Business API Ecosystem Documentation

Release latest

September 15, 2016

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 coordinatelly to provide the complete functionality.

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

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

CHAPTER 1

Index

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

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

1.1 Installation and Administration Guide

1.1.1 Introduction

This installation and administration guide covers the Business API Ecosystem version 5.4.0, corresponding to FIWARE release 5. Any feedback on this document is highly welcomed, including bugs, typos or things you think should be included but aren't. Please send them to the "Contact Person" email that appears in the Catalogue page for this GEi. Or create an issue at GitHub Issues

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

1.1.2 Installation

Requirements

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

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

TM Forum APIs and RSS requirements

- Java 8
- Glassfish 4.1+
- MySQL 5.5

Charging Backend requirements

- Python 2.7
- MongoDB
- wkhtmltopdf

Logic Proxy requirements

- NodeJS 4.5.0 (Including NPM)

Installing basic dependencies

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

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

Installing basic dependencies using the script

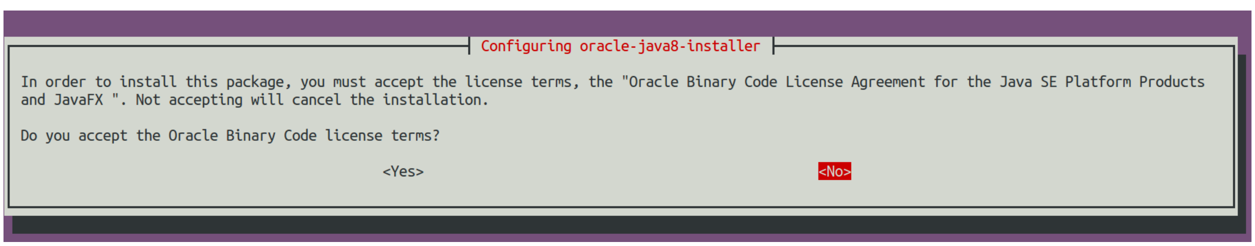
In order to automate the installation of the basic dependencies, the script *resolve-basic-dep.sh* has been provided. This script, located in the directory *scripts/*, installs all the needed packages for Ubuntu, Debian, and CentOS systems.

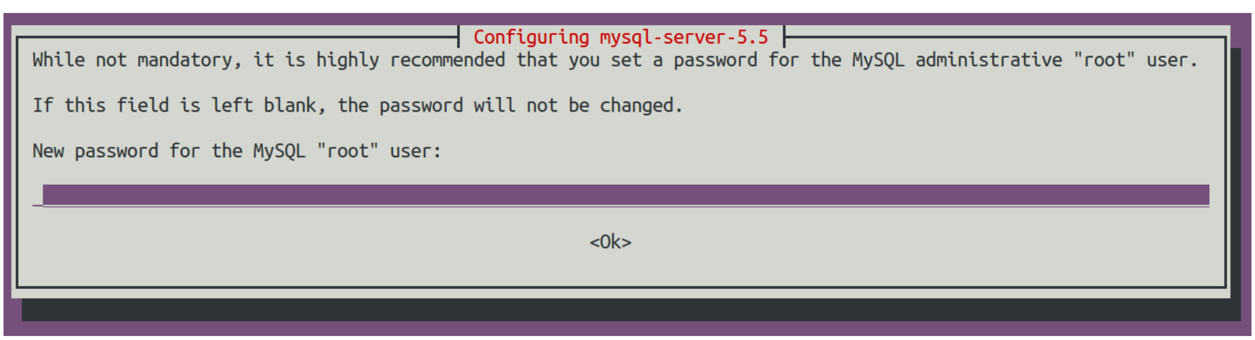
Additionally, this script creates a directory */opt/biz-ecosystem* where Glassfish 4.1 and Node 4.5.0 are downloaded.

To execute the script, run the following command from the *scrips/* directory of the project

```
$ sudo ./resolve-basic-dep.sh
```

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





Installing basic dependencies manually

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

APIs dependencies **Java 8 Debian/Ubuntu**

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

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

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

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

Then Java 8 can be installed using the following commands:

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

Java 8 CentOS 7

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

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

Then Java can be installed using *alternatives*

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

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

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

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

MySQL and Maven CentOS 7

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

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

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

Then, for installing Maven

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

Glassfish The next step is downloading and extracting Glassfish

```
$ wget http://download.java.net/glassfish/4.1.1/release/glassfish-4.1.1.zip
$ unzip glassfish-4.1.1.zip
```

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

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

$ gunzip mysql-connector-java-5.1.39.tar.gz
$ tar -xvf mysql-connector-java-5.1.39.tar

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

Charging Backend dependencies **Python 2.7 Debian/Ubuntu**

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

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

Python 2.7 CentOS

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

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

MongoDB Debian/Ubuntu

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

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

MongoDB CentOS 7

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

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

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

Wkhtmltopdf Debian/Ubuntu

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

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

Wkhtmltopdf CentOS 7

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

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

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

```
$ wget https://nodejs.org/dist/v4.5.0/node-v4.5.0-linux-x64.tar.xz
$ tar -xvf node-v4.5.0-linux-x64.tar.xz
```

Installing the Business API Ecosystem

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

Installing the Business API Ecosystem using the script

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

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

Debian/Ubuntu

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

CentOS 7

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

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

```
$ export PATH=$PATH:/opt/biz-ecosystem/glassfish4/glassfish/bin
$ export PATH=$PATH:/opt/biz-ecosystem/node-v4.5.0-linux-x64/bin

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

Note: Including the previous command is your .bashrc file, prevents you to have to execute them each time

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

Debian/Ubuntu

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

CentOS 7

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

Finally, during the deployment of the RSS API, the script saves the properties file in the default RSS properties directory. Since this directory is */etc/default/rss*, it is required to have root privileges to create it. In this way, this directory must exist and must be accessible by the user executing the script. To do that

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

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

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

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

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

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

- **clone**: Downloads from GitHub the different components of the Business API Ecosystem
- **maven**: Compiles the downloaded APIs using Maven
- **tables**: Creates the required databases in MySQL
- **persistence**: Builds persistence.xml files of the different APIs
- **pools**: Creates database pools in Glassfish
- **resources**: Creates database resources in Glassfish
- **redeploy**: Deploys APIs and RSS war files in Glassfish
- **proxy**: Installs proxy Node libs
- **charging**: Installs charging Python libs

Installing the Business API Ecosystem Manually

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

- Catalog Management API
- Product Ordering Management API
- Product Inventory Management API

- Party Management API
- Customer Management API
- Billing Management API
- Usage Management API

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

```
$ git clone https://github.com/FIWARE-TMForum/DSPRODUCTCATALOG2.git
$ cd DSPRODUCTCATALOG2
$ git checkout v5.4.0
```

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.o
    <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-crea
        </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 selected database name to the previuos command.

Once that that 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.Driv
```

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

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

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

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

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

```
$ mvn install
```

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

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

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

Installing the RSS The RSS sources can be found in GitHub

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

```
$ git clone https://github.com/FIWARE-TMForum/business-ecosystem-rss.git
$ cd business-ecosystem-rss
$ git checkout v5.4.0
```

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/DSRevenueSh
```

Installing the Charging Backend The Charging Backend sources can be found in in GitHub

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

```
$ git clone https://github.com/FIWARE-TMForum/business-ecosystem-charging-backend.git
$ cd business-ecosystem-charging-backend
$ git checkout v5.4.0
```

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

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

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

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

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

Installing the Logic Proxy The Charging Backend sources can be found in 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
$ git checkout v5.4.0
```

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

```
$ npm install
```

1.1.3 Configuration

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

Configuring the RSS

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

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

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

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

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

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

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

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

Configuring the Charging Backend

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

```
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'
```

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

- INVENTORY: URL of the inventory API including its path
- ORDERING: URL of the ordering API including its path
- BILLING: URL of the billing API including its path
- RSS: URL of the RSS including its path
- USAGE: URL of the Usage API including its path
- AUTHORIZE_SERVICE: Complete URL of the usage authorization service. This service is provided by the logic proxy, and is used to generate API Keys to be used by accounting systems when providing usage information.

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

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

This setting contains the following fields:

- ENGINE: Database engine, must be fixed to django_mongodb_engine

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

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

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

This settings contain the following values:

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

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

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

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

---

Note: The email configuration 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
```

This settings contain the following values:

- PAYPAL_CLIENT_ID: Id of the application provided by PayPal
- PAYPAL_CLIENT_SECRET: Secret of the application provided by PayPal

- MODE: Mode of the connection. It can be *sandbox* if using the PayPal sandbox for testing the system. Or *live* if using the real PayPal APIs

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

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

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

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

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

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

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

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

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

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

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

```
$ ./manage.py crontab show

$ ./manage.py crontab remove
```

Configuring the Logic Proxy

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

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

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

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

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

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

```
    port: 443
};
```

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

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

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

Additionally, the proxy is the component that acts as the front end of the Business API Ecosystem, both providing a web portal, and providing the endpoint for accessing to the different APIs. In this regard, the Proxy has to have the OAUth2 configuration of the FIWARE IDM.

To provide OAUth2 configuration, an application has to be created in an instance of the FIWARE IdM (e.g *https://account.lab.fiware.org*), providing the following information:

- URL: http|https://<proxy_host>:<proxy_port>
- Callback URL: http|https://<PROXY_HOST>:<PROXY_PORT>/auth/fiware/callback
- Create a role *Seller*

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

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

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

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

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

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

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

```
config.appHost = 'localhost';

config.endpoints = {
    'catalog': {
        'path': 'DSProductCatalog',
        'port': '8080'
    },
    'ordering': {
         'path': 'DSProductOrdering',
        'port': '8080'
    },

    ...
```

The setting *config.appHost* contain the host where the APIs are running. On the other hand, *config.endpoints* contains the specific configuration of each of the APIs, including its *path*, and its *port*.

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 this fields

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

1.1.4 Final steps

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

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

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

1.1.5 Running the Business API Ecosystem

Running the APIs and the RSS

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

```
$ asadmin start-domain
```

Running the Charging Backend

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

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

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

Or in background

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

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

Running the Logic Proxy

The Logic Proxy can be started using Node as follows

```
$ node server.js
```

Or if you want to start it in background:

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

1.1.6 Installing Asset Plugins

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

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

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

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

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

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

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

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

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

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

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

- WireCloud Component: Allows the monetization of WireCloud components, including Widgets, operators, and mashups

- Accountable Service : Allows the monetization of services protected by the Accounting Proxy, including Orion Context Broker queries
- CKAN Dataset : Allows the monetization of CKAN datasets

1.1.7 Sanity check Procedures

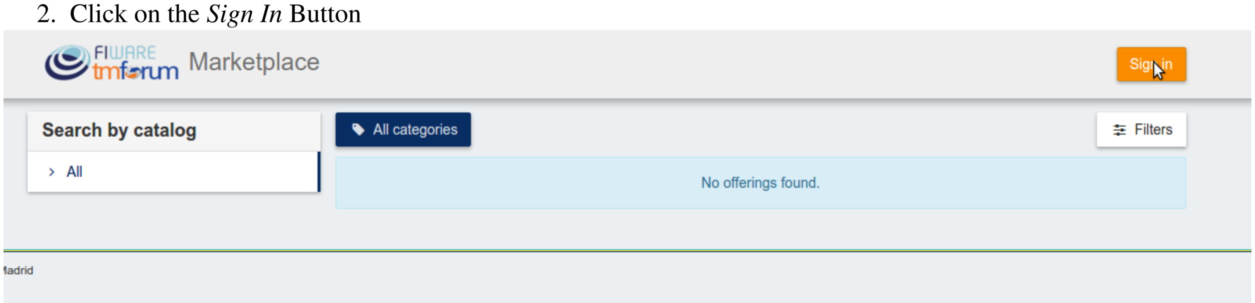
The Sanity Check Procedures are the steps that a System Administrator will take to verify that an installation is ready to be tested. This is therefore a preliminary set of tests to ensure that obvious or basic malfunctioning is fixed before proceeding to unit tests, integration tests and user validation.

End to End Testing

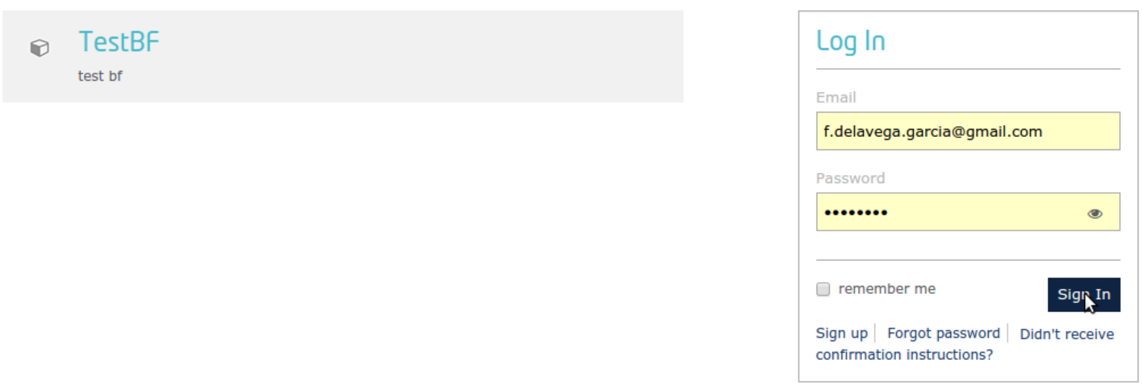
Please note that the following information is required before starting with the process: * The host and port where the Proxy is running * A valid IdM user with the *Seller* role

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

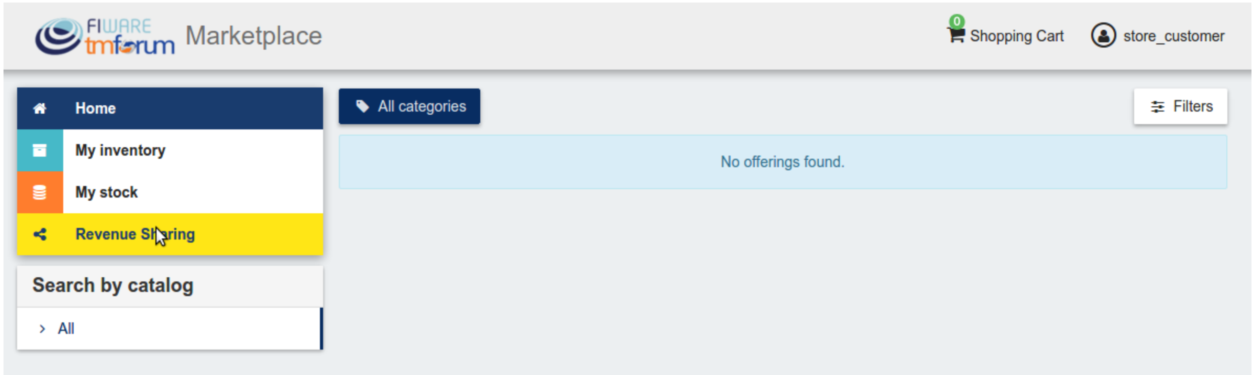
1. Open a browser and enter to the Business API Ecosystem
2. Click on the *Sign In* Button



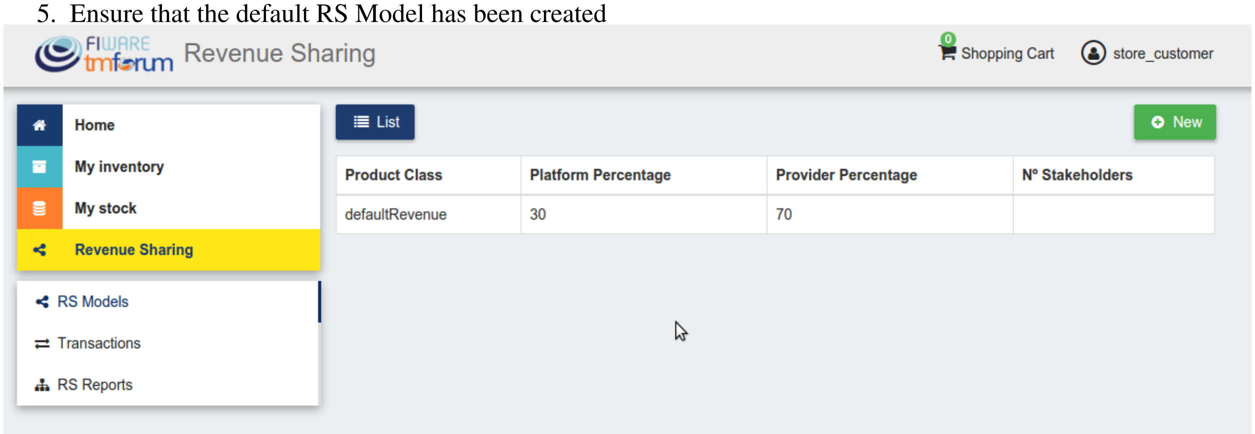
3. Provide your credentials in the IdM page



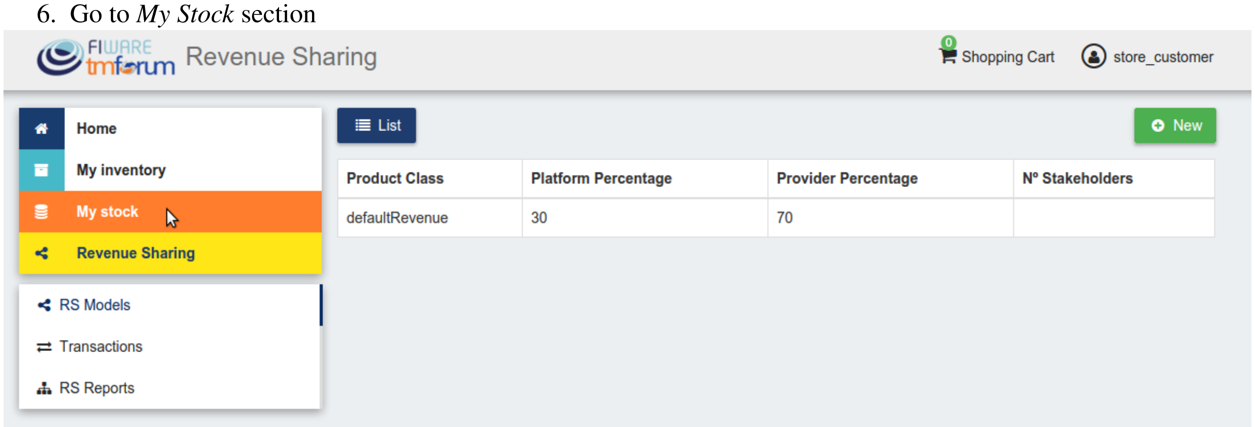
4. Go to the *Revenue Sharing* section



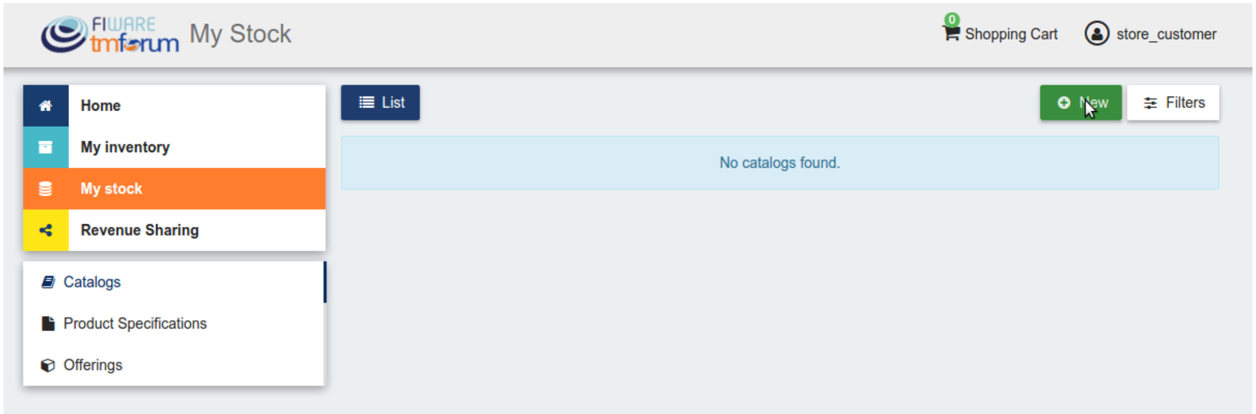
5. Ensure that the default RS Model has been created



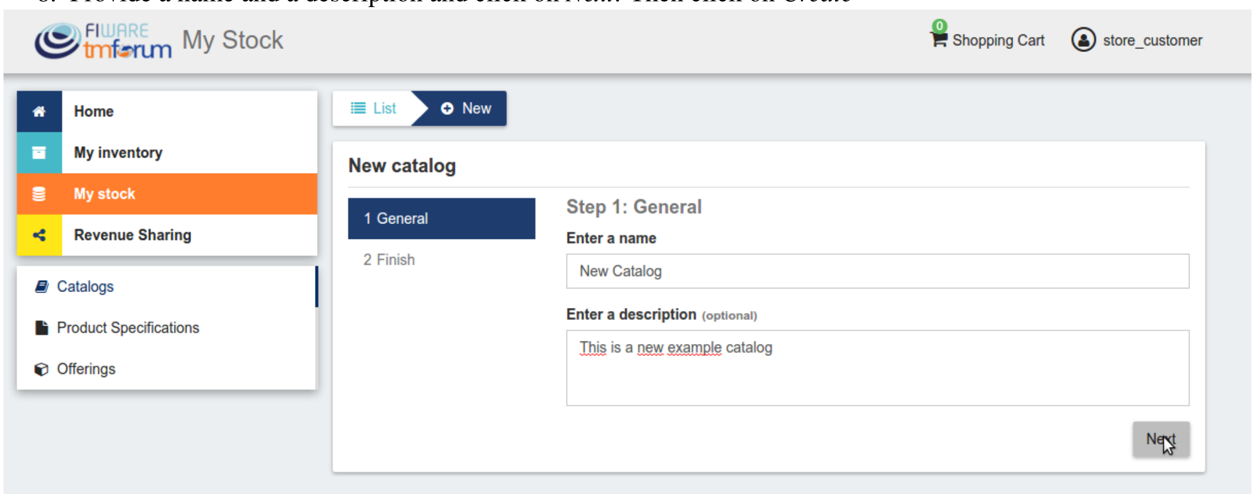
6. Go to *My Stock* section

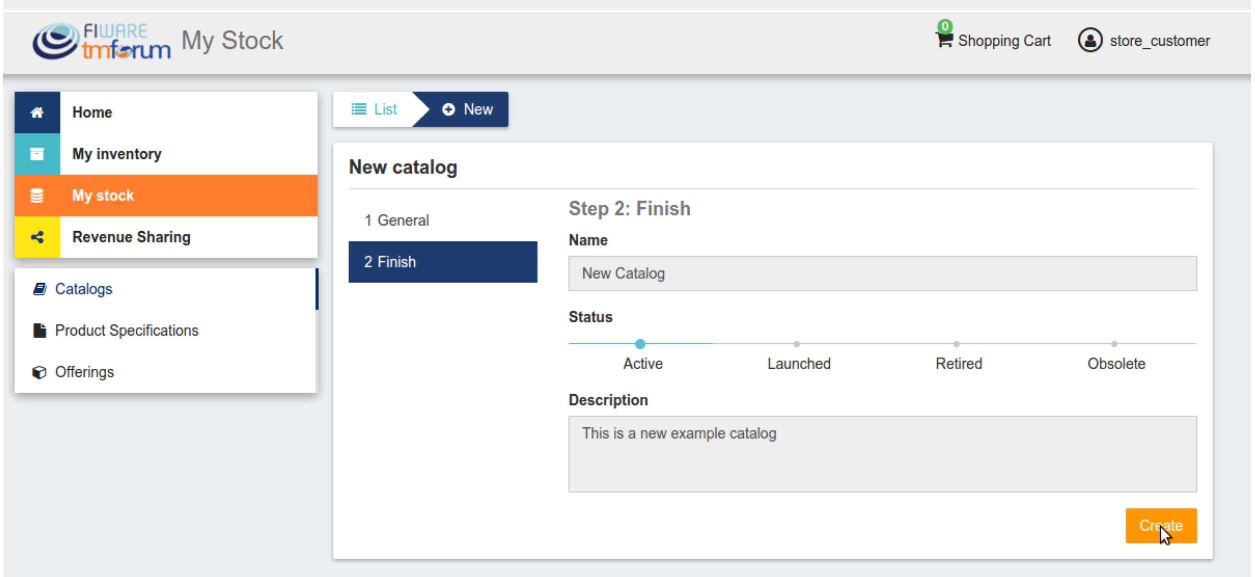


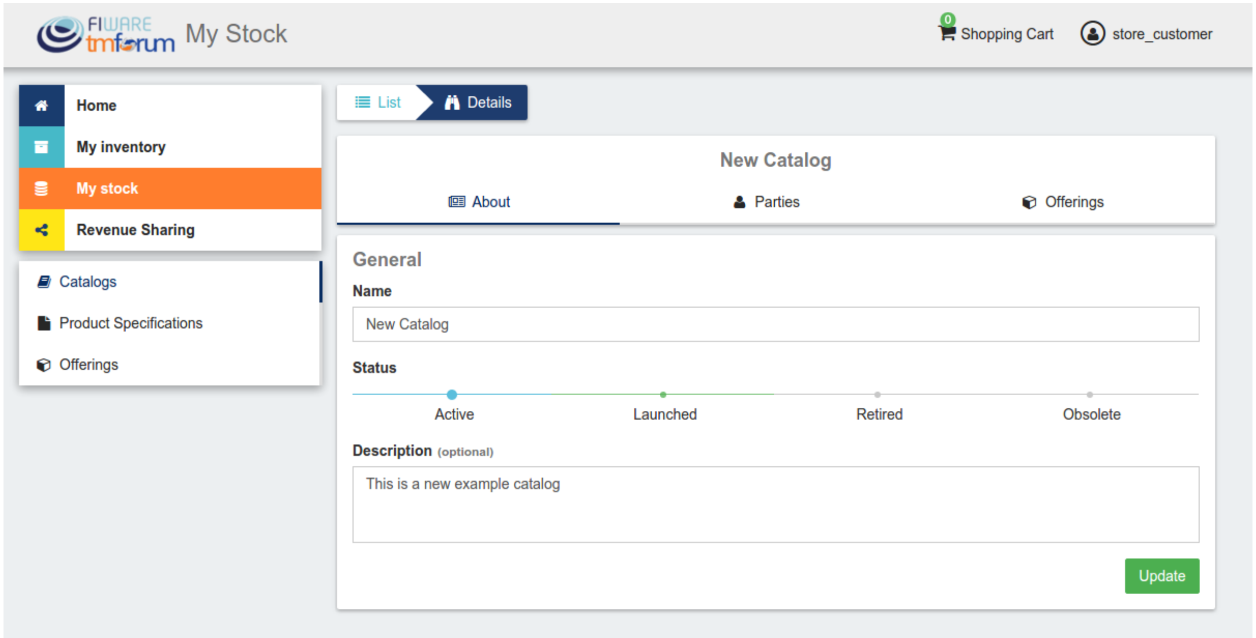
7. Click on *New* for creating a new catalog



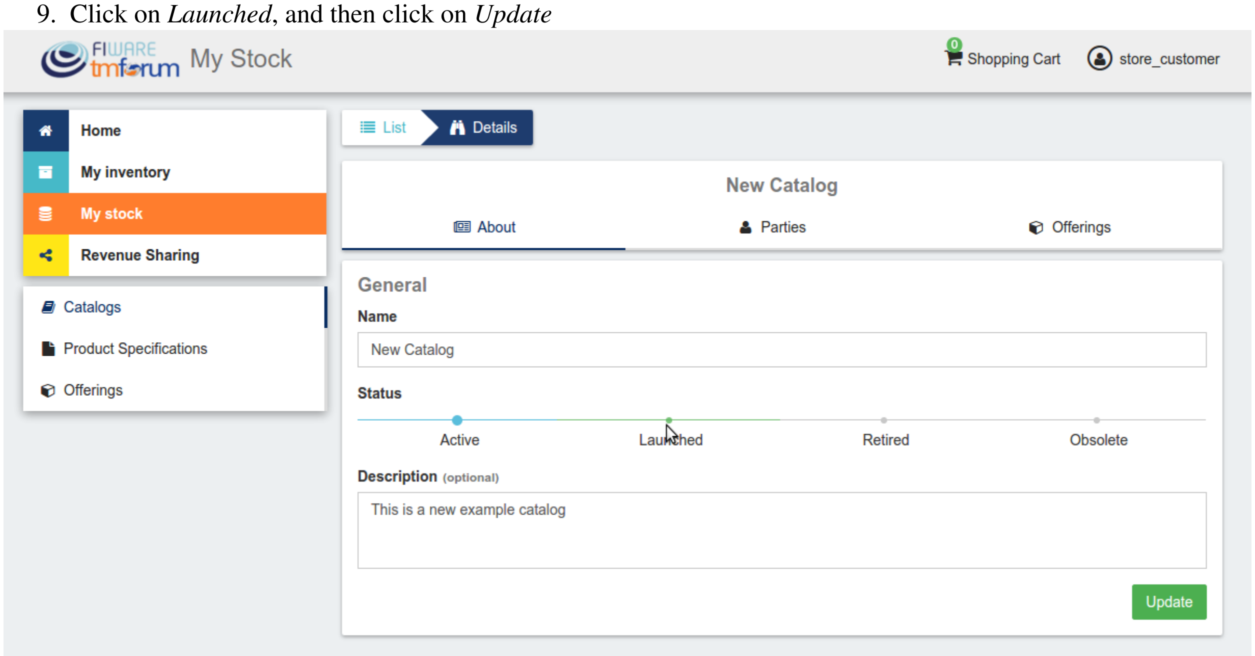
8. Provide a name and a description and click on *Next*. Then click on *Create*

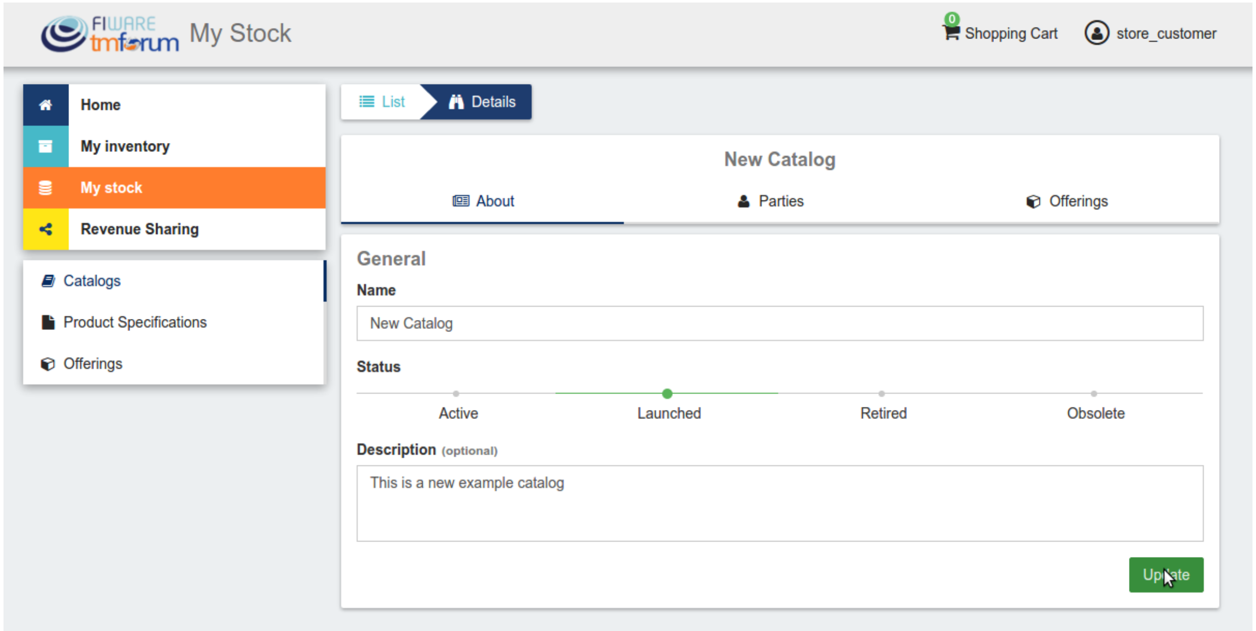




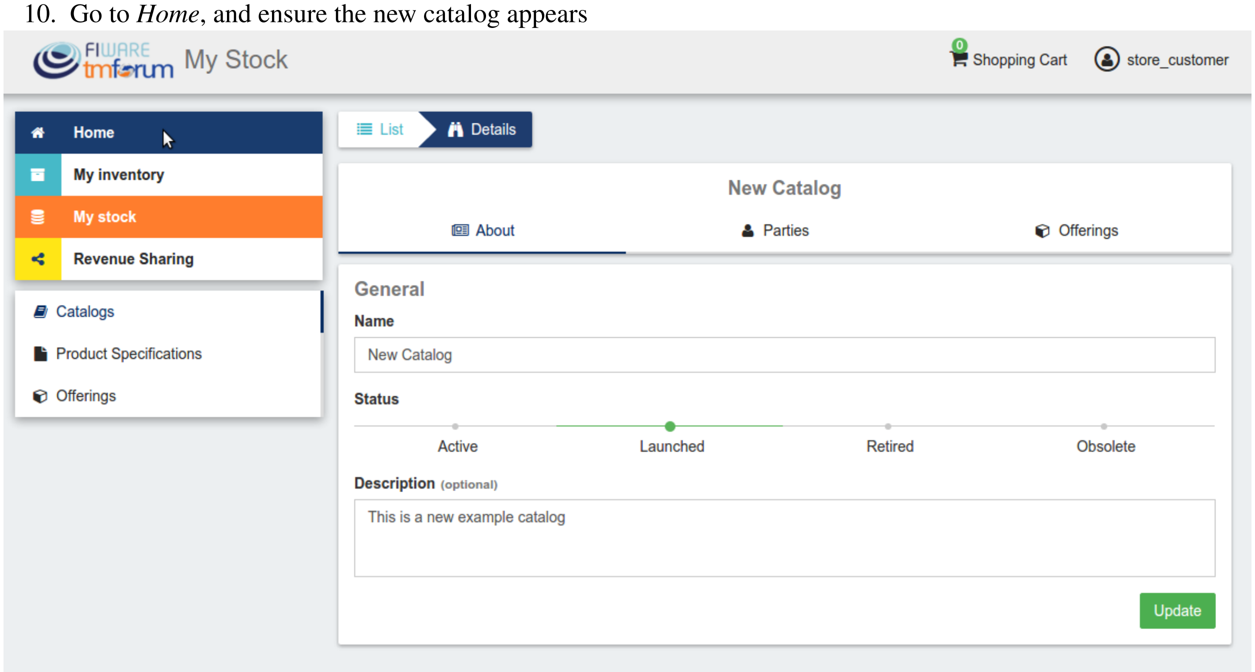


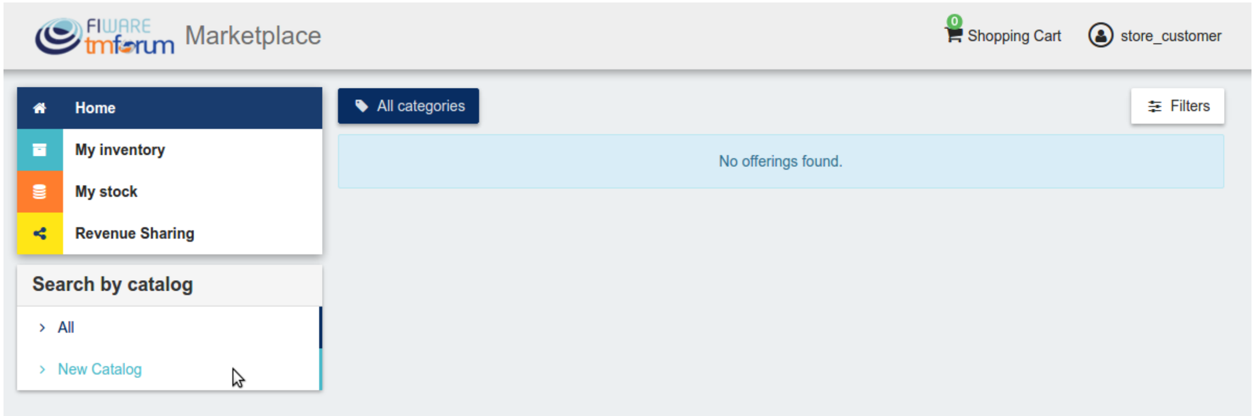
9. Click on *Launched*, and then click on *Update*





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





List of Running Processes

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

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

It should show something similar to the following:

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

Network interfaces Up & Open

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

