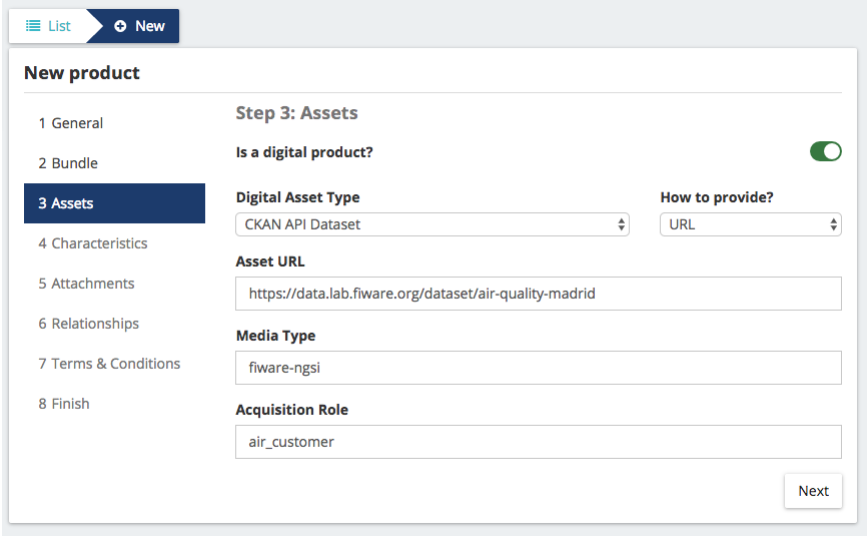
The *CKAN Dataset* plugin only allows to provide the asset with a URL that must match the dataset URL in CKAN.




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.

On the other hand, the *CKAN API Dataset* also requires an *Acquisition role* to be provided. This role is the one that will be granted to customers in the IdM in order to enable their access to the backend service, so the role must exist and define a proper set of permissions for accessing the data.




This plugins 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_post_product_spec_validation**: In this handler, the plugin validates that the API resources included in the

CKAN dataset are valid, the permissions of the seller to offer that services, and that the provided acquisition role exist and is valid.

- **on_post_product_offering_validation**: In this handler the plugin validates that pricing models are supported when creating a pay-per-use offering
- **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.
- **get_pending_accounting**: In this handler, the plugins retrieves pending accounting information when the access to the data has been acquired under a pay-per-use pricing model.

In addition, the *CKAN API Dataset* requires some settings to be configured before being deployed. This settings are available in the *setting.py* file, and are:

- **UMBRELLA_SERVER**: Administration endpoint of the API Umbrella instanceused to sercure backend services
- **UMBRELLA_KEY**: API Key used for accessing to the API Umbrella instance used to secure the backend service
- **UMBRELLA_ADMIN_TOKEN**: Admin token used for accessing to the API Umbrella instance used to secure the backend service
- **KEYSTONE_USER**: Keystone user used for authenticate requests to the FIWARE IdM
- **KEYSTONE_PASSWORD**: Keystone password used for authenticate requests to the FIWARE IdM
- **KEYSTONE_HOST**: Host of the Keystone service of the FIWARE IdM used for authorizing customers
- **IS_LEGACY_IDM**: False if the FIWARE Idm is at least v7.0.0
- **CKAN_TOKEN_TYPE**: Whether CKAN has to be accessed using X-Auth-Token or Authorization headers

In addition, these settings can be configured using environment variables:

- BAE_ASSET_UMBRELLA_SERVER
- BAE_ASSET_UMBRELLA_KEY
- BAE_ASSET_UMBRELLA_TOKEN
- BAE_ASSET_IDM_USER
- BAE_ASSET_IDM_PASSWORD
- BAE_ASSET_IDM_HOST
- BAE_ASSET_LEGACY_IDM
- BAE_ASSET_TOKEN_TYPE

### Umbrella Service

The *Umbrella Service* plugin is available in GitHub. This plugin deines an asset type intended to manage and monetize any HTTP service secured with the combination of a FIWARE IDM for users and roles management and API Umbrella as PEP proxy.