```
$ sudo netstat -nltp
```

The expected results must be something similar to the following:

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

Databases

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

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

You should see something similar to:

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

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

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

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

mysql>
```

For MongoDB, execute the following command:

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

You should see something similar to:

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

1.1.8 Diagnosis Procedures

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

Resource Availability

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

Remote Service Access

N/A

Resource Consumption

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

- Glassfish main memory consumption should be between 500 MB and 2048 MB
- MongoDB main memory consumption should be between 30 MB and 500 MB
- Pyhton main memory consumption should be between 30 MB and 200 MB

- Node main memory consumption should be between 30 MB and 200 MB
- MySQL main memory consumption should be between 30 MB and 500 MB

I/O Flows

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

1.2 User and Programmer Guide

1.2.1 Introduction

This user and programmer guide covers the Business API Ecosystem version 5.4.0, corresponding to FIWARE release 5. Any feedback on this document is highly welcomed, including bugs, typos or things you think should be included but aren't. Please send them to the "Contact Person" email that appears in the Catalogue page for this GEi. Or create an issue at GitHub Issues

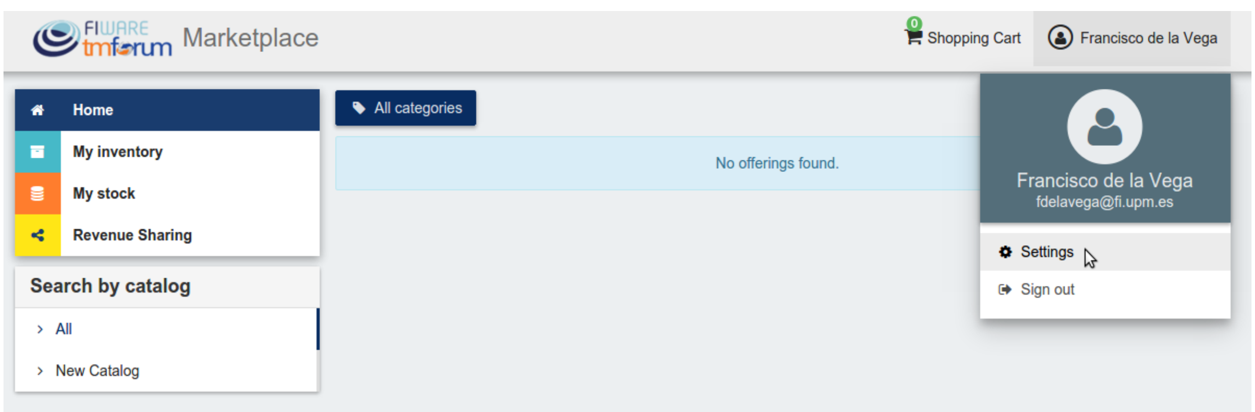
1.2.2 User Guide

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

Profile Configuration

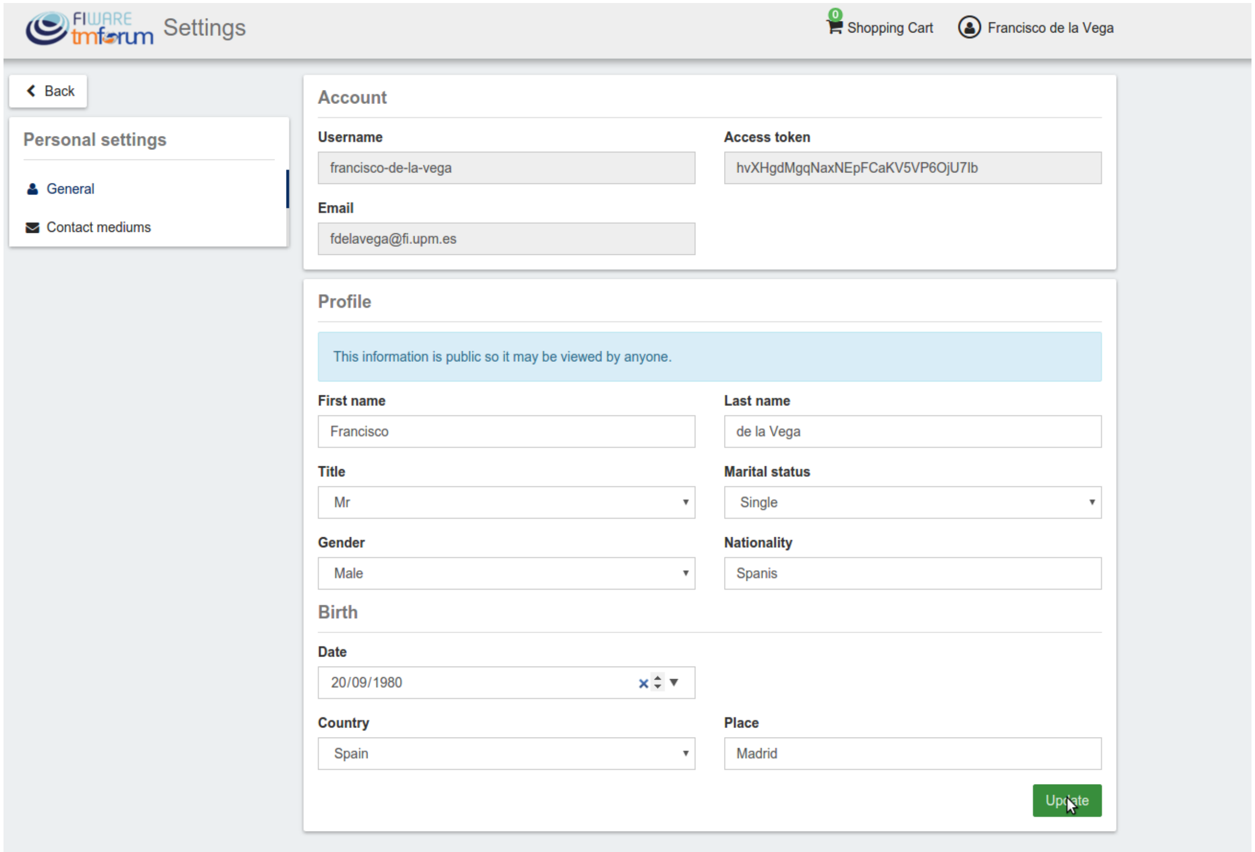
All the users of the system can configure their profile, so they can configure their personal information as well as their billing addresses and contact mediums.

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



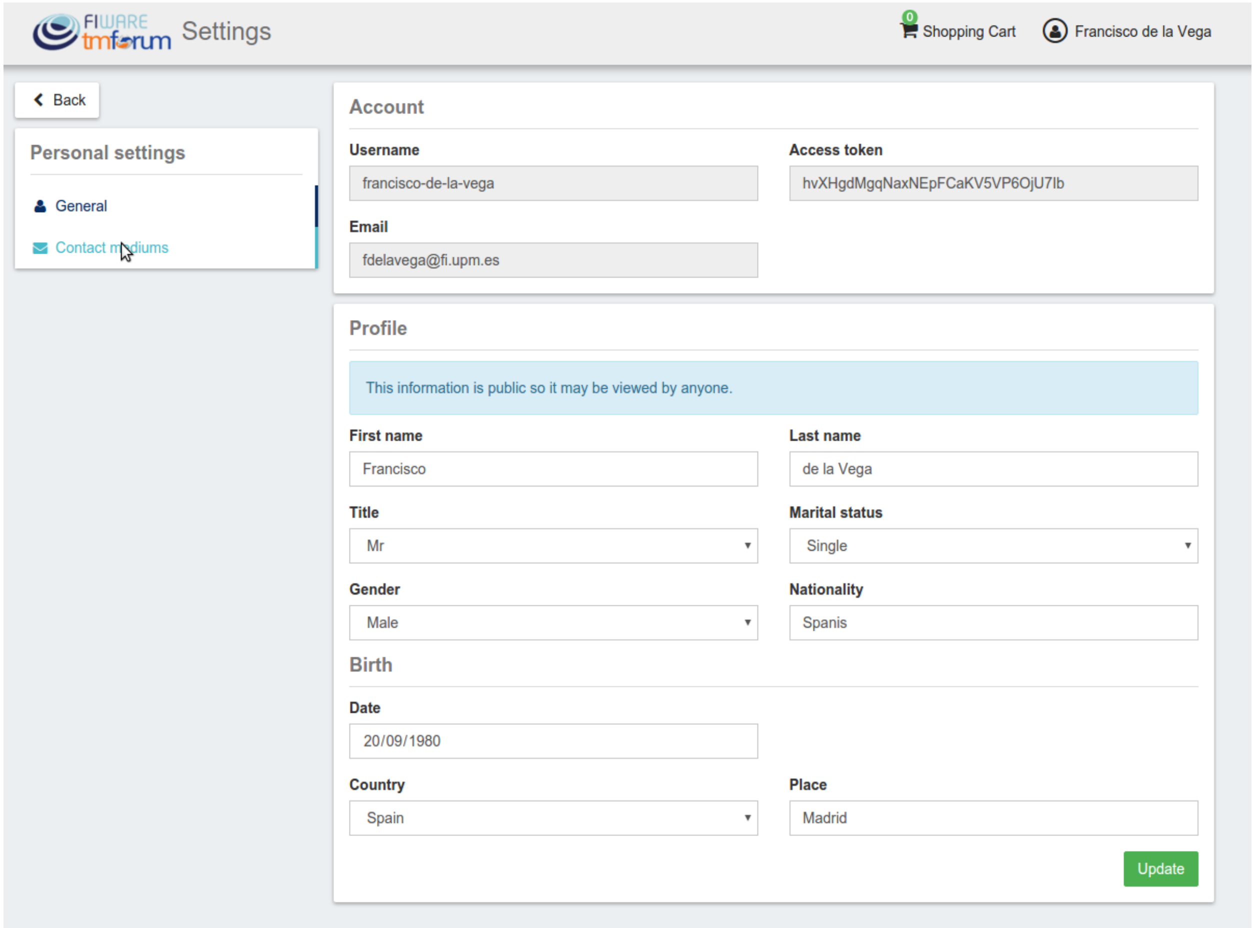
In the displayed view, it can be seen that some information related to account is already included (*Username*, *Email*, *Access token*). This information is the one provided by the IdM after the login process.

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



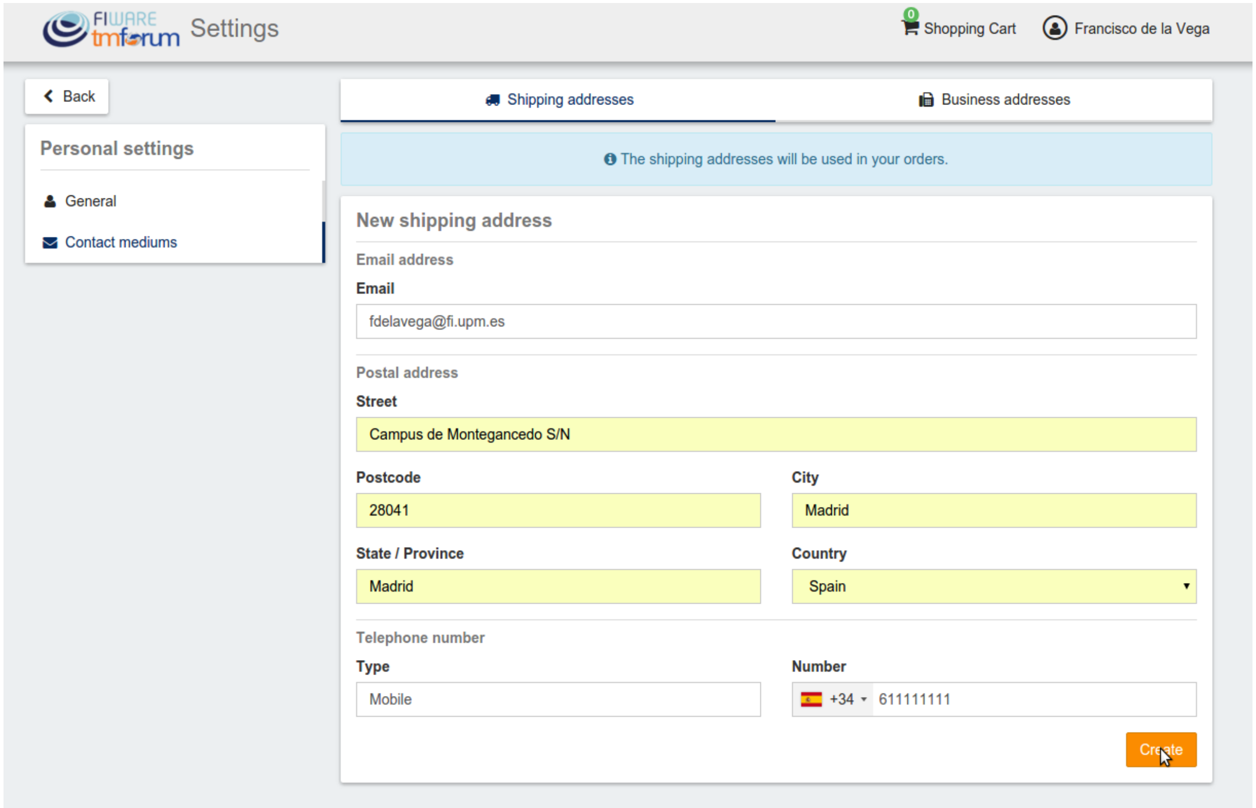
Note: Only the *First name* and *Last name* fields are mandatory

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

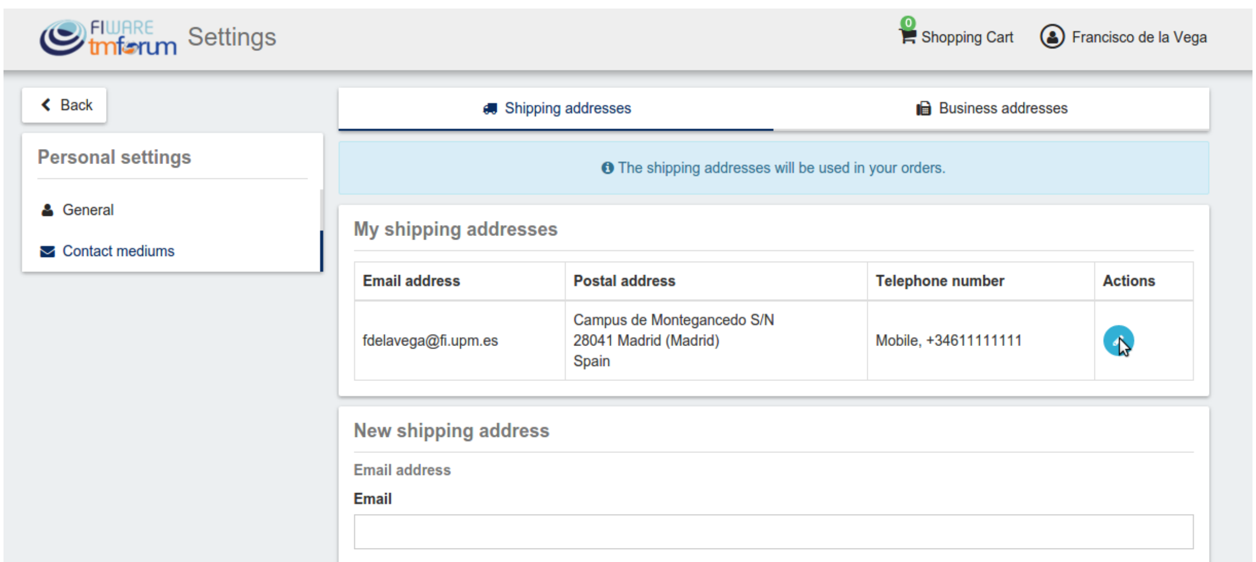


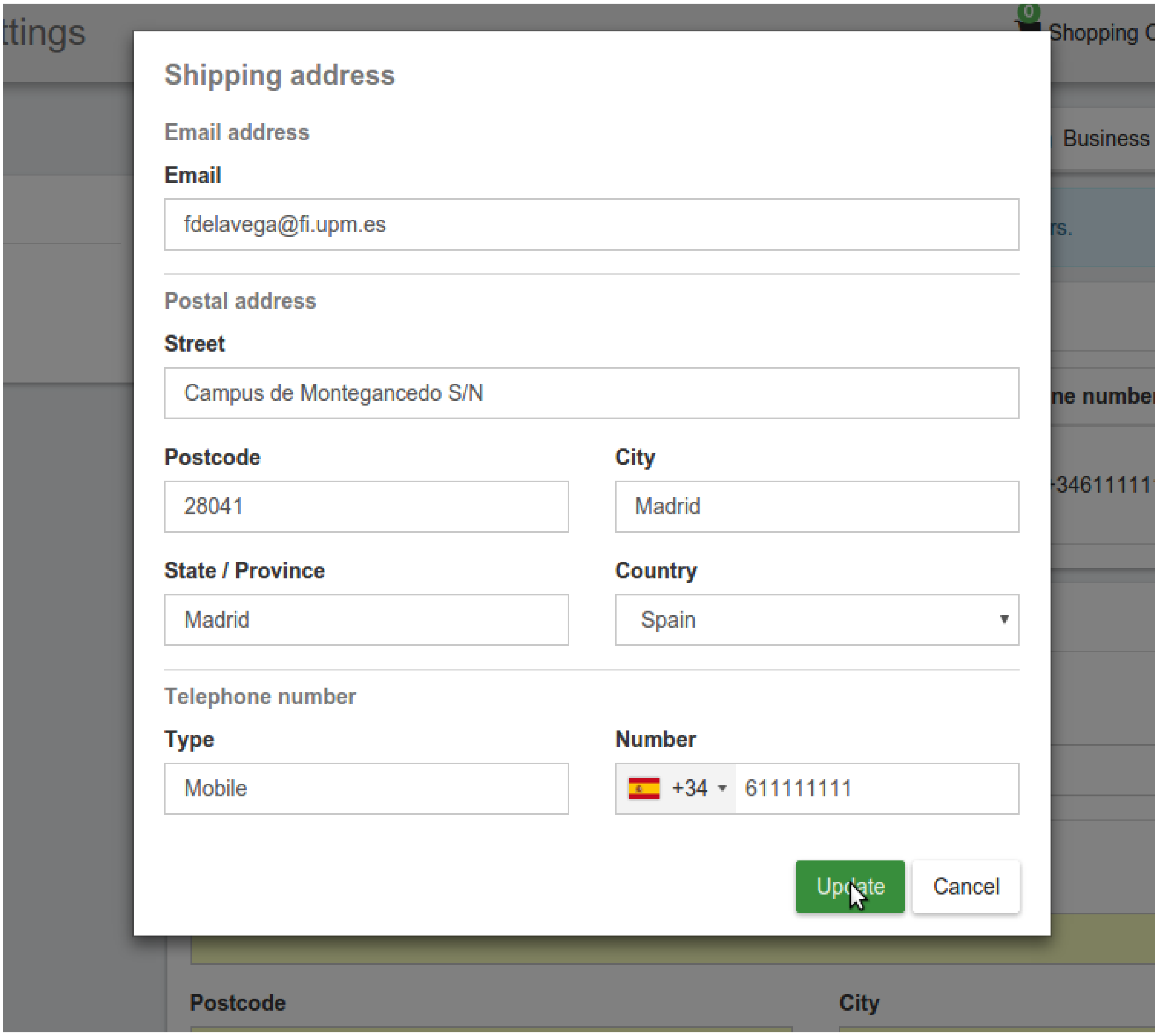
In the *Contact Medium* section, there are two different tabs. On the one hand, the *Shipping addresses* tab, where you can register the shipping addresses you will be able to use when creating order and purchasing products.

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

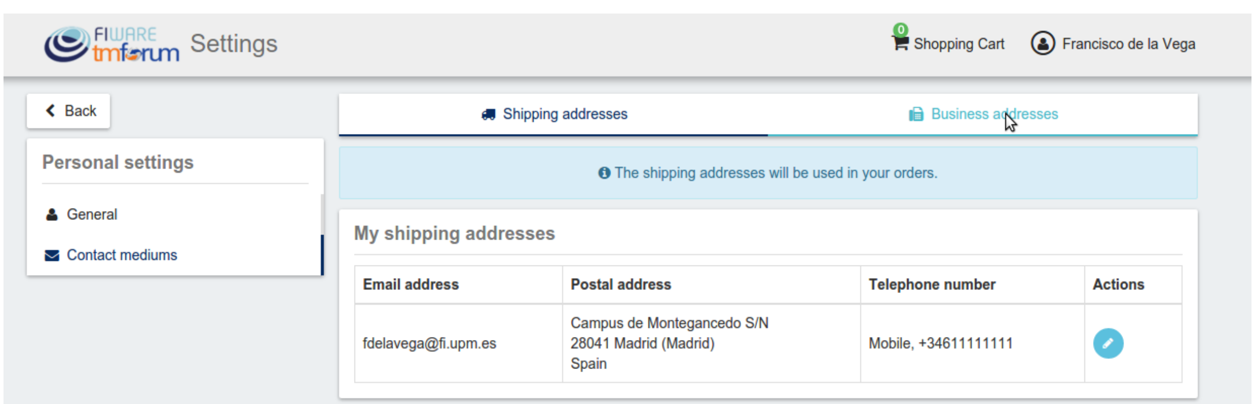


Once created, you can edit the address by clicking on the *Edit* button of the specific address, and changing the wanted fields.

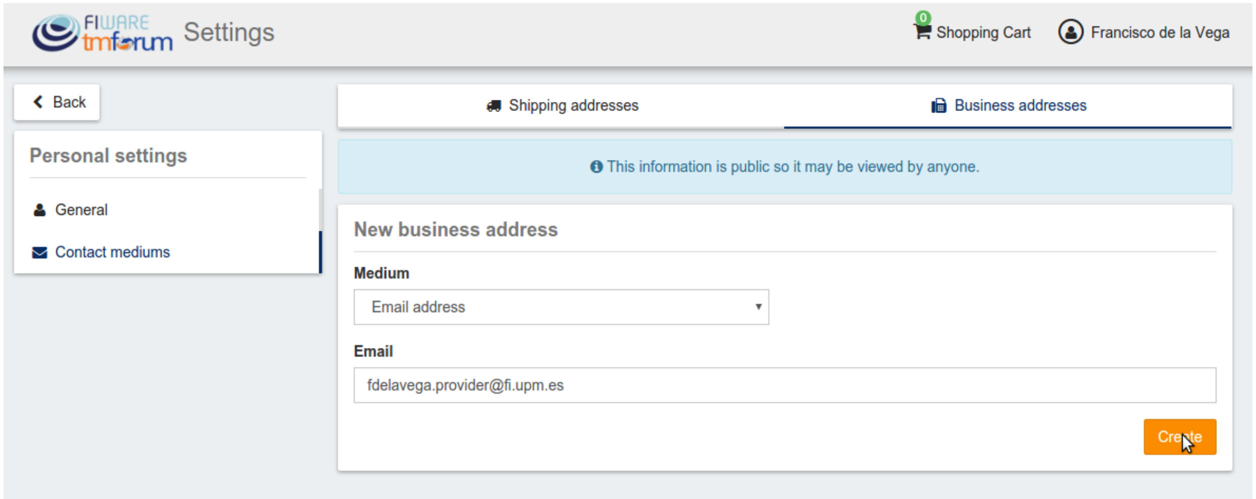


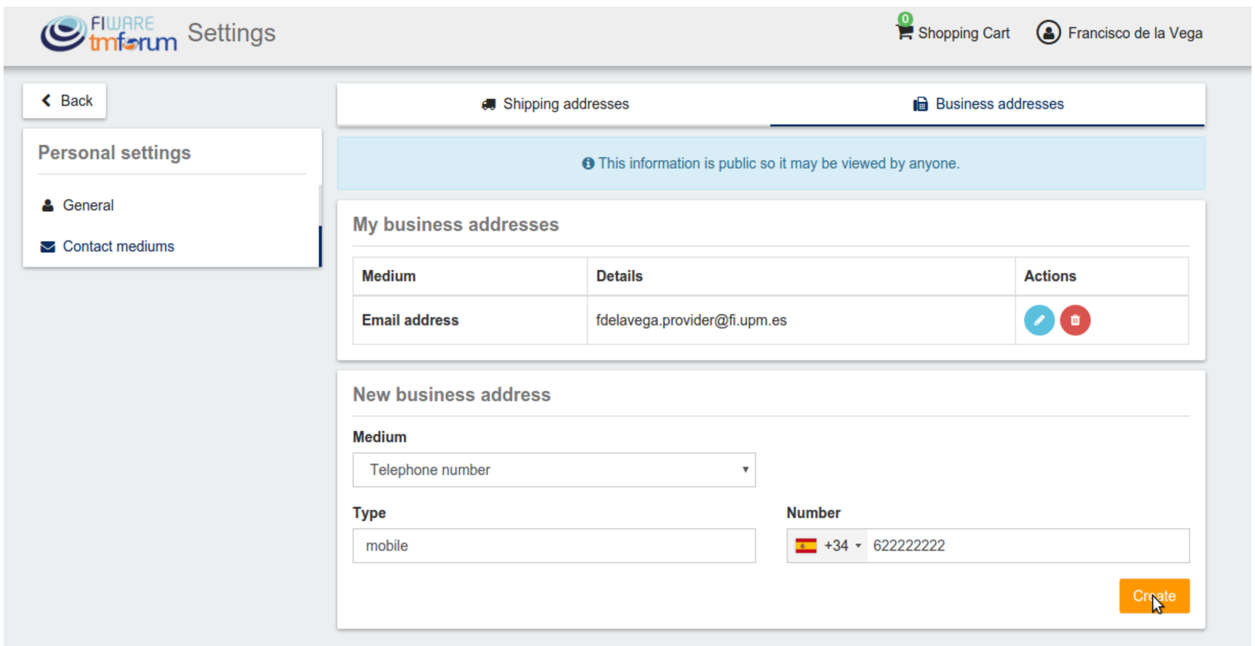


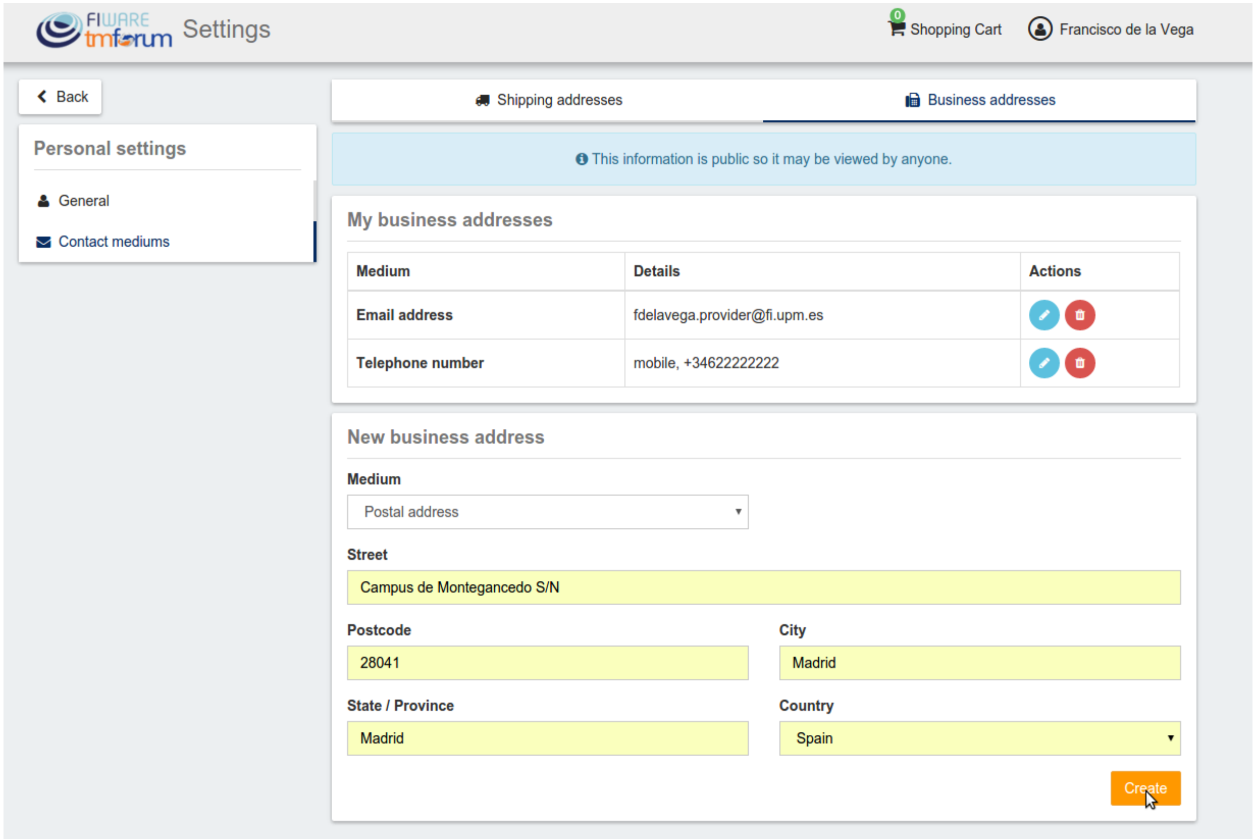
On the other hand, if you have the *Seller* role you can create *Business Addresses*, which can be used by your customers in order to allow them to contact you.



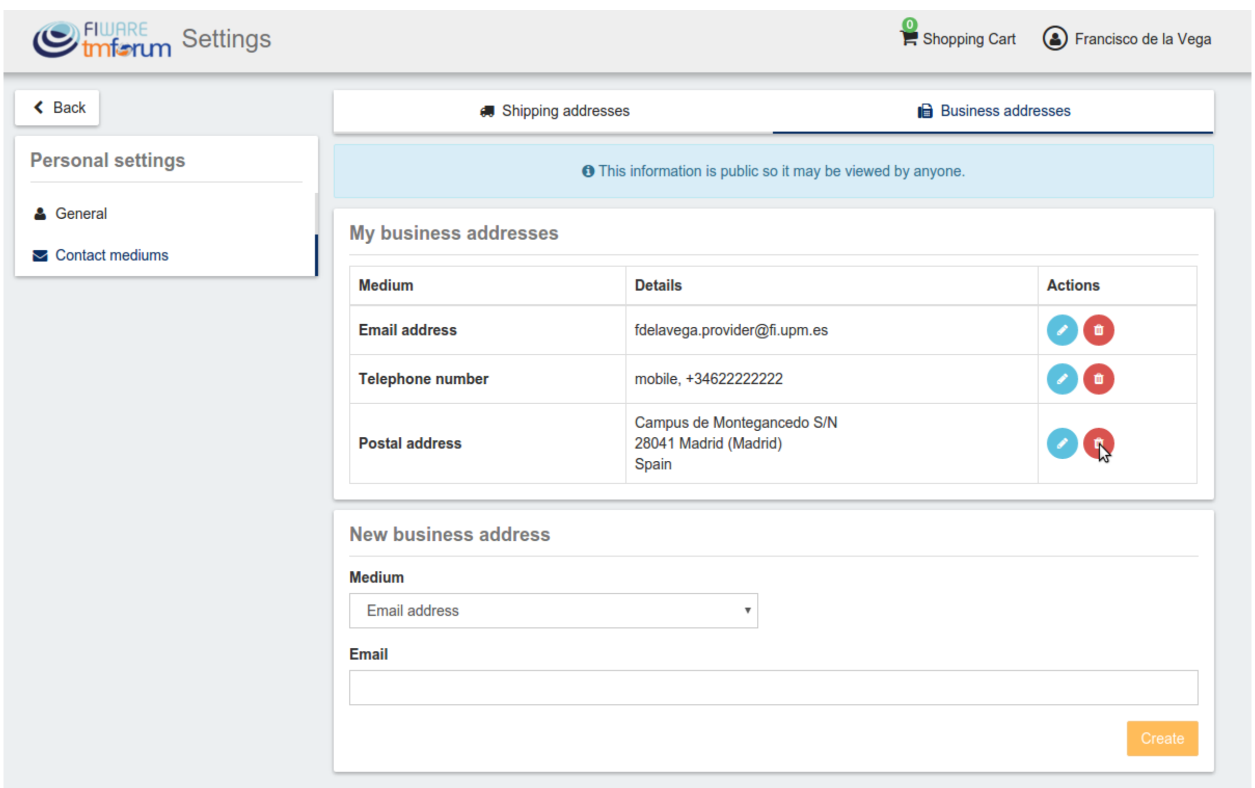
In the *Business Addresses* tab you can create, different kind of contact mediums, including emails, phones, and addresses. To create a contact medium, fill in the fields and click on *Create*


Settings
0
Shopping Cart
Francisco de la Vega
Back
Personal settings
General
Contact mediums
Shipping addresses
Business addresses
This information is public so it may be viewed by anyone.
New business address
Medium
Email address
Email
fdelavega.provider@fi.upm.es
Create


Settings
0
Shopping Cart
Francisco de la Vega
Back
Personal settings
General
Contact mediums
Shipping addresses
Business addresses
This information is public so it may be viewed by anyone.
My business addresses
Medium
Details
Actions
Email address
fdelavega.provider@fi.upm.es
New business address
Medium
Telephone number
Type
mobile
Number
+34
622222222
Create

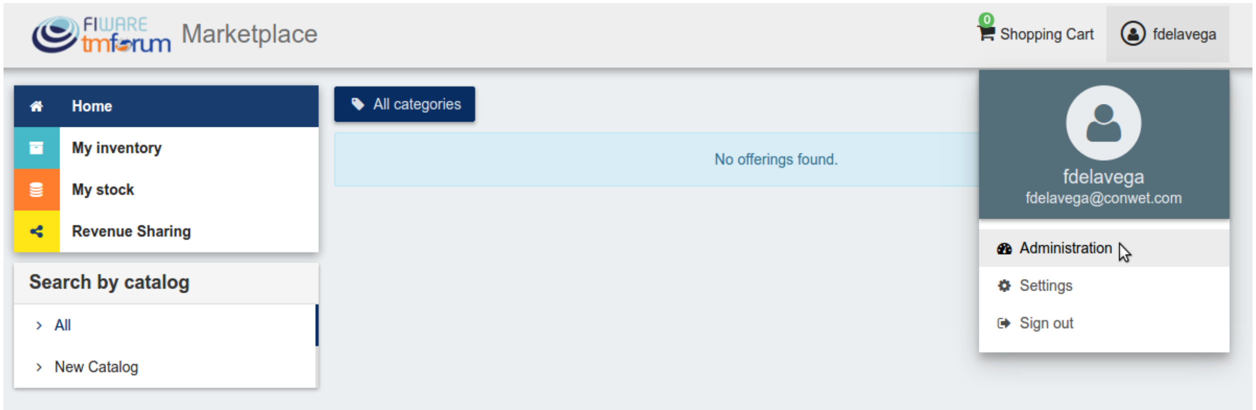


You can *Edit* or *Remove* the contact medium by clicking on the corresponding button



Admin

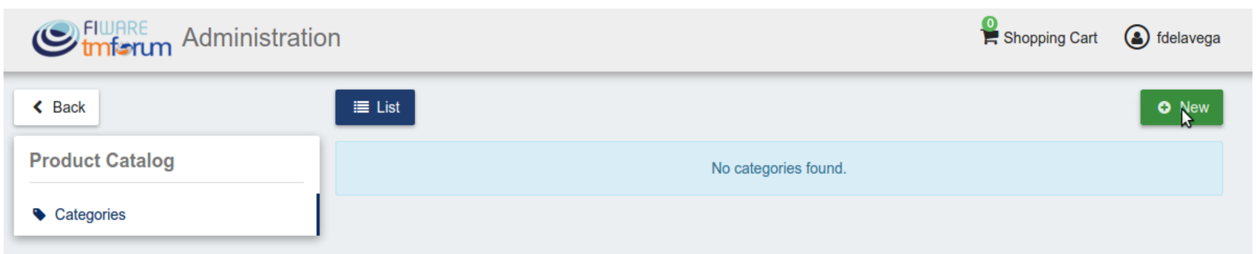
If the user of the Business API Ecosystem is an admin, he will be able to access the *Administration* section of the web portal. This section is located in the user menu.