The Umbrella Service plugin allows to provide services in different ways using the options it defined in its metadata form, which can be selected by sellers when registering the product. In particular:

- **Authorization Method**: Whether user access to backend service is controlled using FIWARE IDM roles or API Umbrella native roles
- **Acquisition Role**: Role to be granted to customers
- **Access to sub-paths allowed**: If true, customers will be able to access to any sub-path of the monetized service
- **Additional query strings allowed**: If true, customers will be able to call the service with different query strings as the included in the asset URL
- **Admin API Key**: API key to be used by the BAE to access to the API Umbrella admin API
- **Admin Auth Token**: Admin token to be used by the BAE to access to the Umbrella admin API

Moreover, this plugin support pay-per-use pricing supporting the *api call* unit. The accounting information is retrieved from the API Umbrella logging API using the service details provided as metadata when the product is created.

This plugin implements the following event handlers:

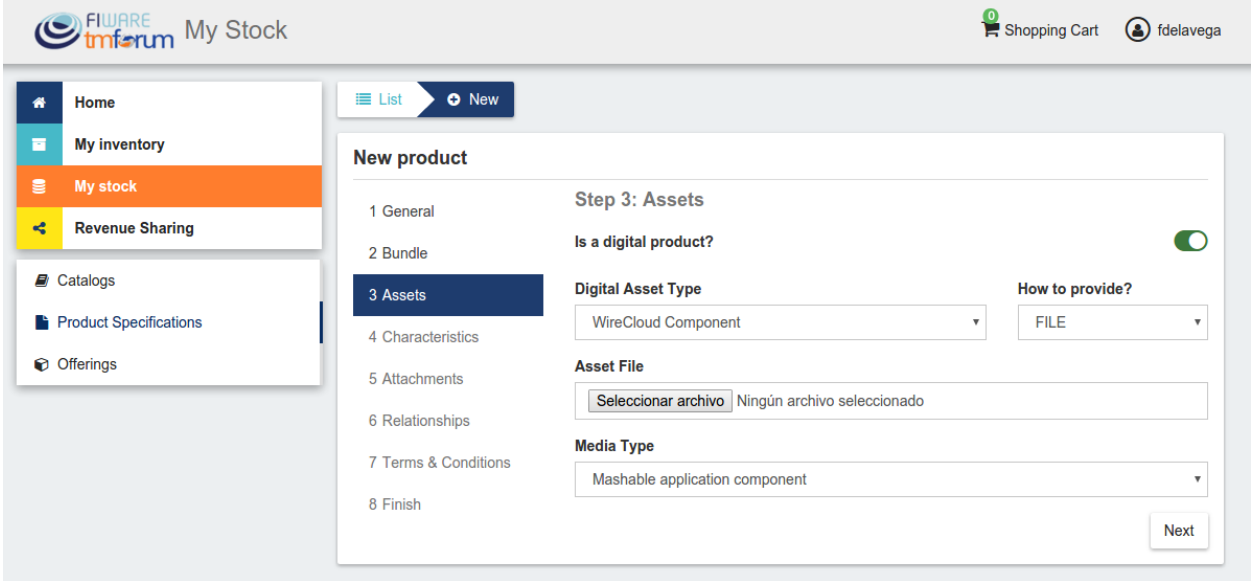
- **on_post_product_spec_validation**: In this event handler the plugin validates all the provided information, including URL, Umbrella credentials and role.
- **on_post_product_offering_validation**: In this event handler the plugin validates that the provided procing model is supported by the plugin (Usage model)
- **on_product_acquisition**: In this event handler the plugin grants access to the customer using the provided role
- **on_product_suspension**: In this event handler the plugin revokes access to the customer removing the provided role
- **get_pending_accounting**: In this event handler the plugin accesses Umbrella API to retrieve the pending accounting information