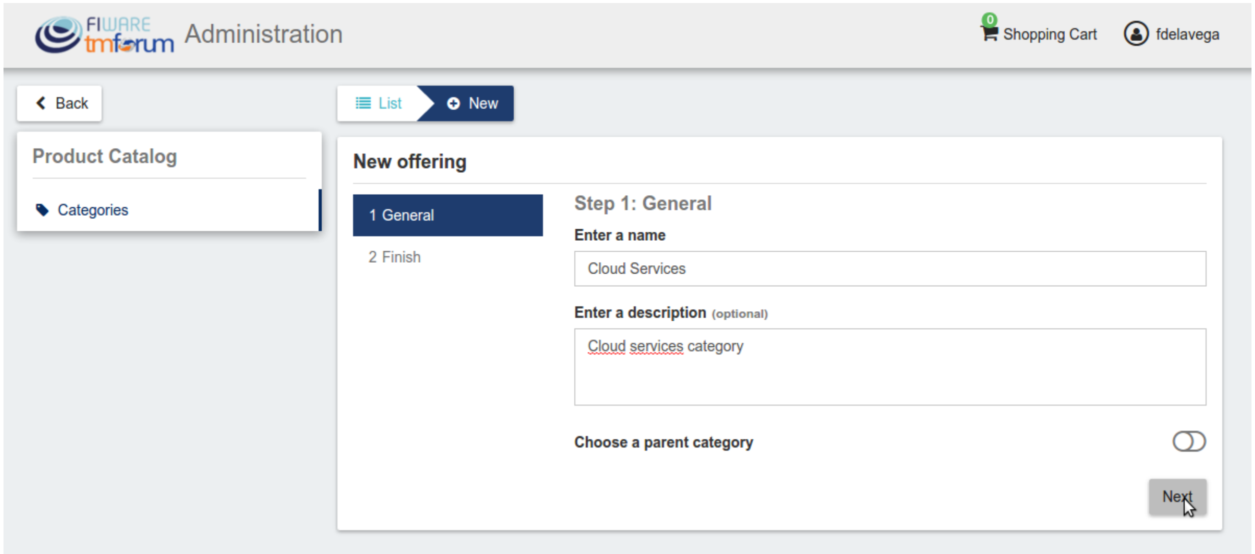
Manage Categories

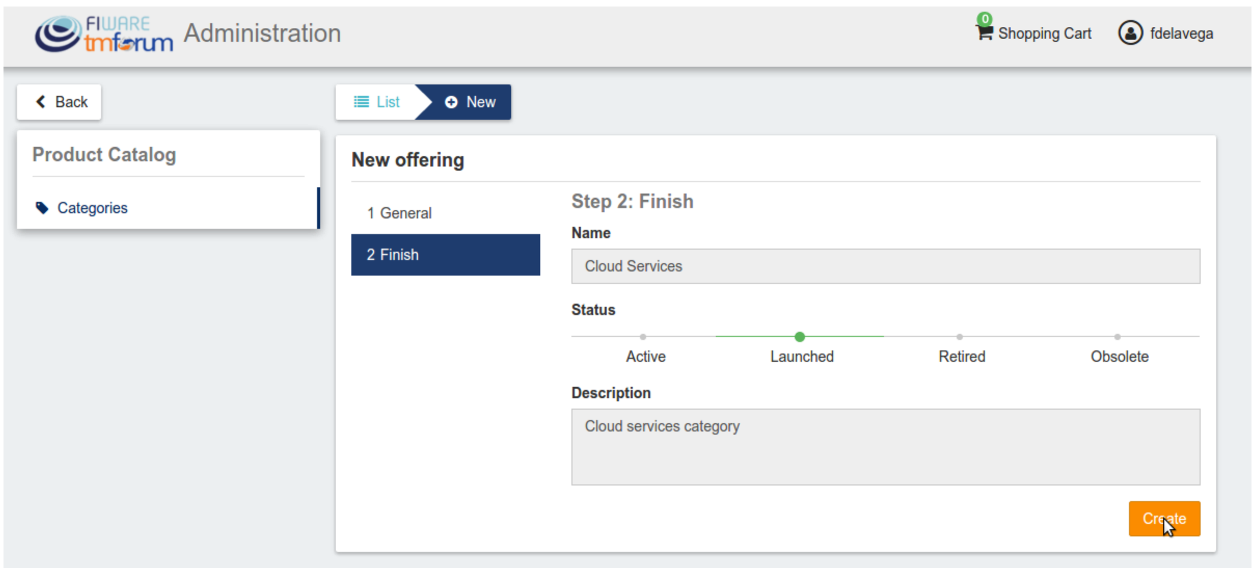
Admin users are authorized to create the system categories that can be used by *Sellers* to categorize their catalogs, products, and offerings.

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

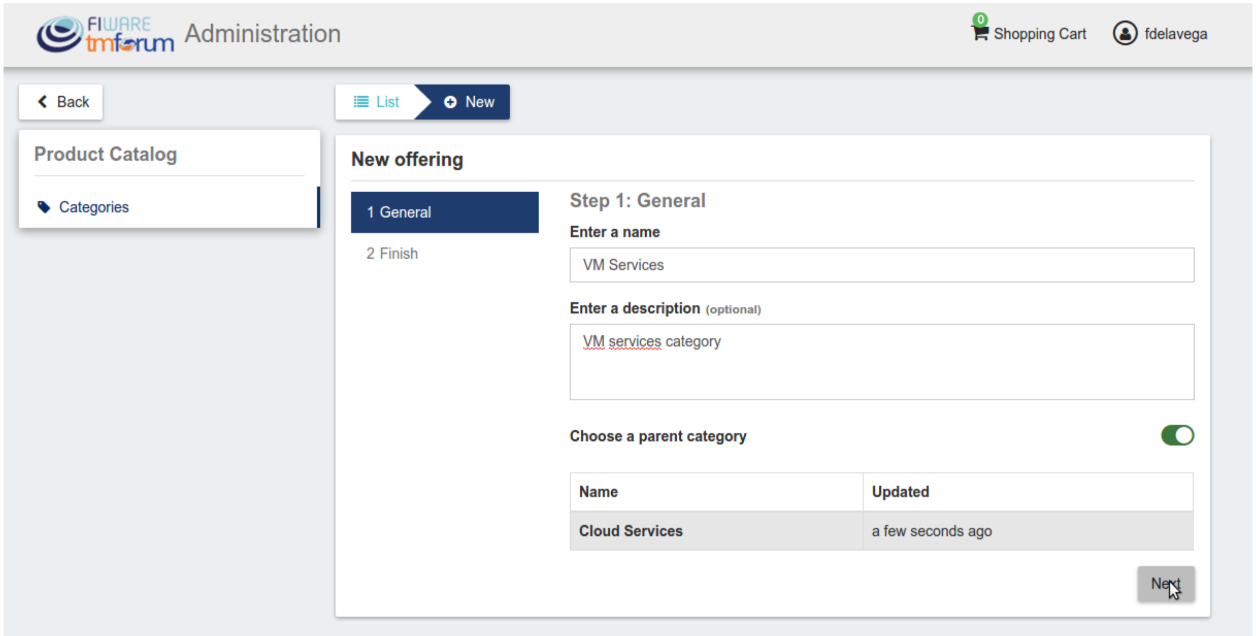


Then, provide a name and an optional description for the category. Once the information has been included, click on *Next*, and then on *Create*

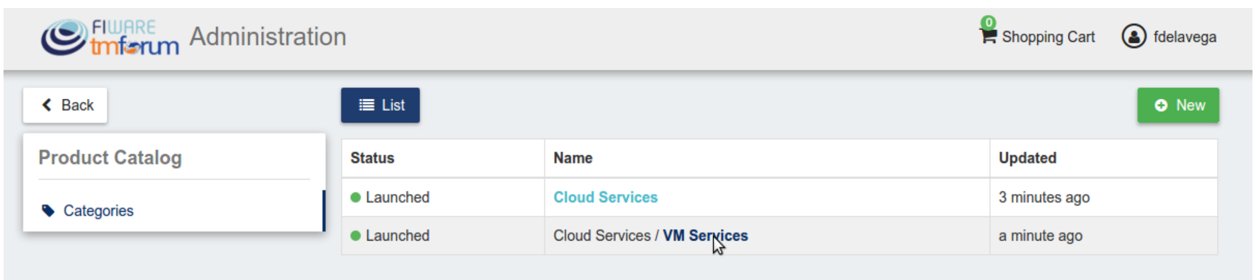




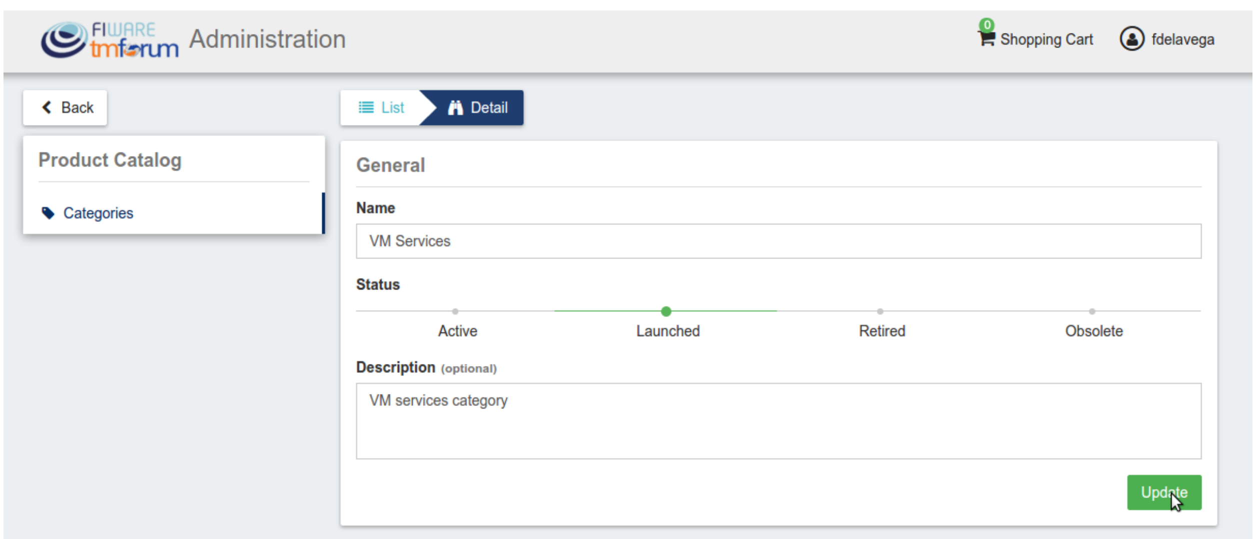
Categories in the Business API Ecosystem can be nested, so you can choose a parent category if you want, while creating.



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

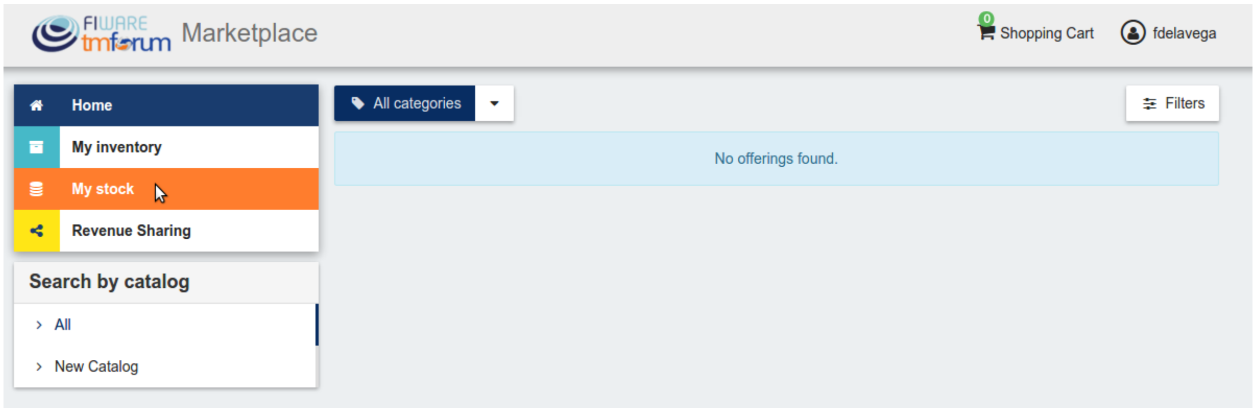


Then edit the corresponding fields and click on *Update*.



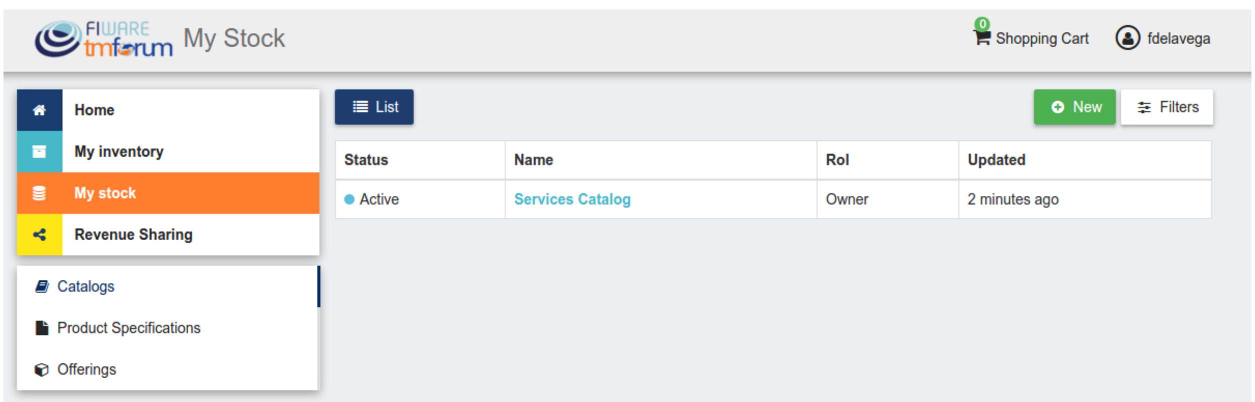
Seller

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

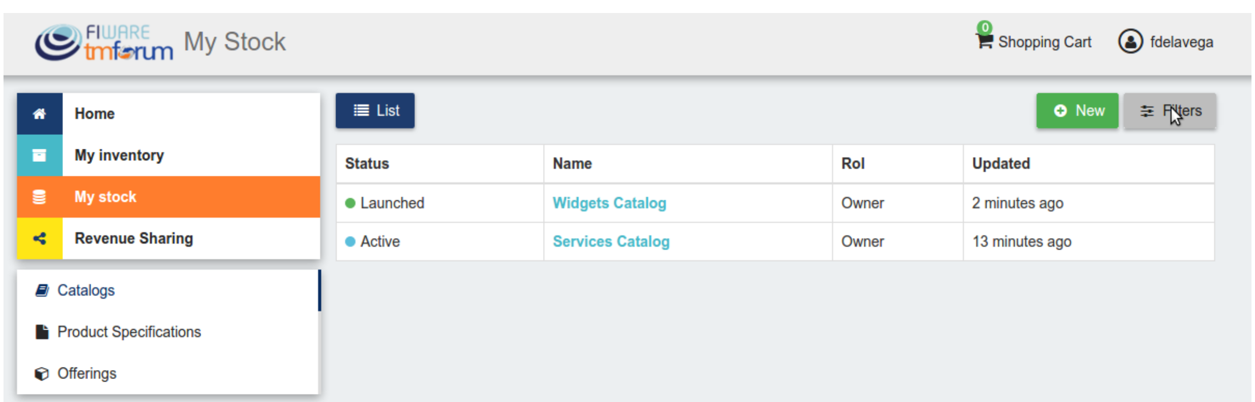


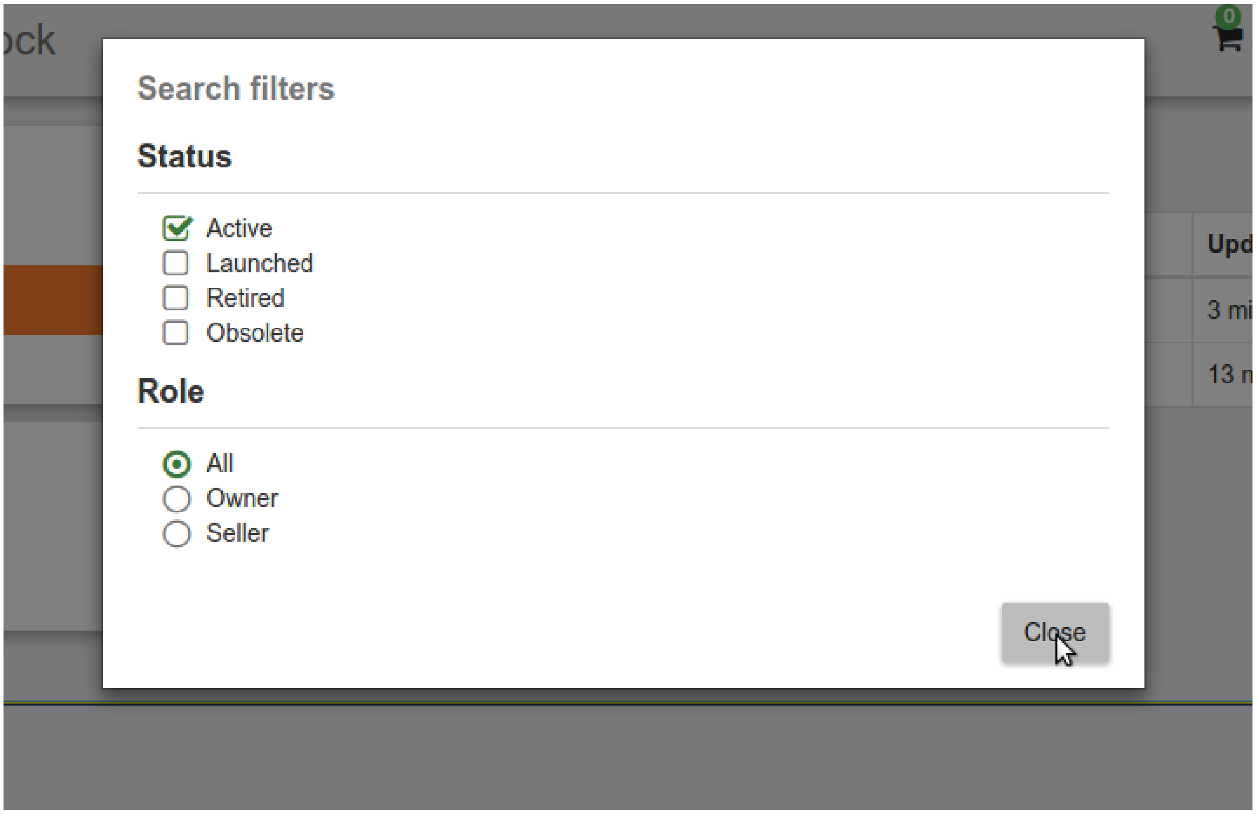
Manage Catalogs

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

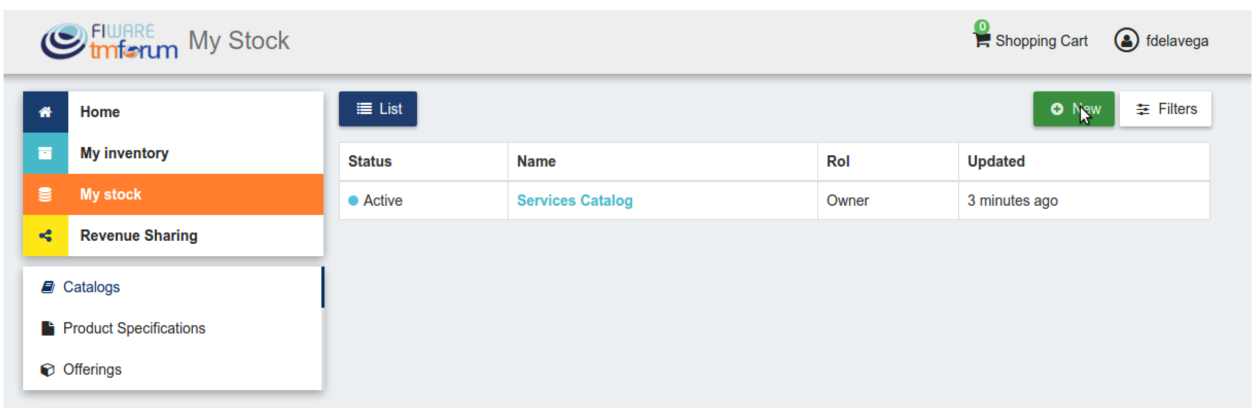


Additionally, it is possible to filter the shown catalogs by status and the role you are playing by clicking on *Filters*, choosing the required ones, and clicking on *Close*