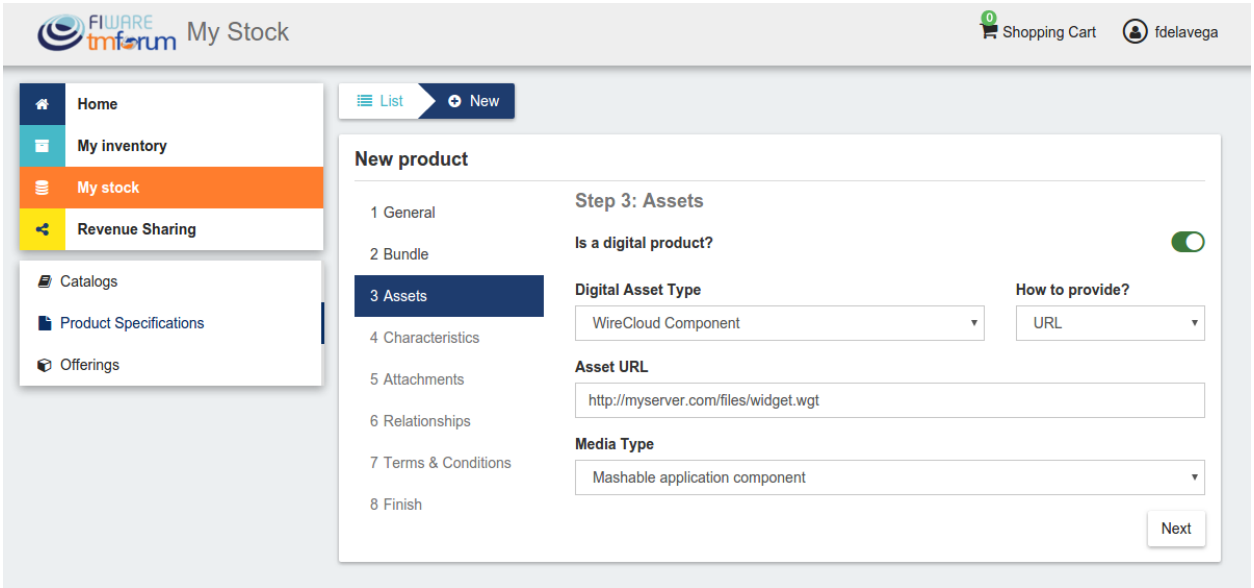
### WireCloud Component

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

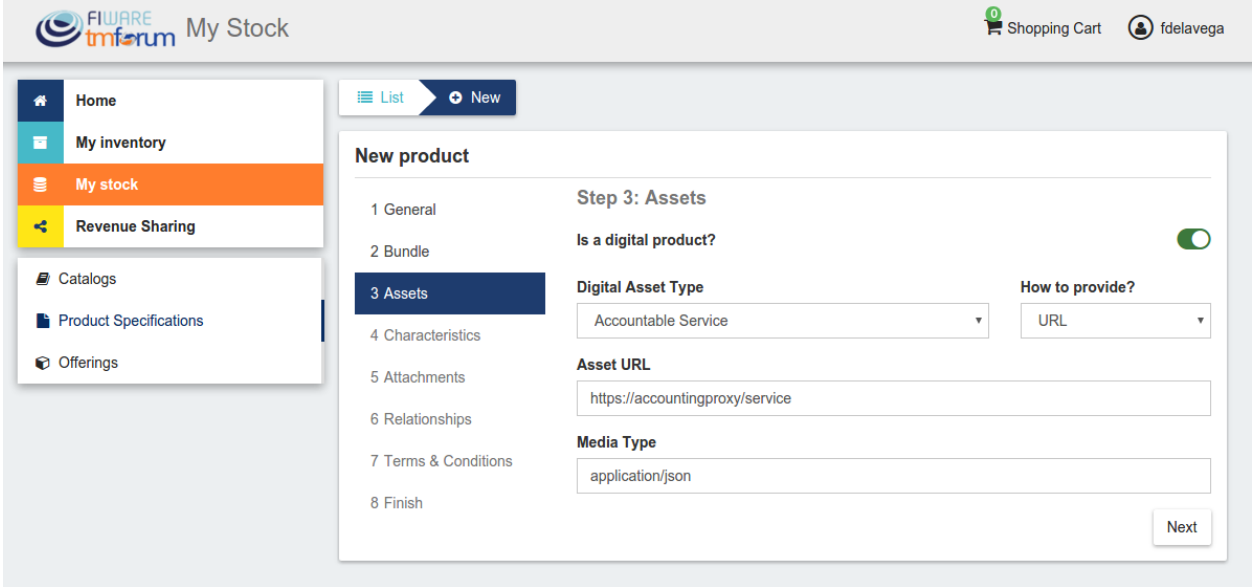
### Accountable Service

**Warning:** This plugin is deprecated, and will not evolve. This plugin has been replaced by Umbrella Service Plugin

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 tyo 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 will be sent.**
  *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 exists 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/contextEntities/Room2/attributes/temperature 1111 GET
```

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": "..."
```

(continues on next page)

(continued from previous page)

```
    }
}
```

- *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.