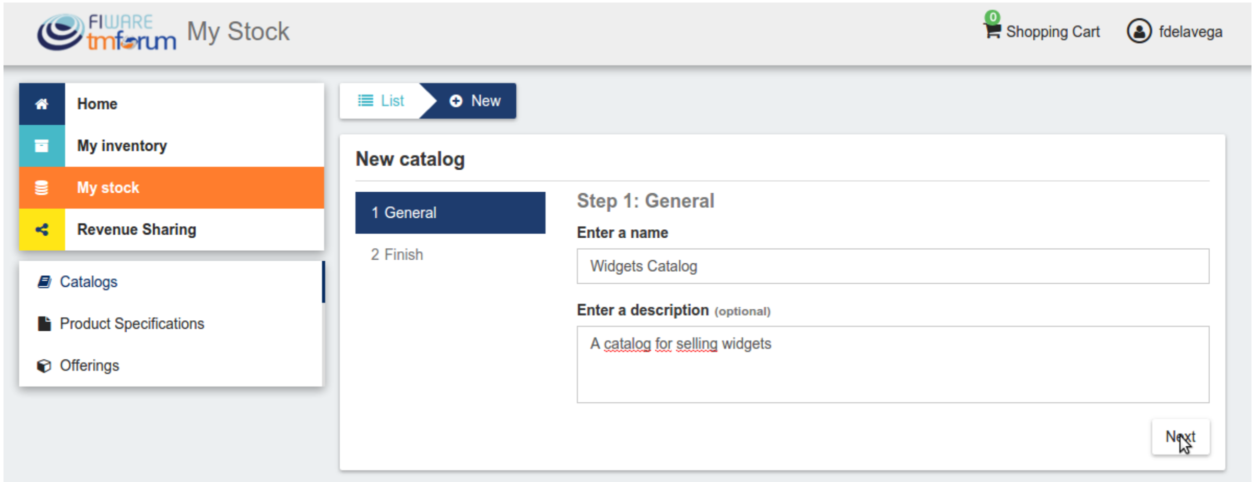


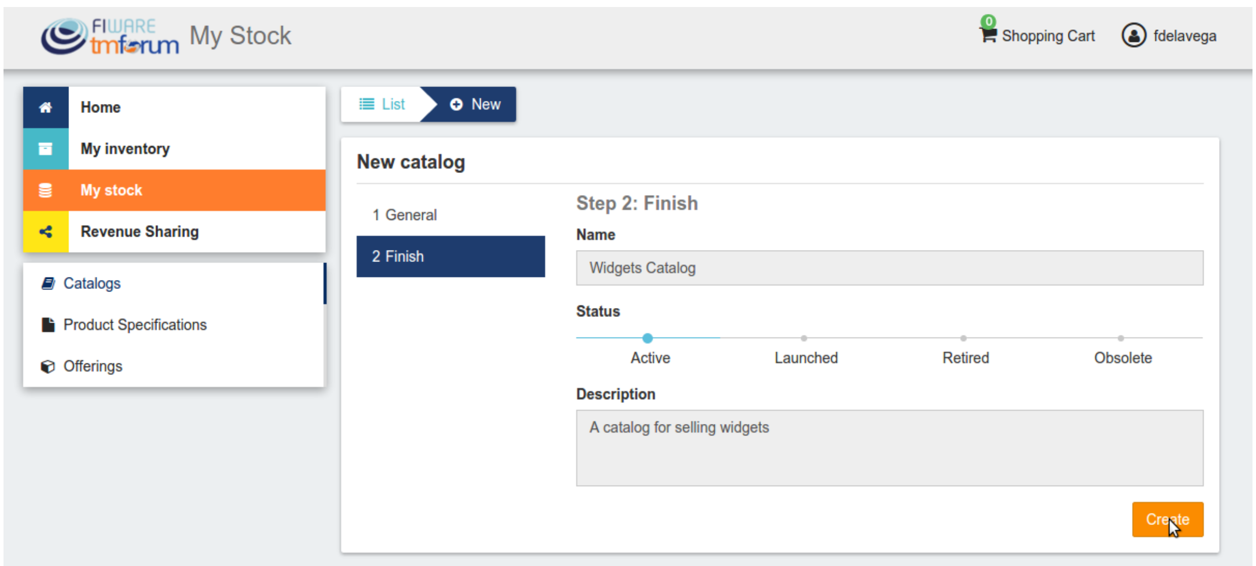


To create a new catalog click on the *New* button.

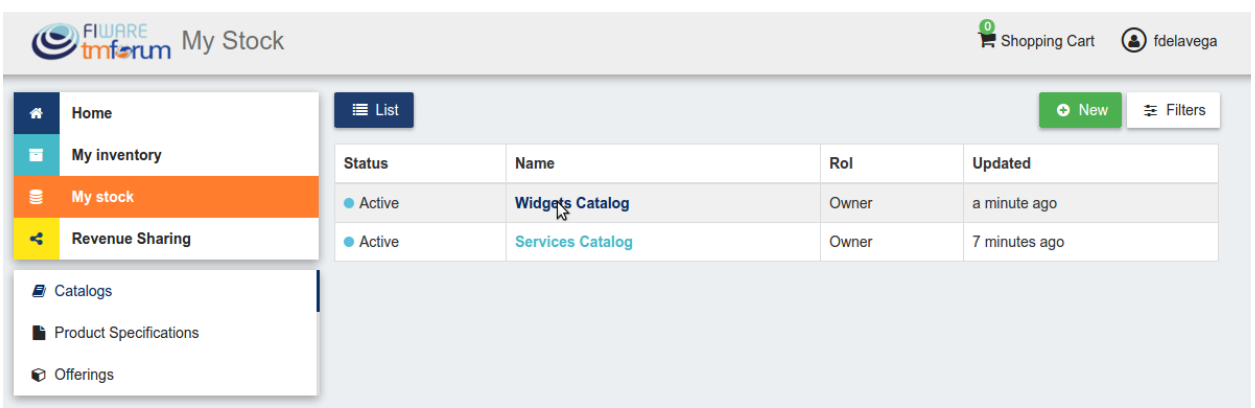


Then, provide a name and an optional description for the catalog. Once you have filled the fields, click on *Next*, and then on *Create*

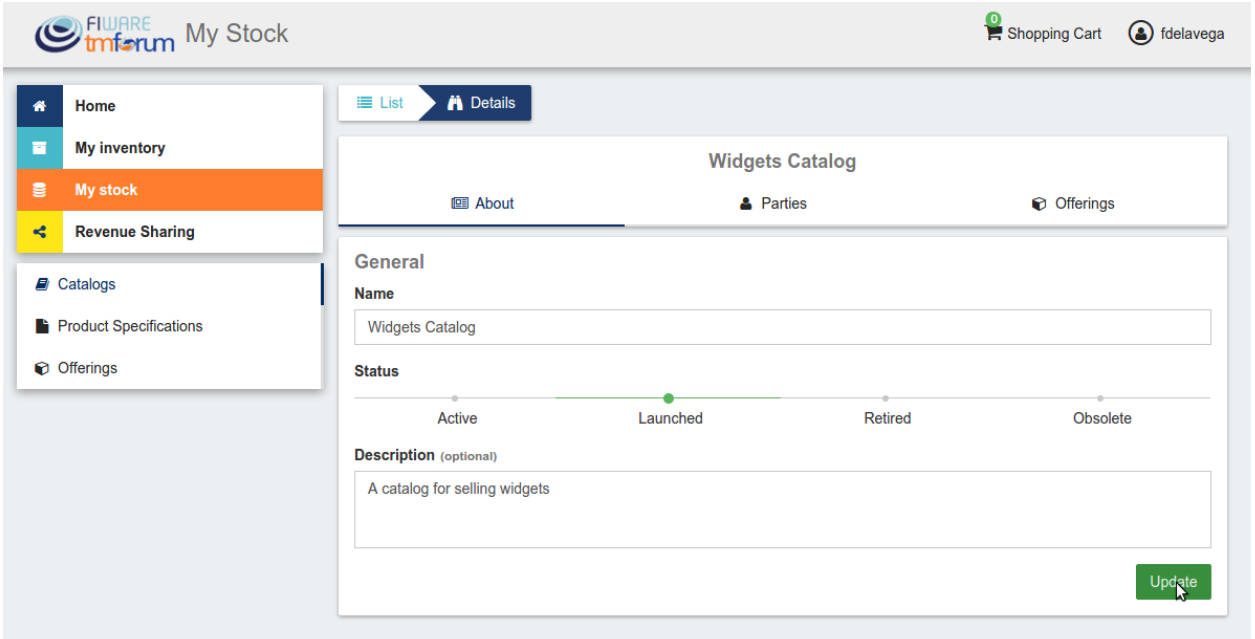




Sellers, can also update their catalogs. To do that, click on the name of the catalog to open the update view.

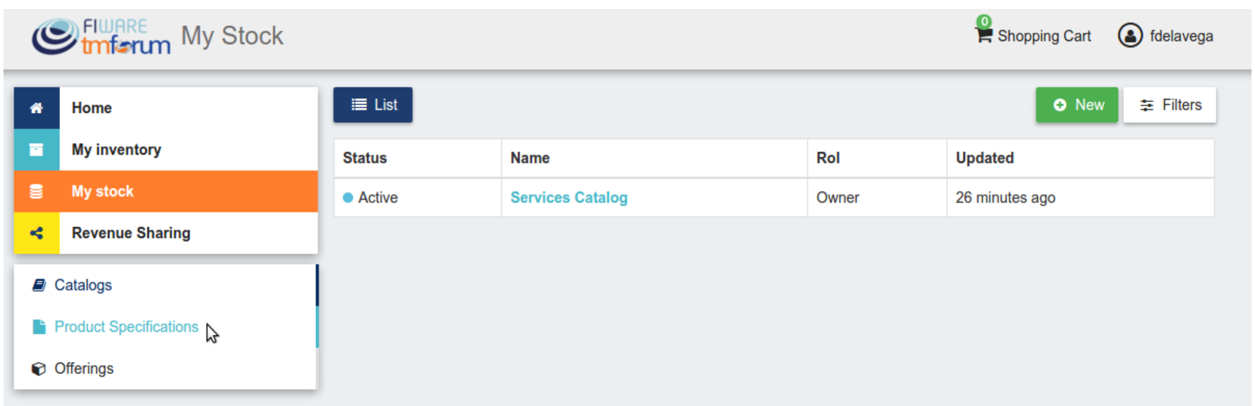


Then update the fields you want to modify and click on *Update*. In this view, it is possible to change the *Status* of the catalog. To start monetizing the catalog, and make it appear in the *Home* you have to change its status to *Launched*

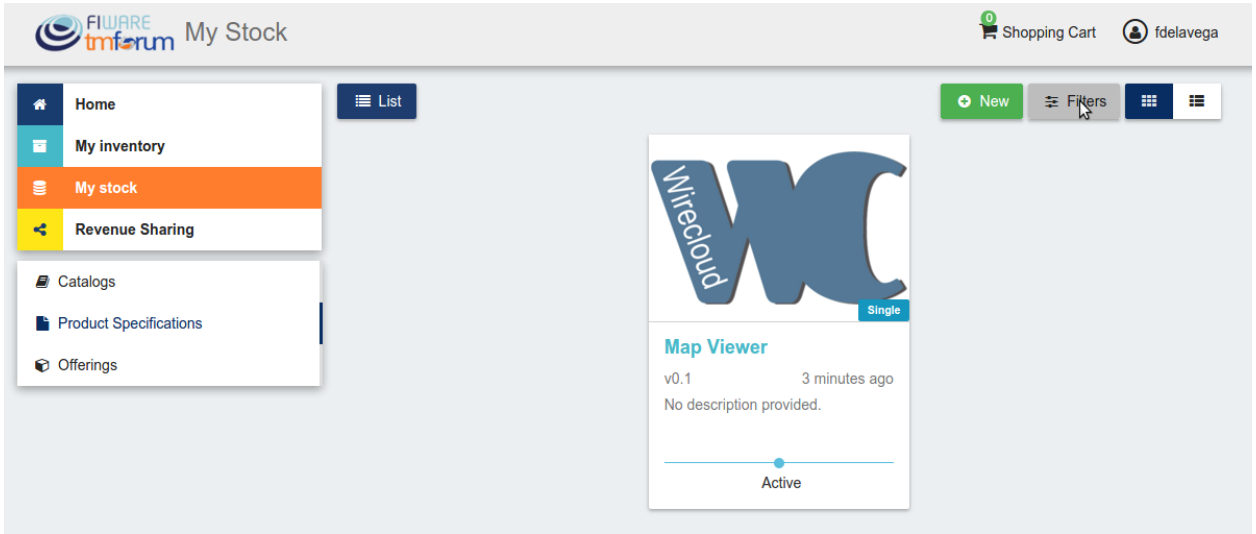


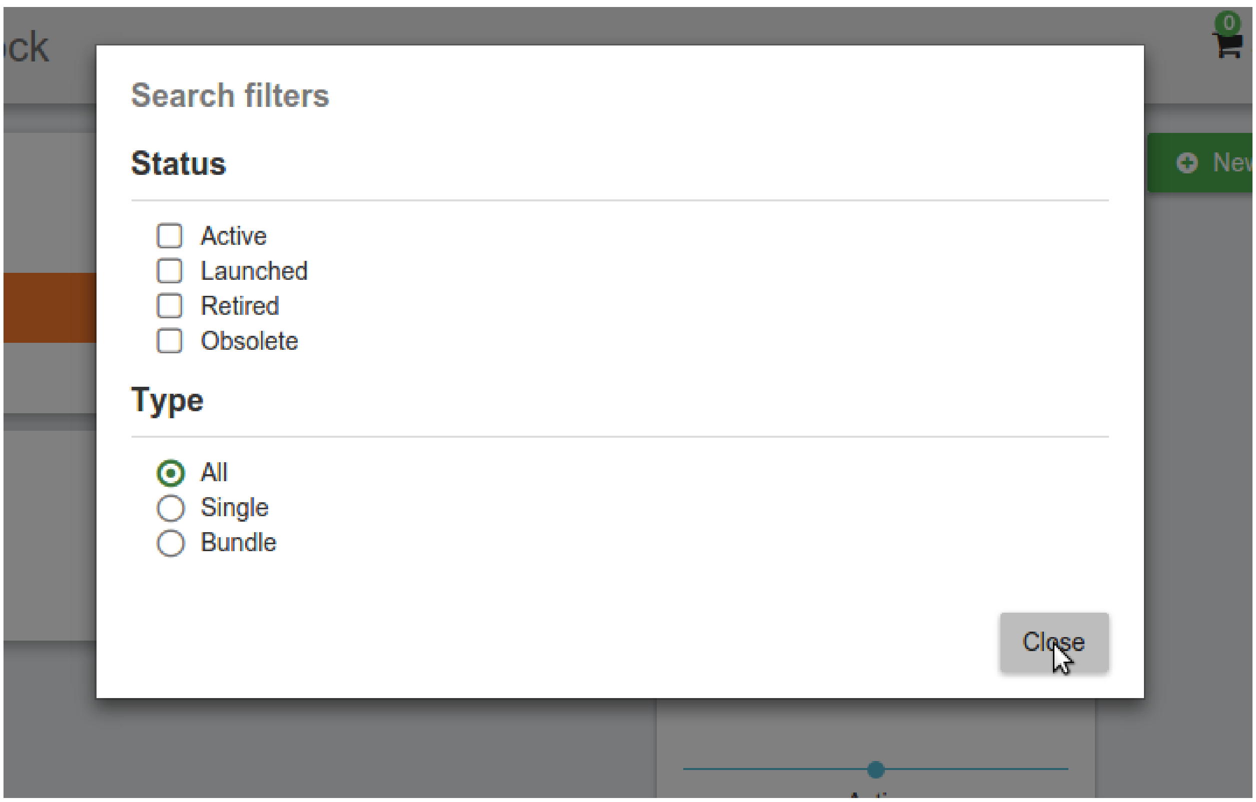
Manage Product Specifications

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

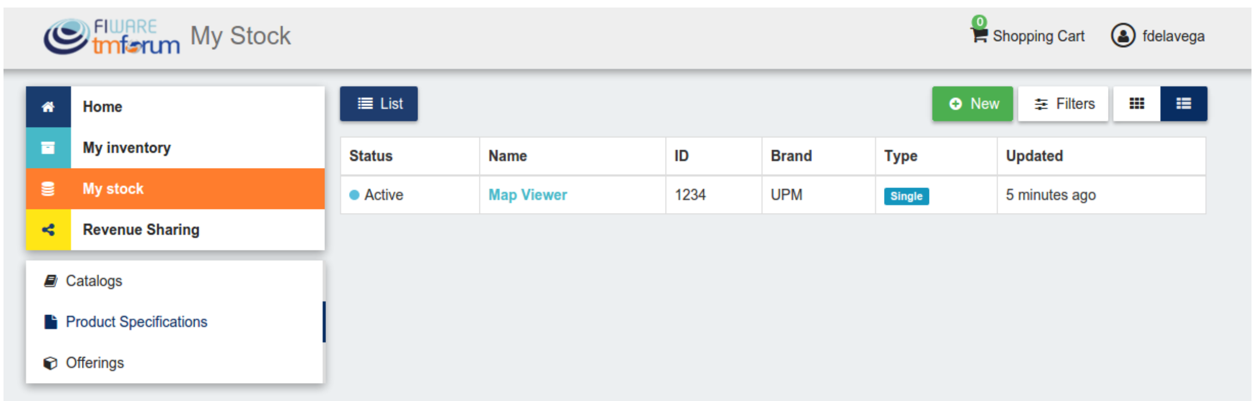


The different product specifications can be filtered by status or by if they are bundles or not. To filter products, click on *Filters*, choose the appropriate ones, and click on *Close*

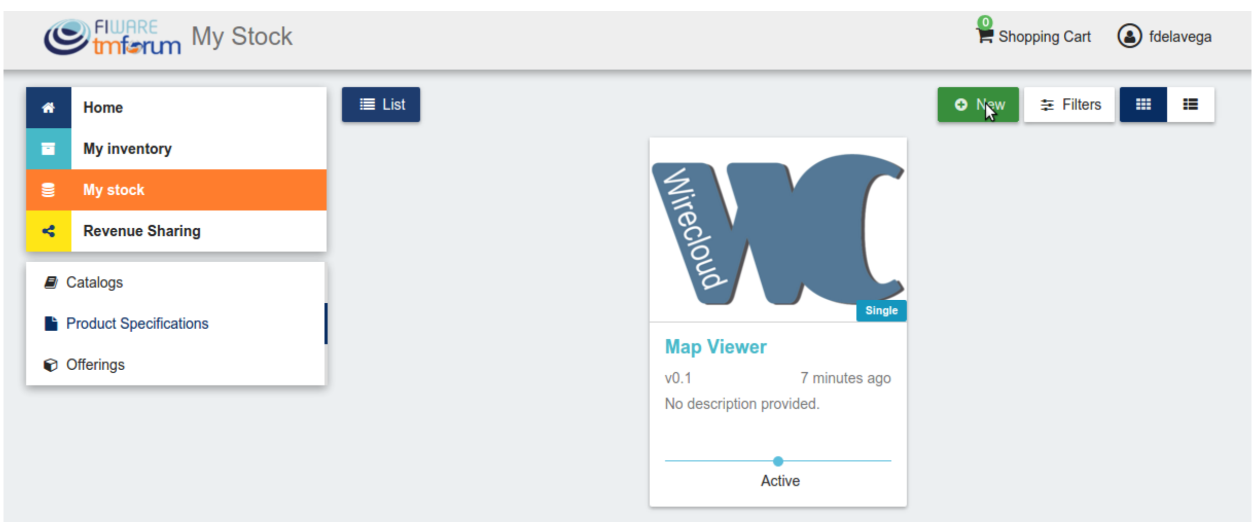




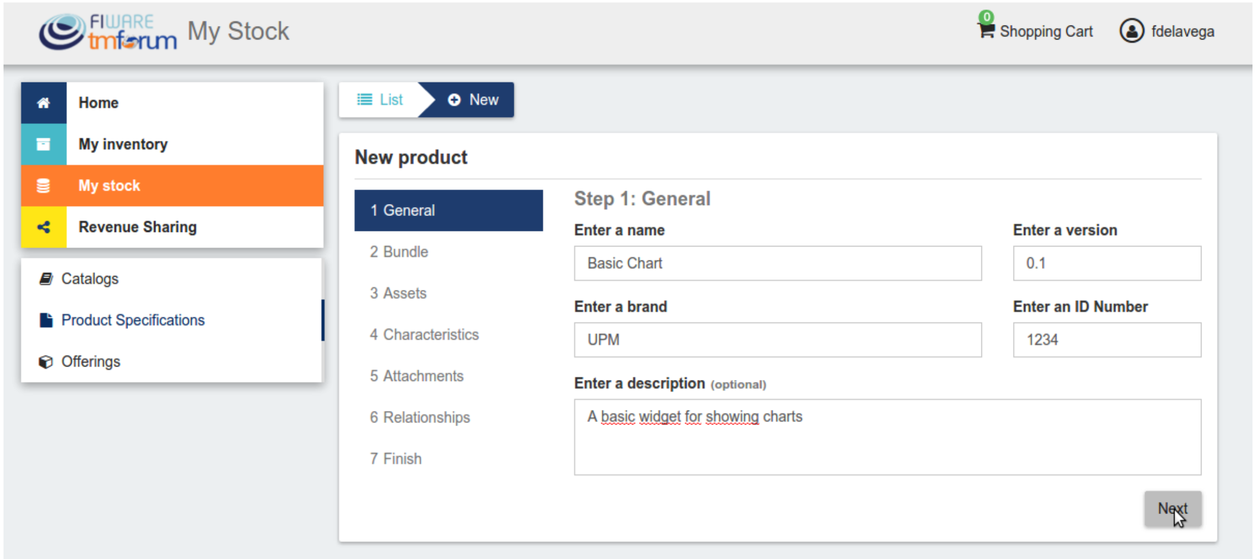
Additionally, it is possible to switch between the grid view and the tabular view using the provided buttons.



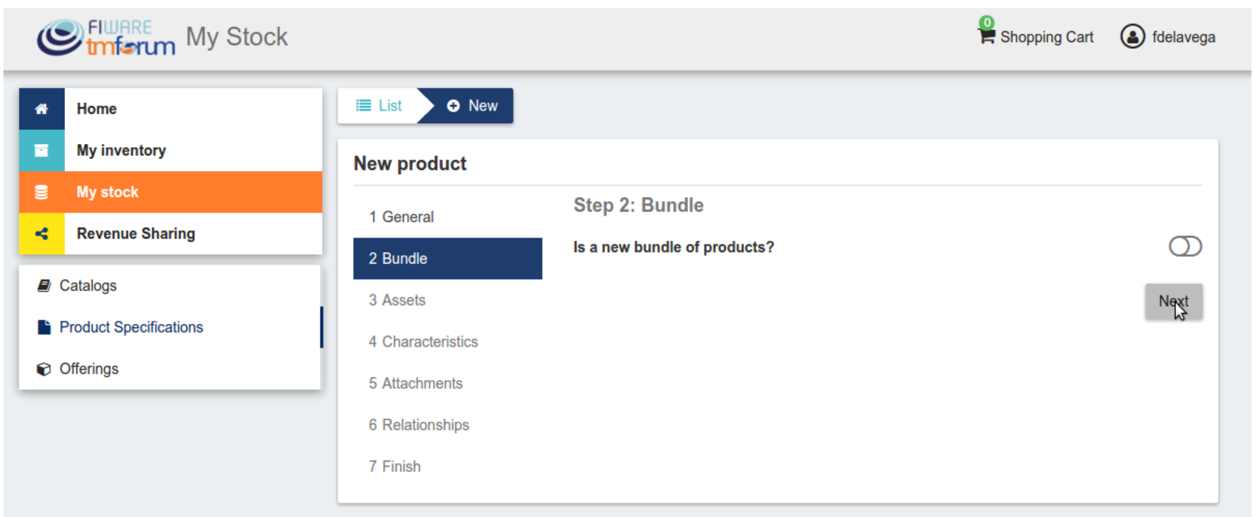
To create a new product specification click on *New*



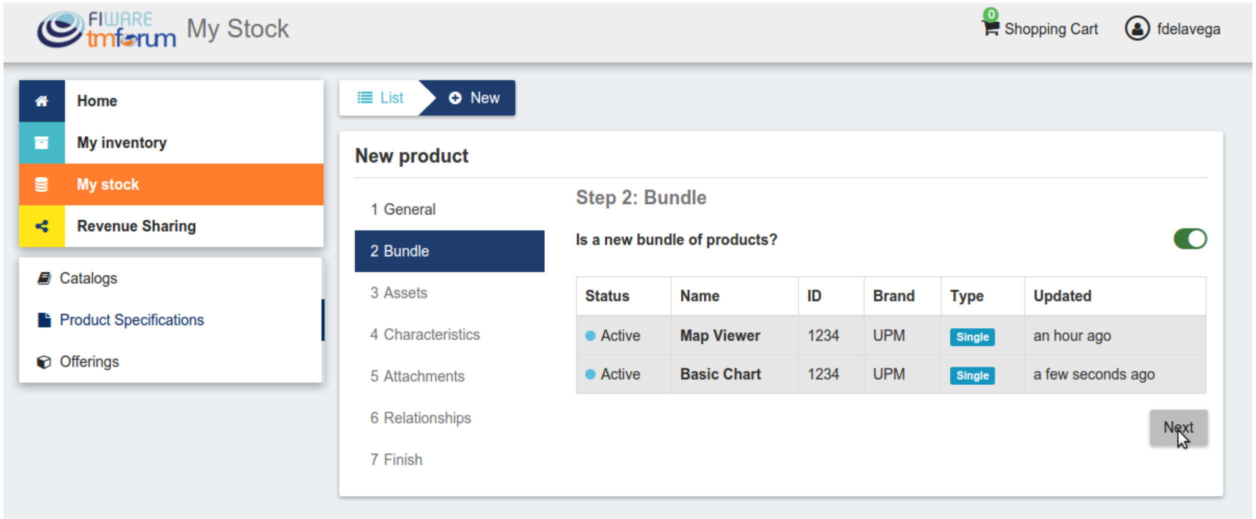
In the displayed view, provide the general information of the product spec. including its name, version, and an optional description. In addition, you have to include the product brand (Your brand), and an ID number which identifies the product in your environment. Then, click on *Next*.



In the next step, you can choose whether your product specification is a bundle or not. Product bundles are logical containers that allows 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.

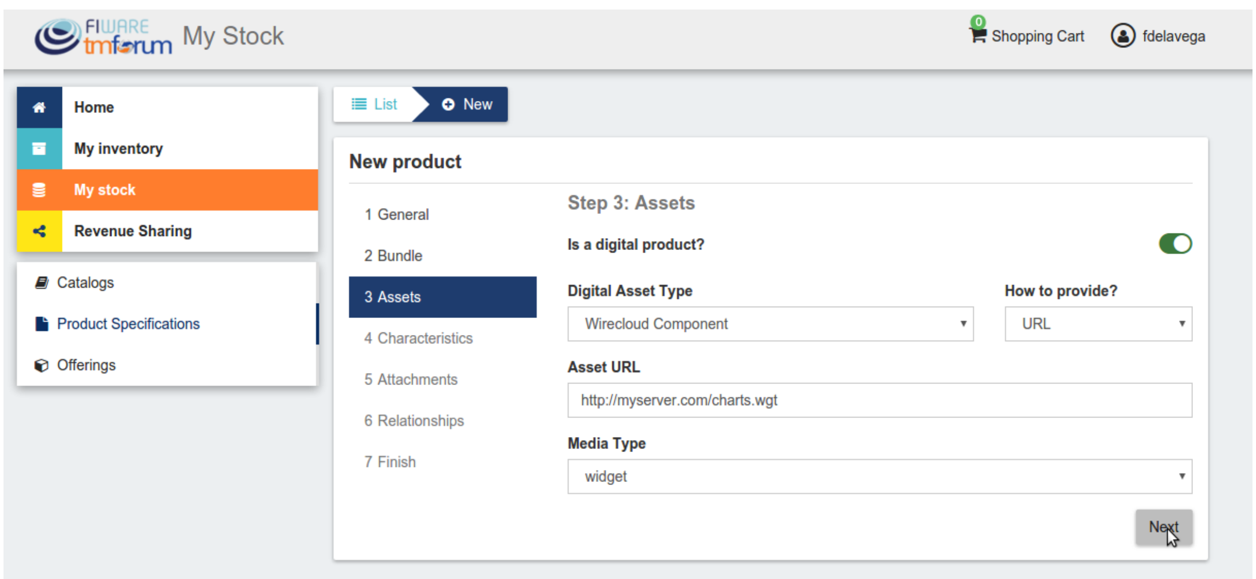


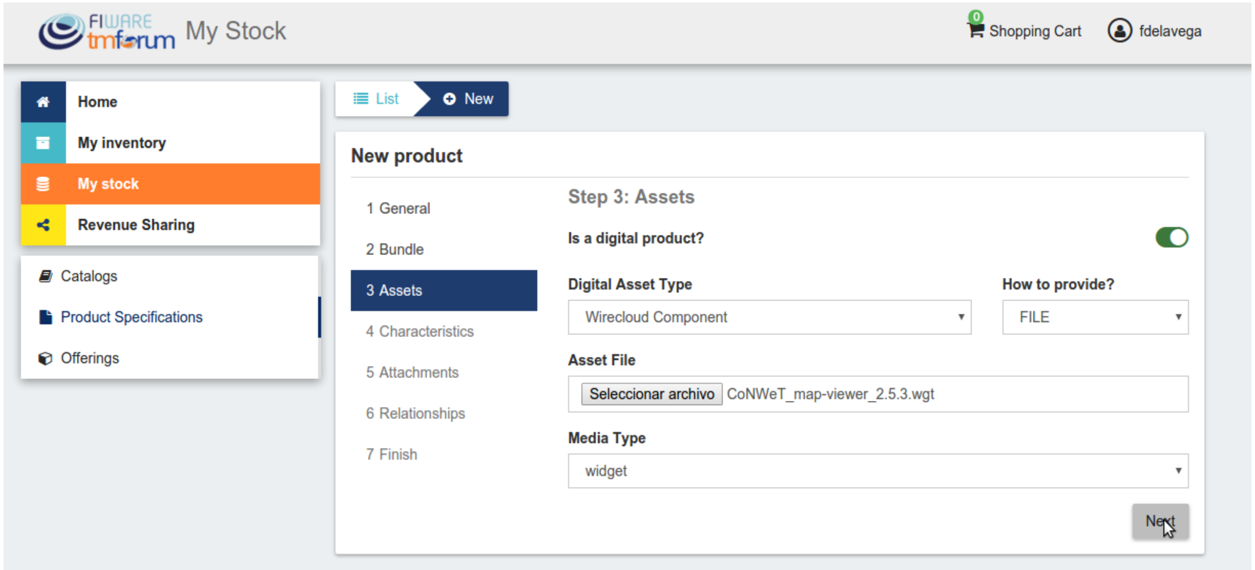
In the next step you can choose if your product is a digital product. If this is the case, you will be required to provide the asset.

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

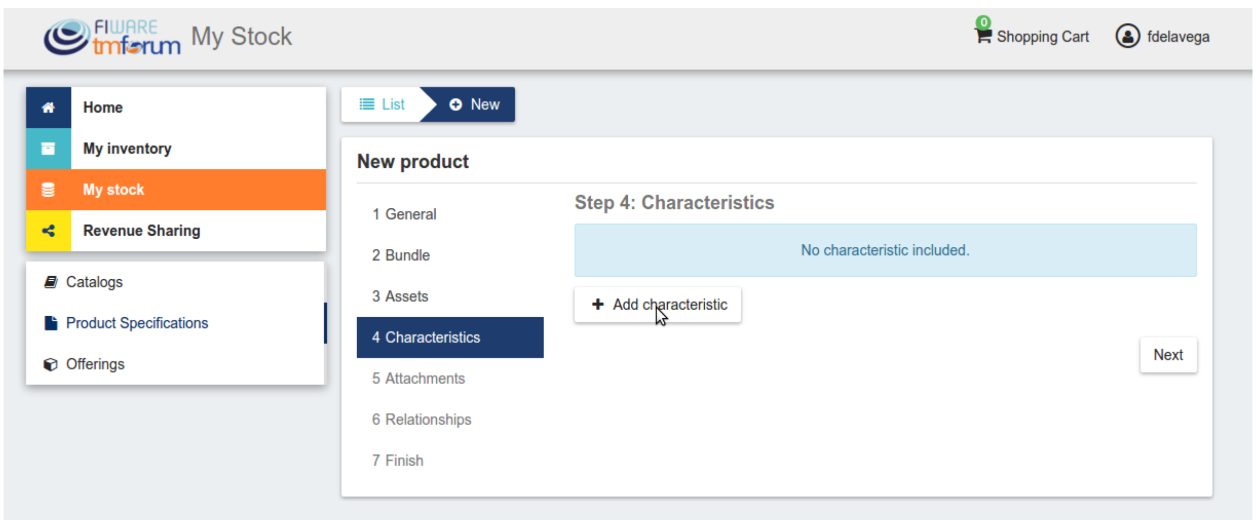
---

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.

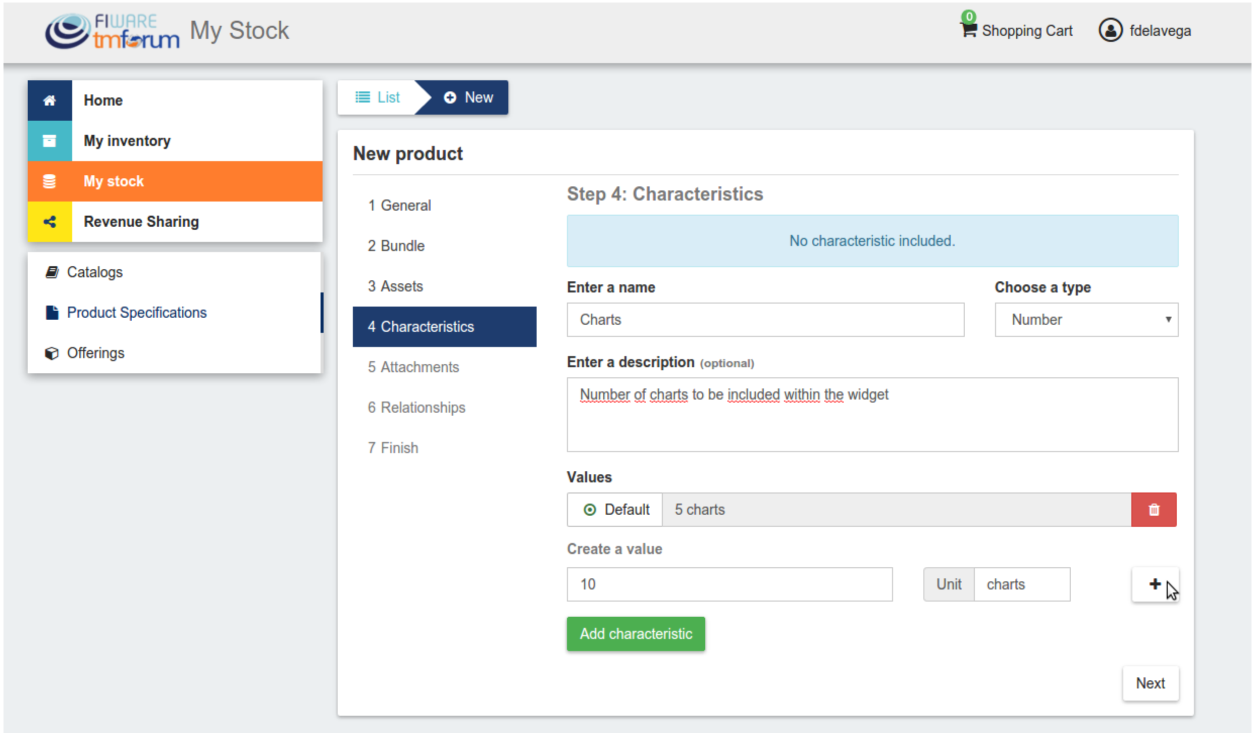




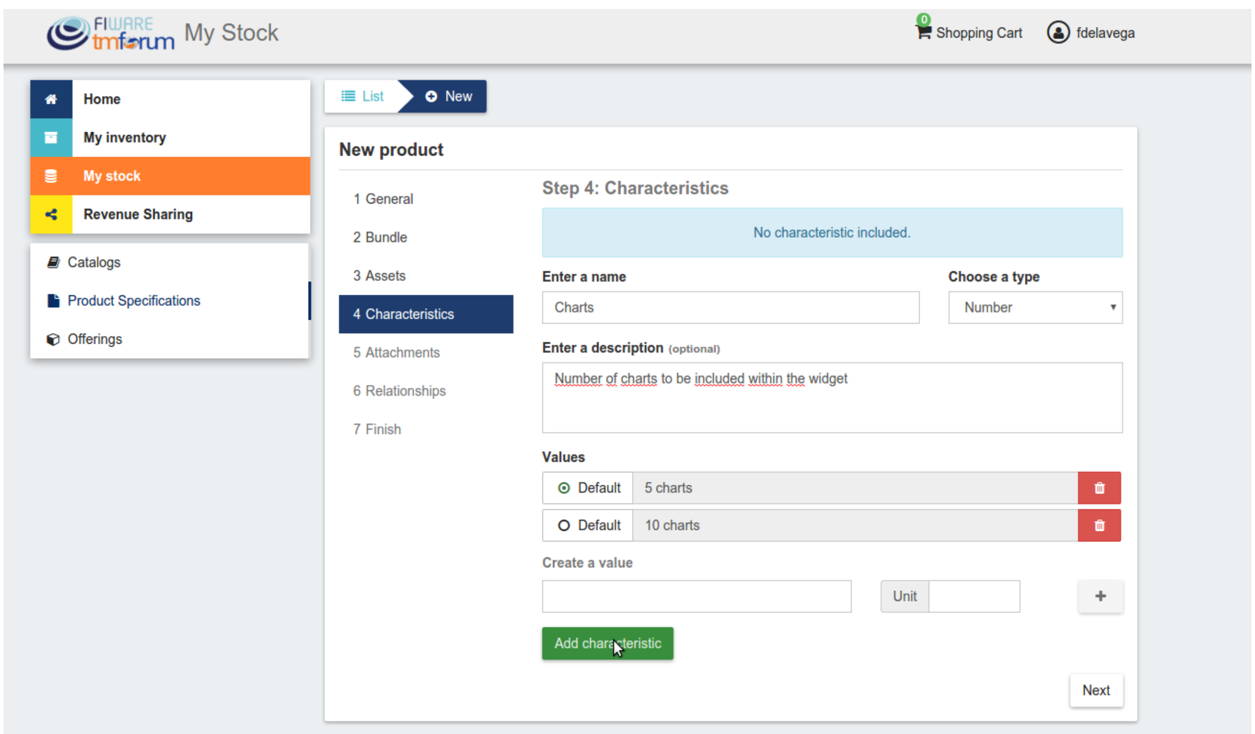
The next step in the creation of a product is including its characteristics. For including a new characteristic click on *Add Characteristic*



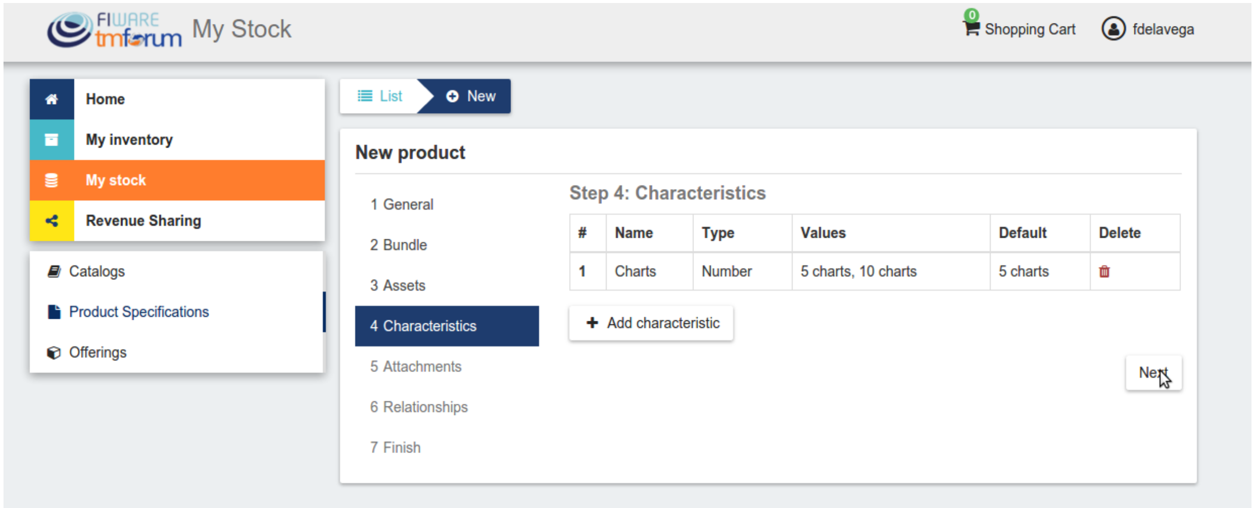
In the form, include the name, the type (string or number) and an optional description. Then create the values of the characteristic by filling the *Create a value* input and clicking on +.



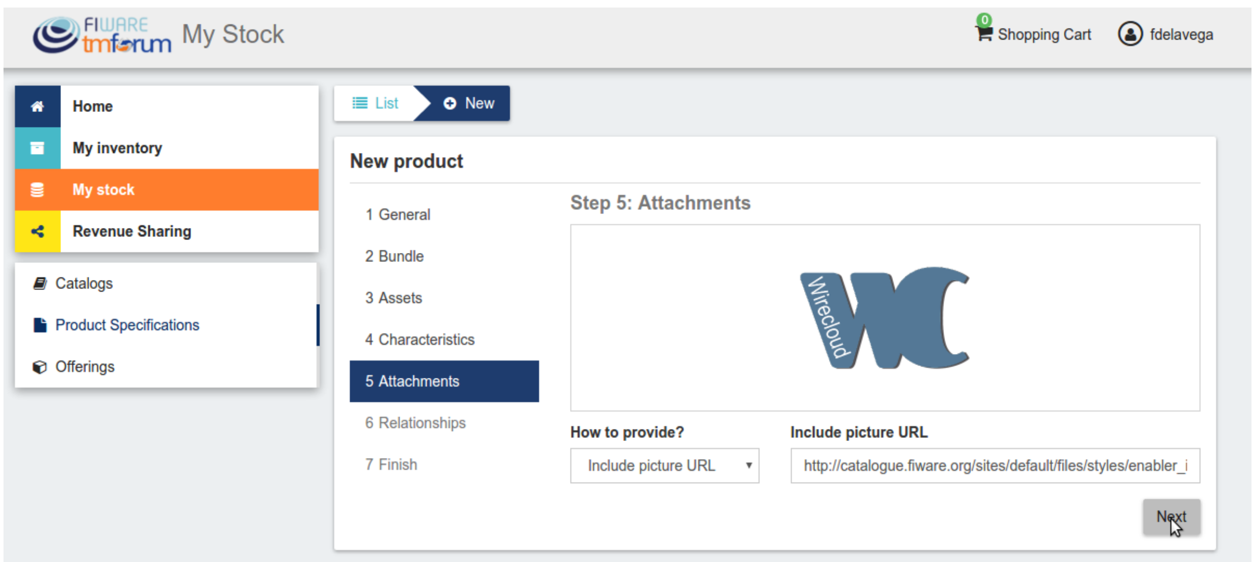
Once you have included all the characteristic info, save it clicking on *Add Characteristic*

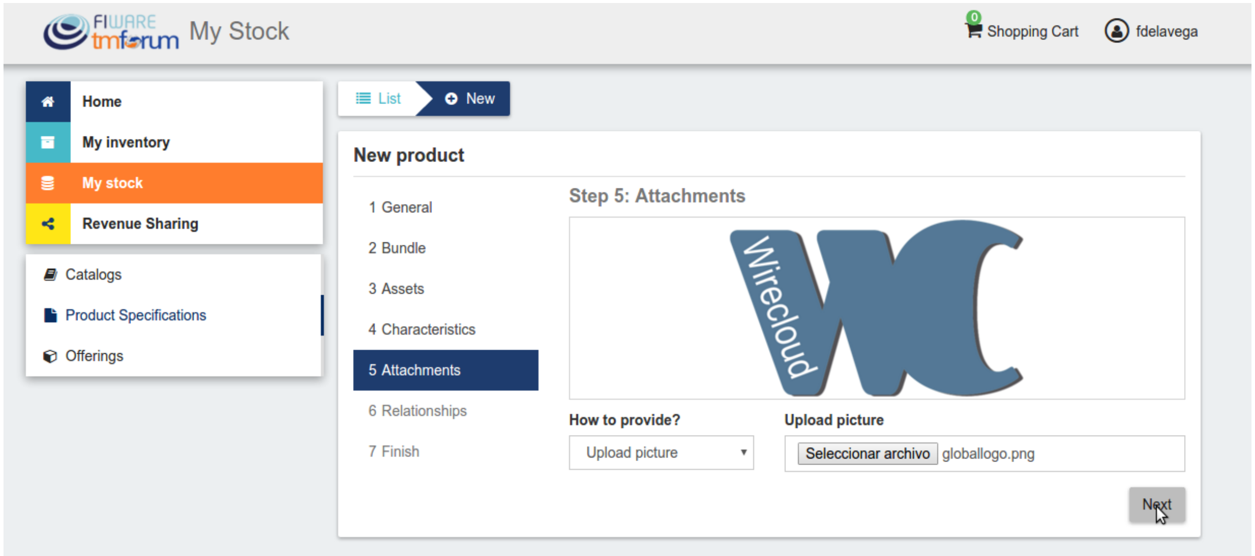


Once you have included all the required characteristics click on *Next*

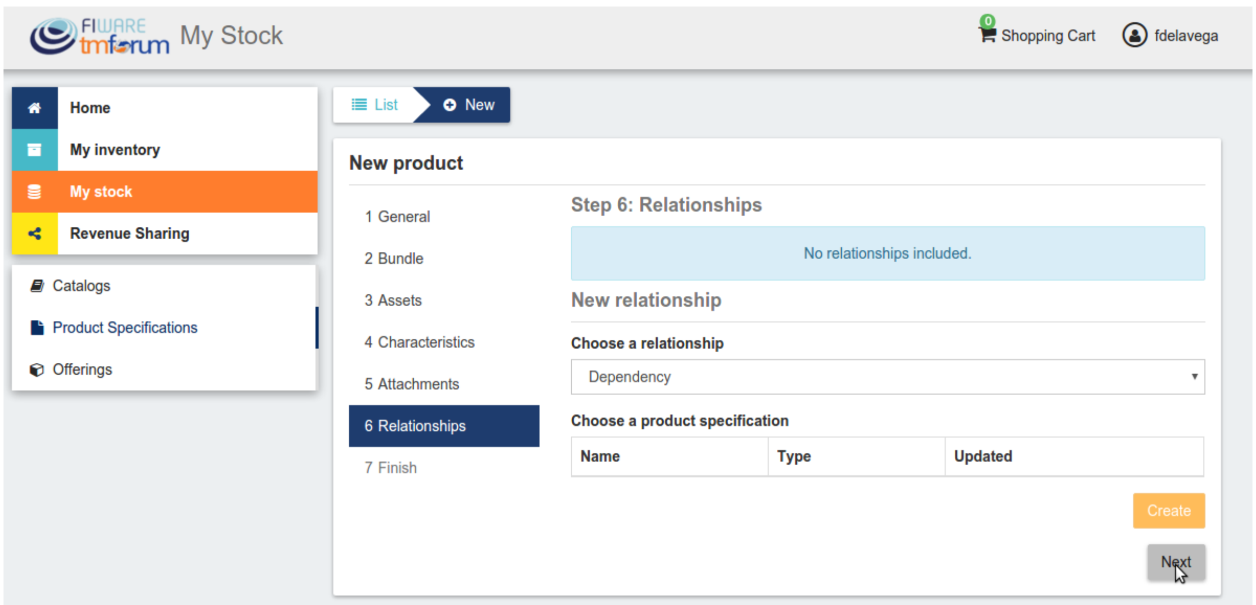


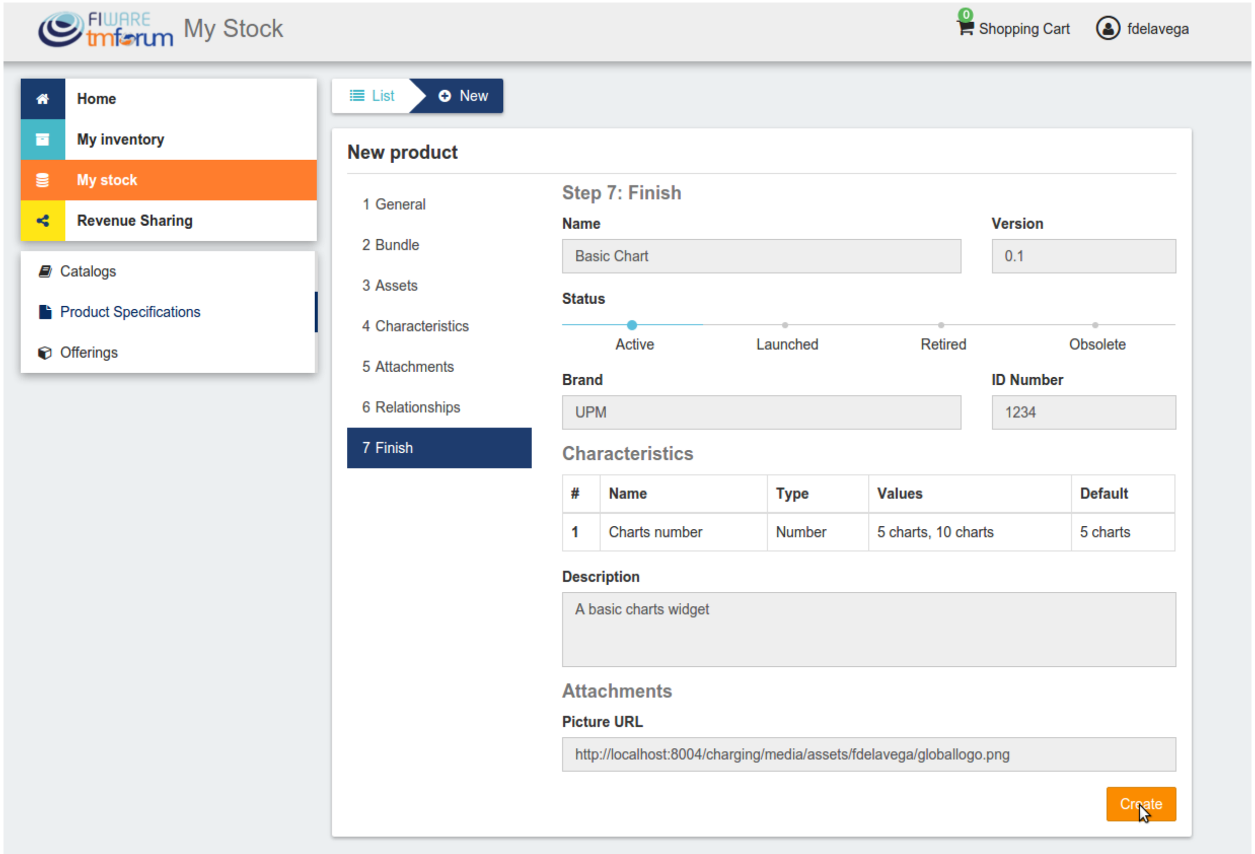
In the next step you can include a picture for your product spec. You have two options, providing an URL pointing to the picture or directly uploading it. Once provided click *Next*



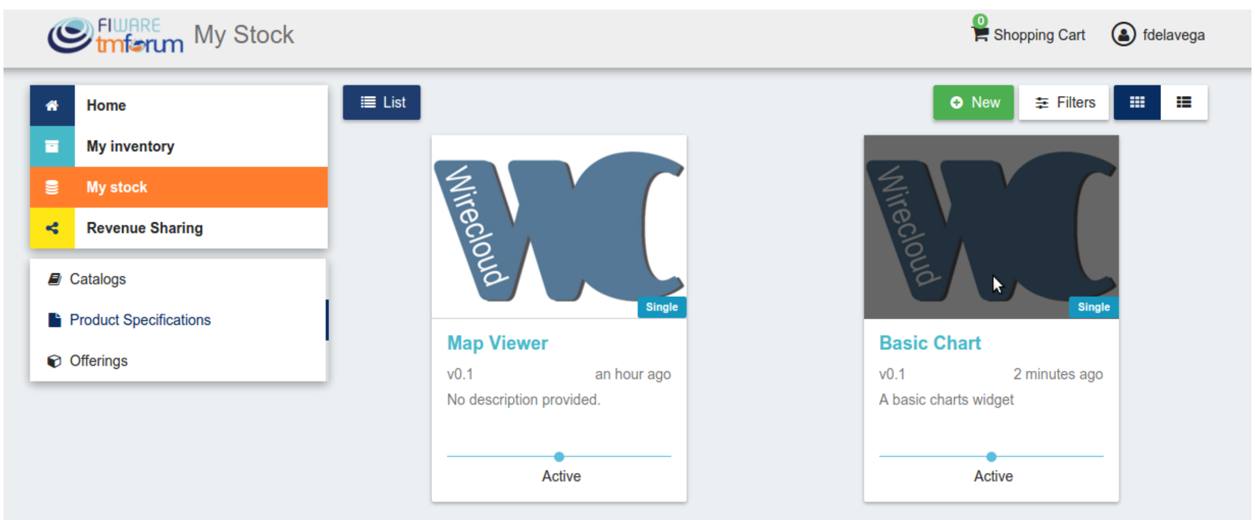


In the last step, you can specify relationships of the product you are creating with other of your product specs. Once done click on *Next* and then on *Create*

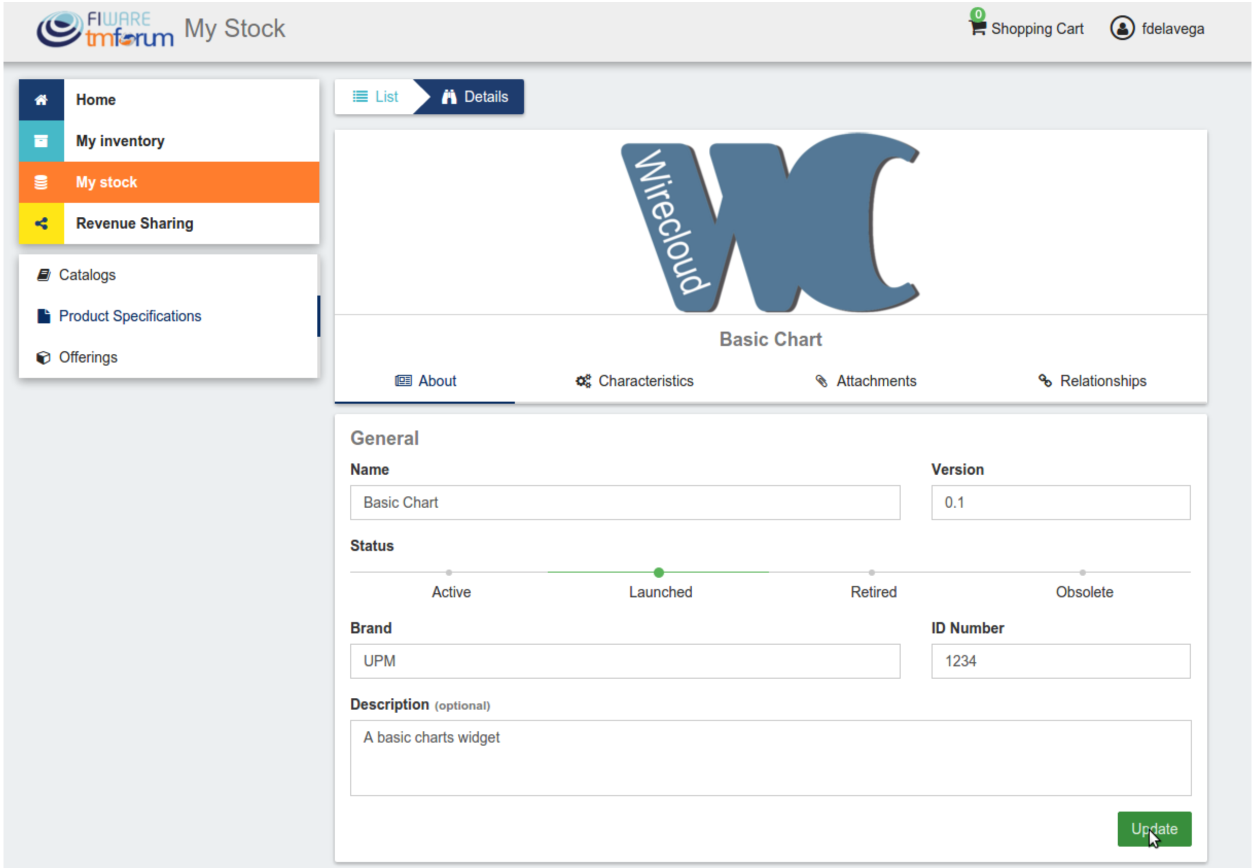




Sellers can update their products. To do that click on the product specification to be updated.

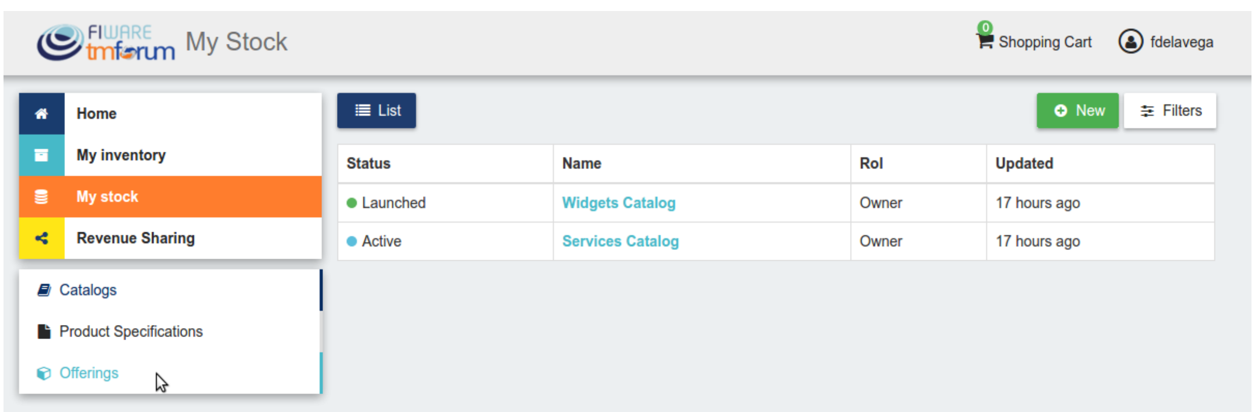


Update the required values and click on *Update*. Note that for start selling an offering that includes the product specification you will be required to change its status to *Launched*

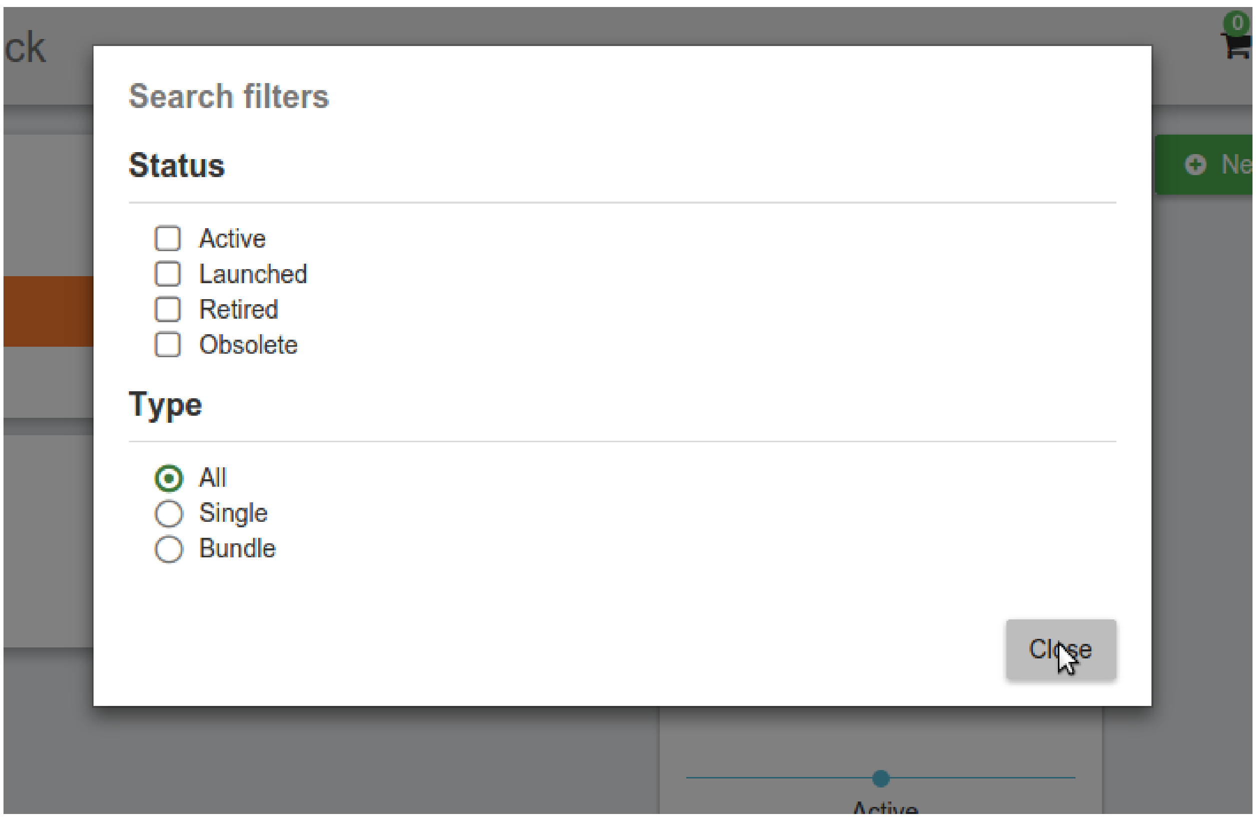


Manage Product Offerings

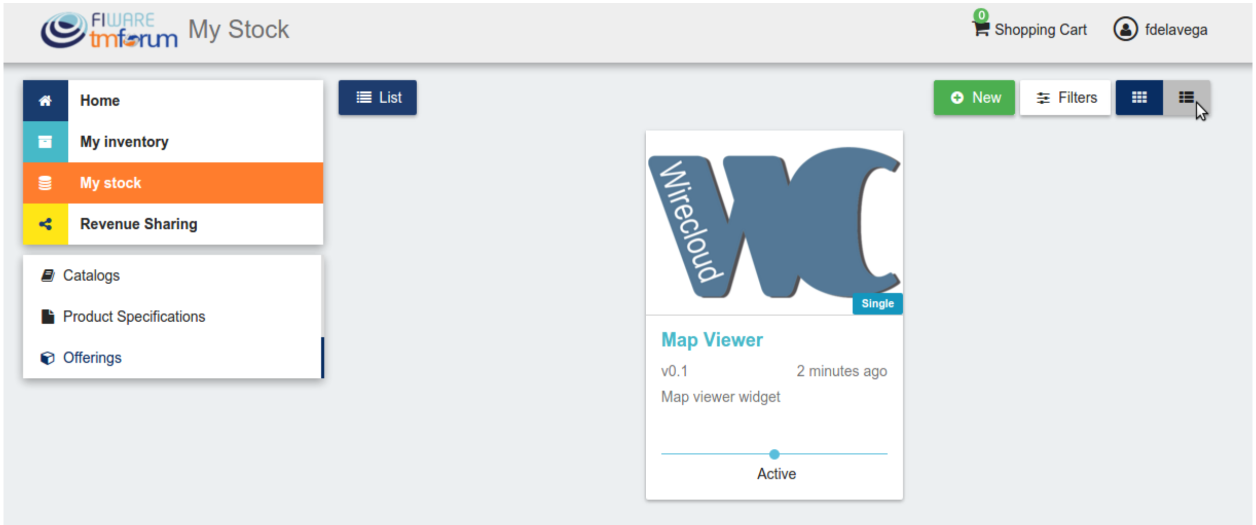
Product Offerings are the entities that contain the pricing models and revenue sharing info used to monetize a product specification. To list your product offerings, go to *My Stock* section and click on *Offerings*

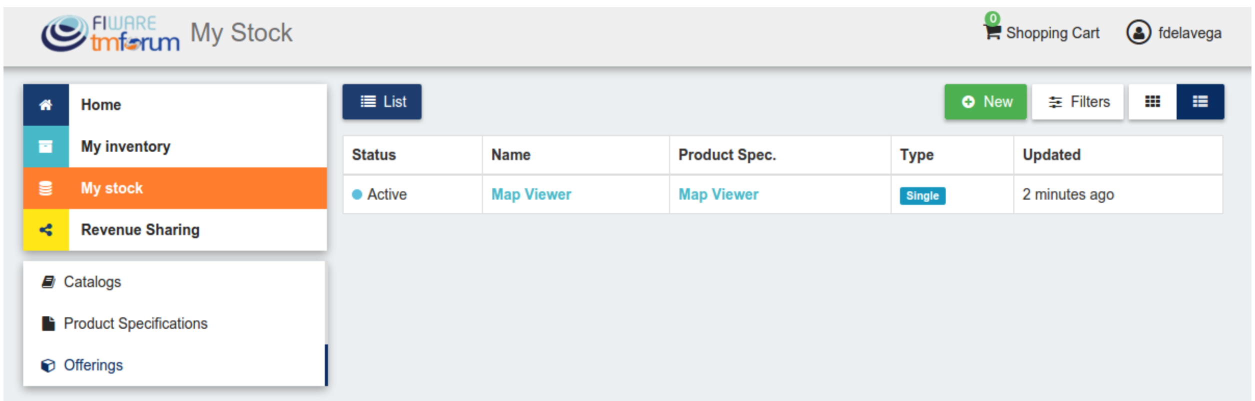


The existing product offerings can be filtered by status or by if they are a bundle or not. To filter offerings click on *Filters* choose the appropriate ones and click on *Close*

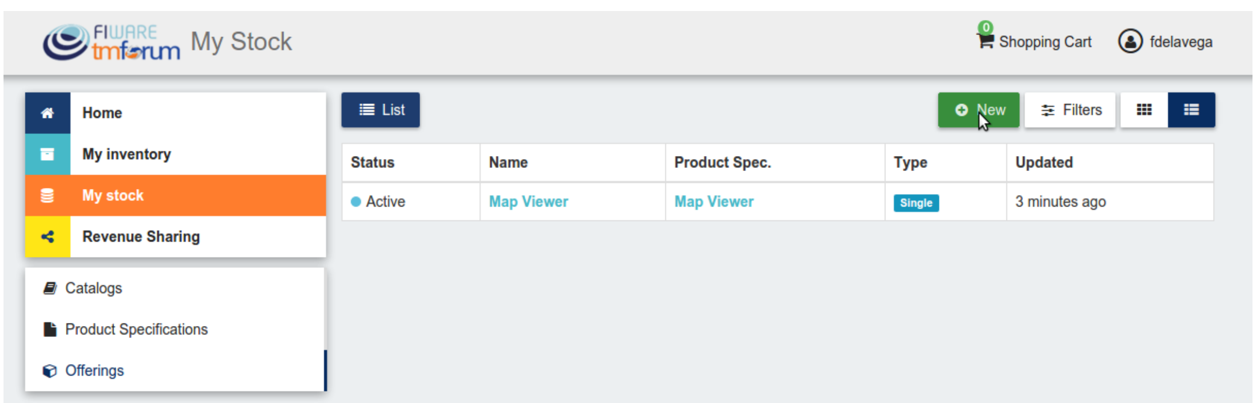


Additionally, it is possible to switch between the grid view and the tabular view by clicking on the specific button.

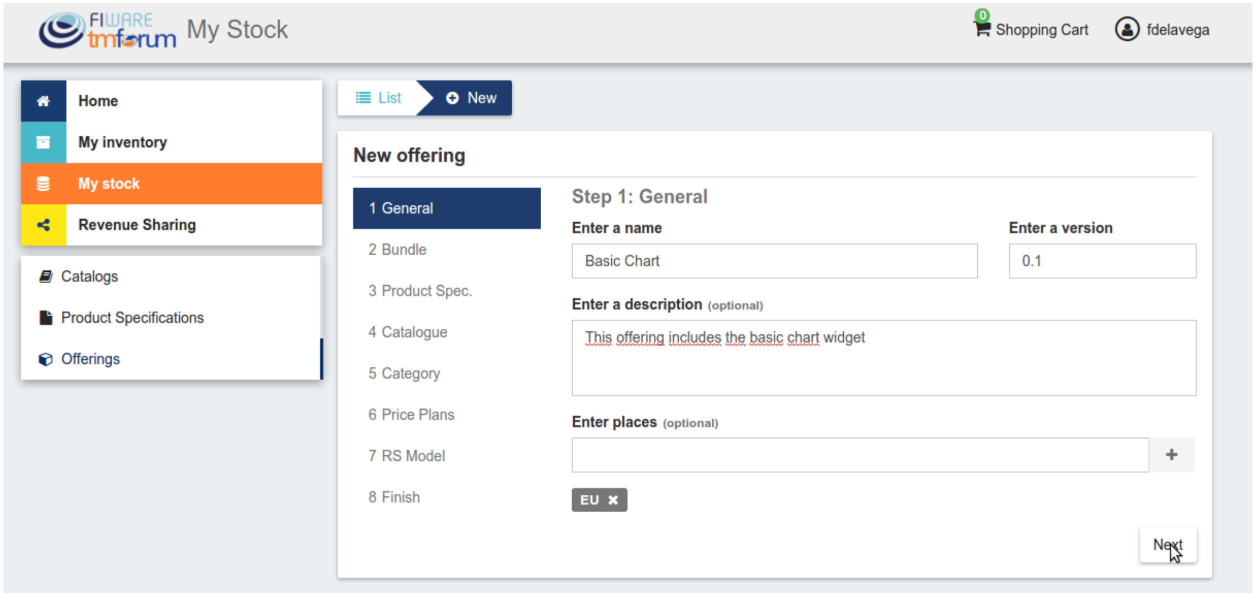




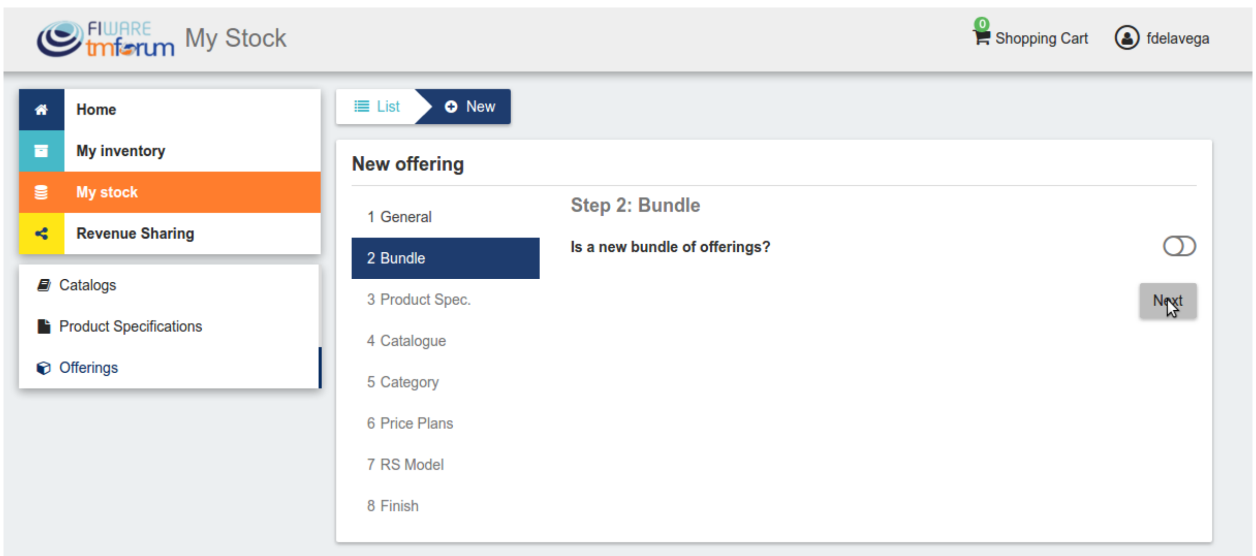
To create a new offering click on *New*



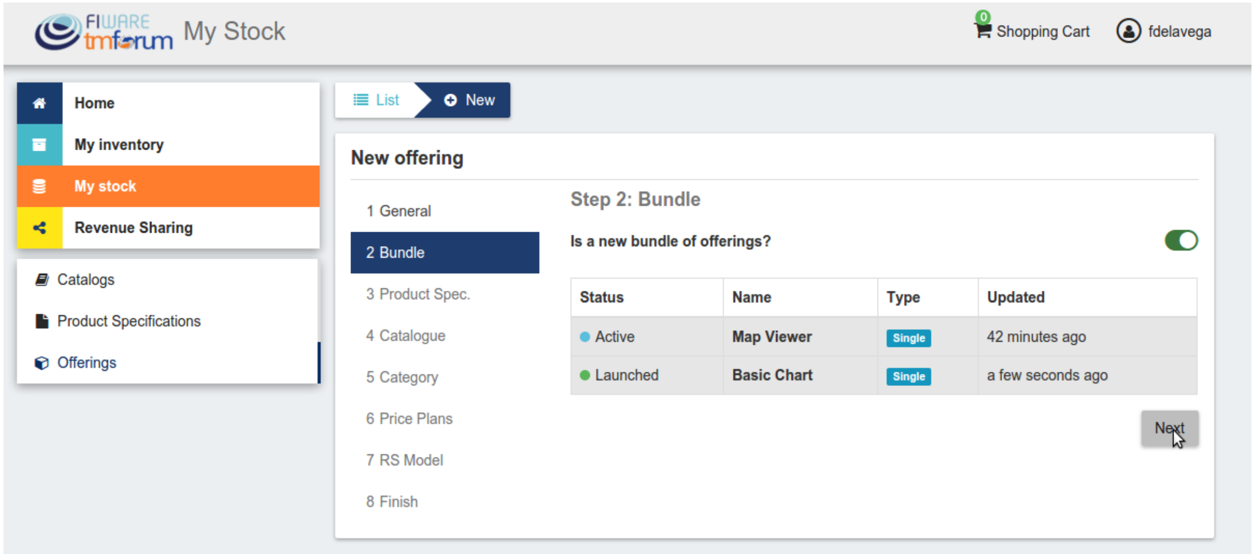
In the displayed form, include the basic info of the offering. Including, its name, version, an optional description, and an optional set of places where the offering is available. Once the information has been provided click on *Next*



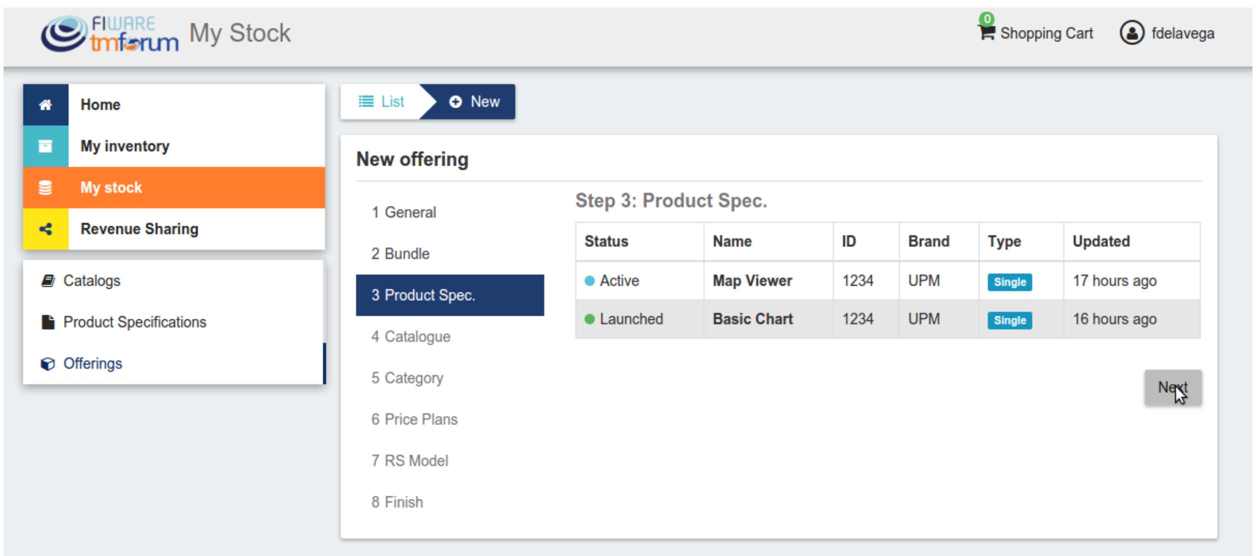
In the next step, you can choose whether your offering is a bundle or not. In this case, offering bundles are logical containers that allow you to provide new pricing models when a set of offerings are acquired together. Once selected click on *Next*



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

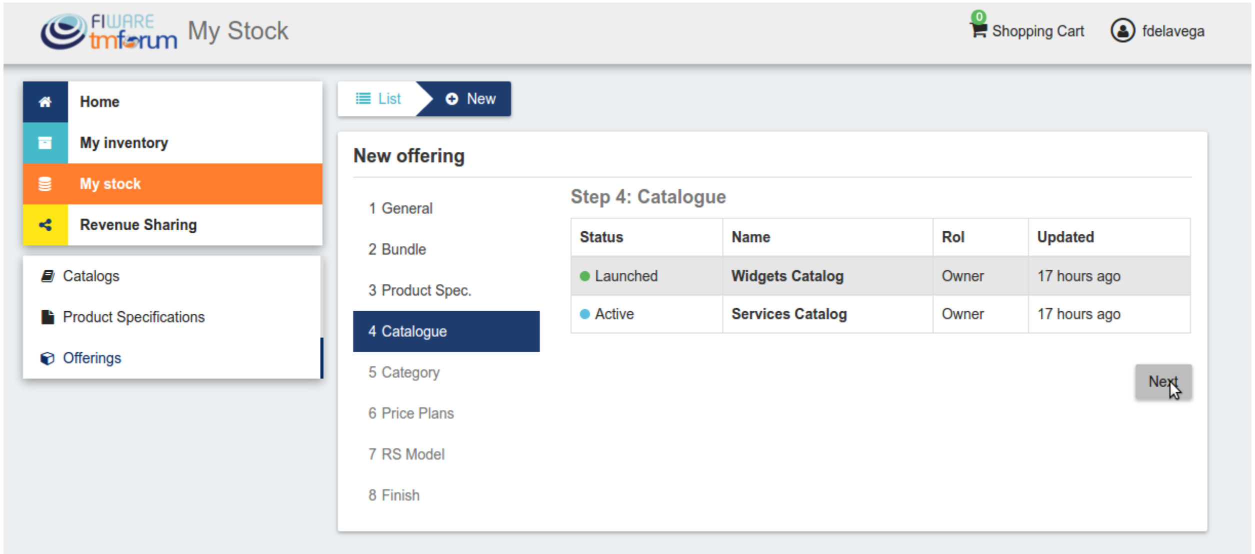


In the next step you have to select the product specification that is going to be monetized in the current offering. Once selected click on *Next*.

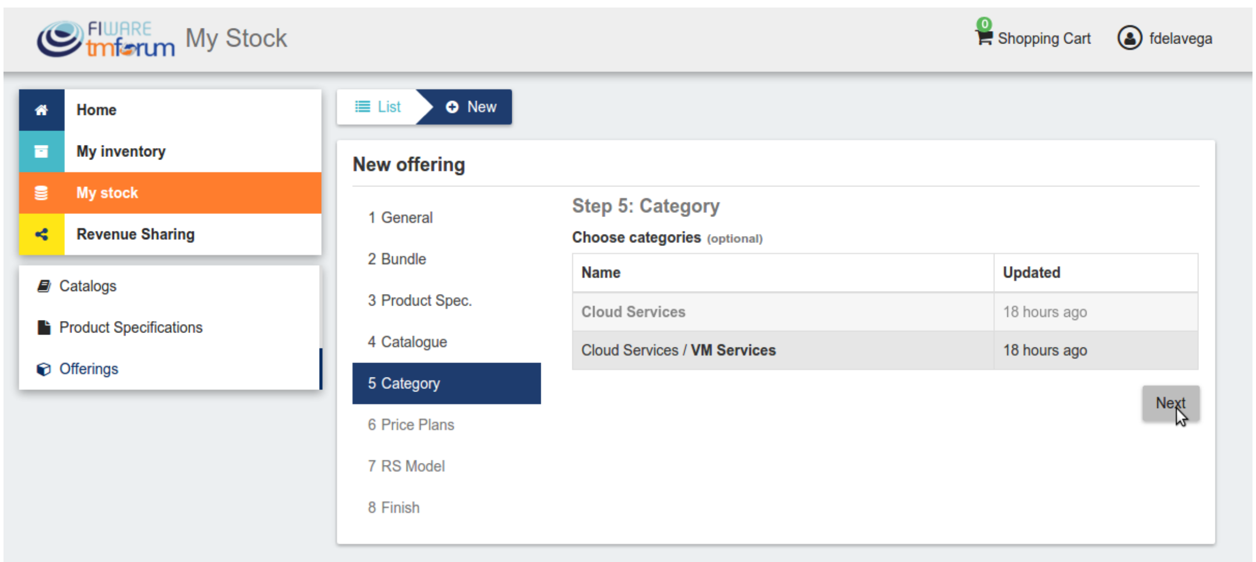


Note: If you are creating an offering bundle, you will not be allowed to include a product specification

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

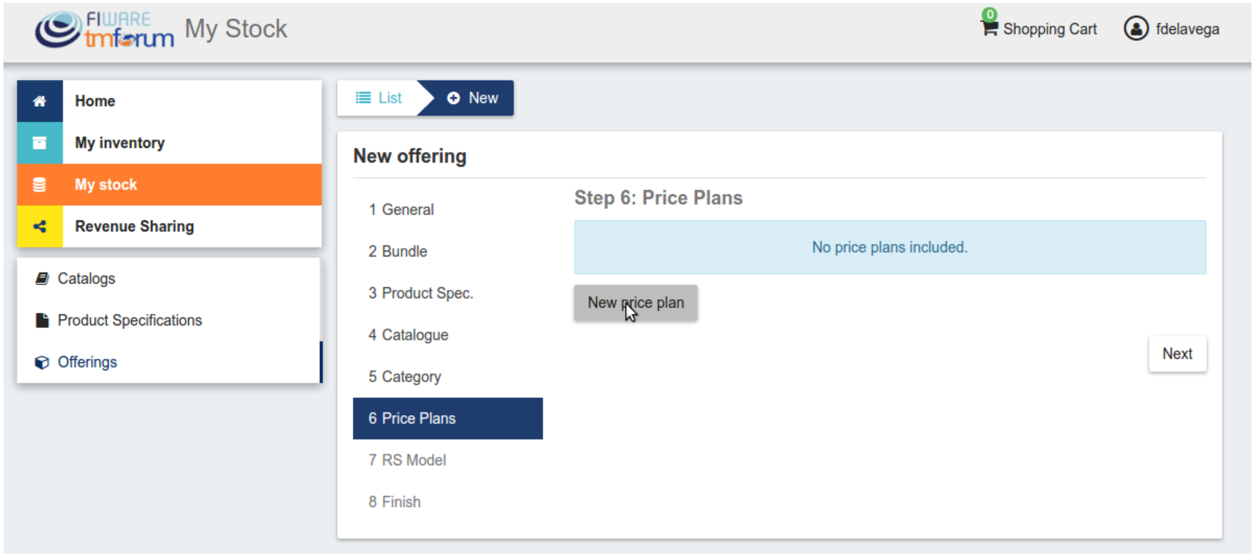


In the next step, you can optionally choose categories for you offering. Once done, click on *Next*



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

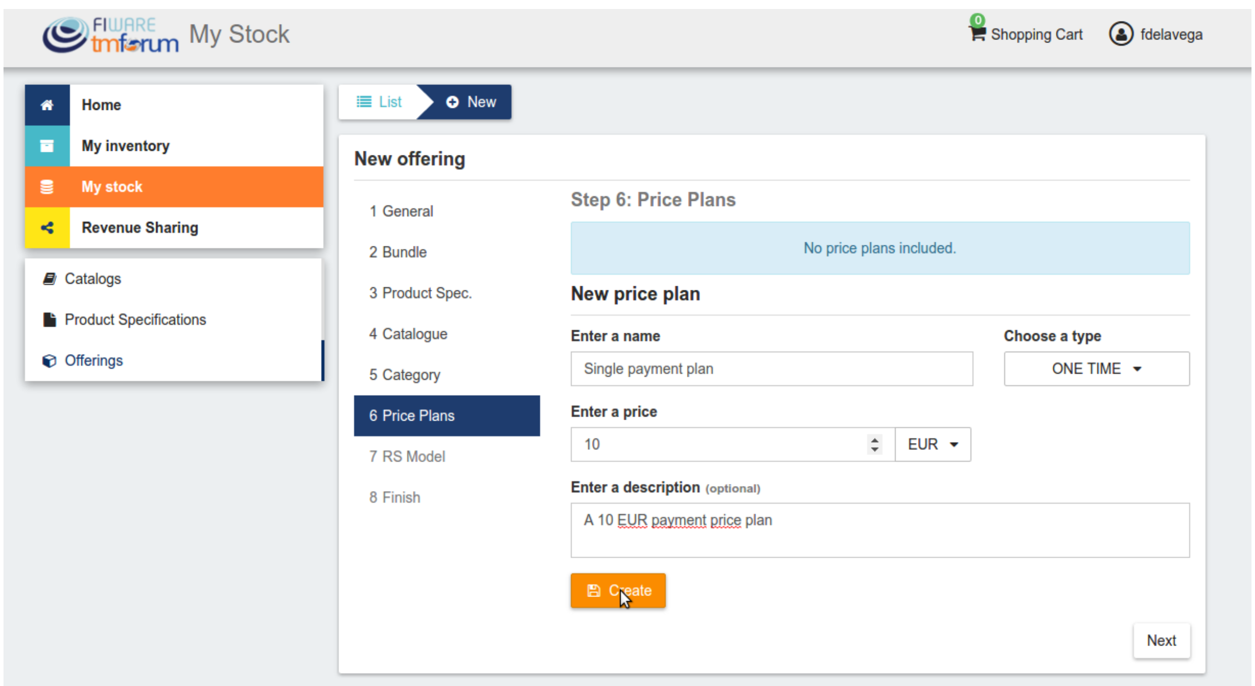
To include a new price plan the first step is clicking on *New Price Plan*



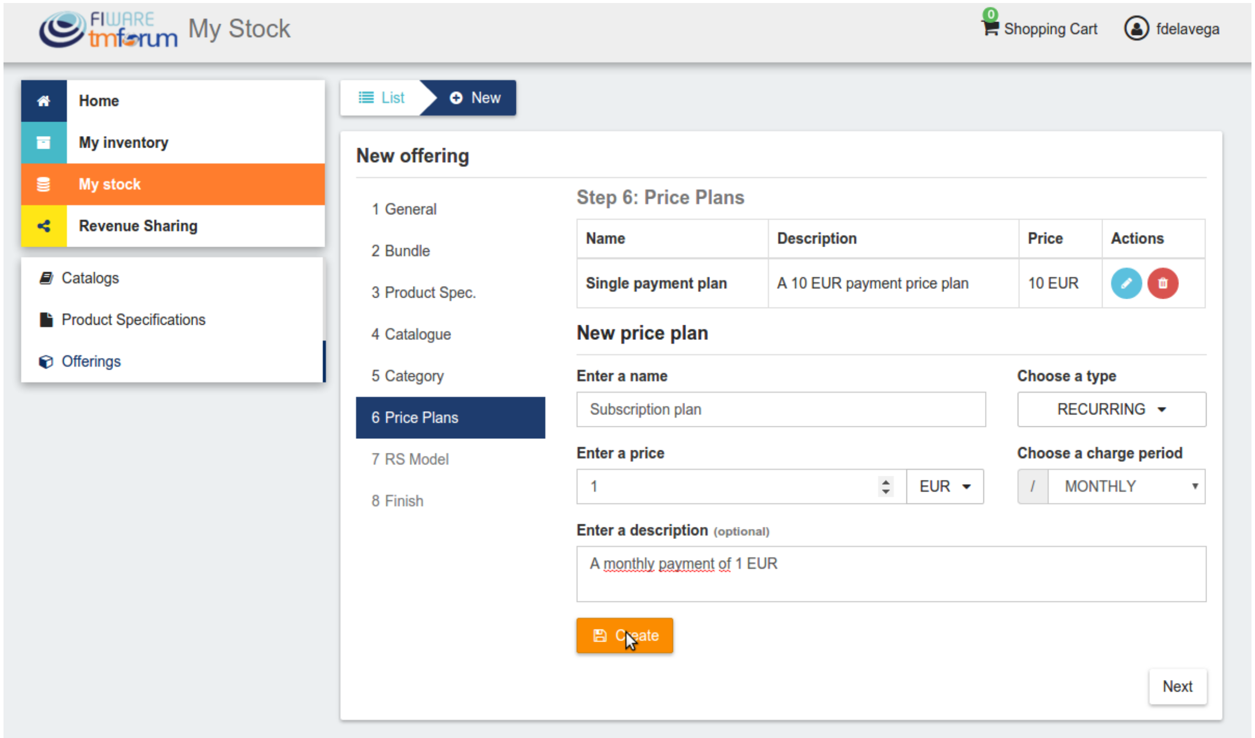
For creating the price plan, you have to provide a name, and an optional description. Then, you have to choose the type of price plan between the provided options.

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

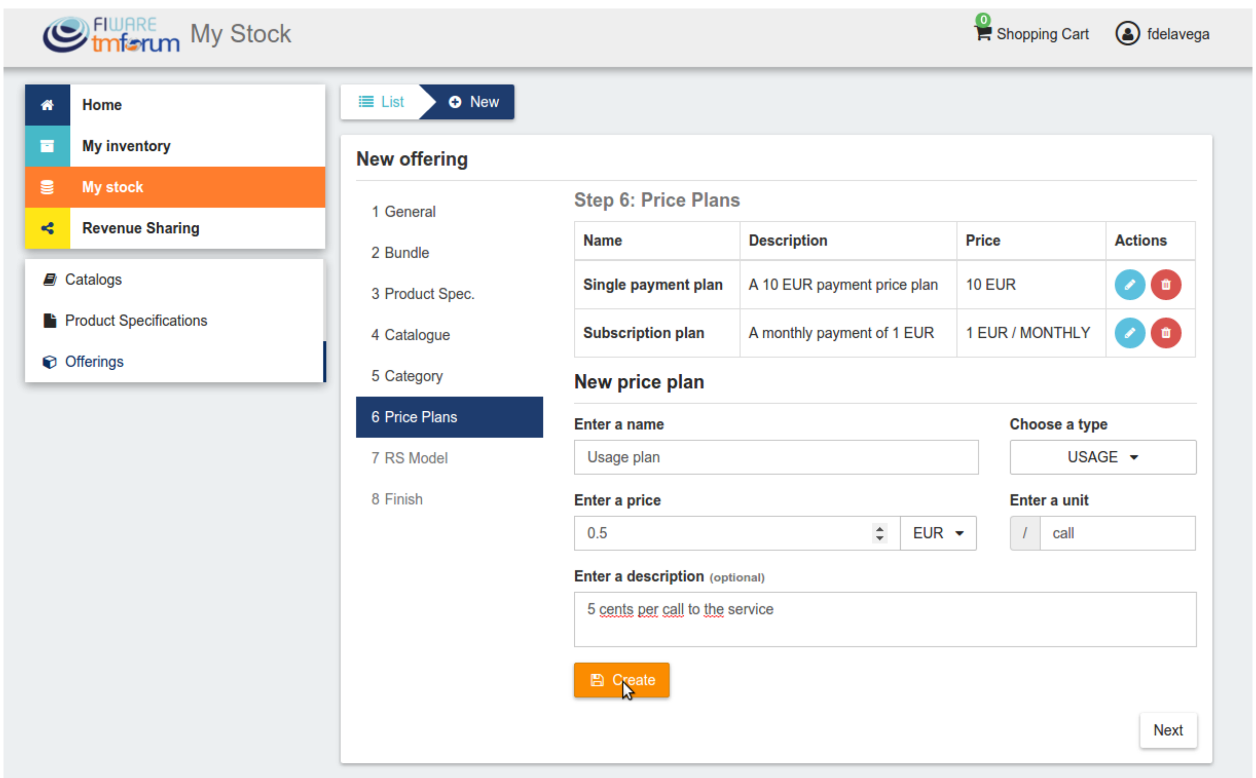
If you choose *one time*, you have to provide the price and the currency.



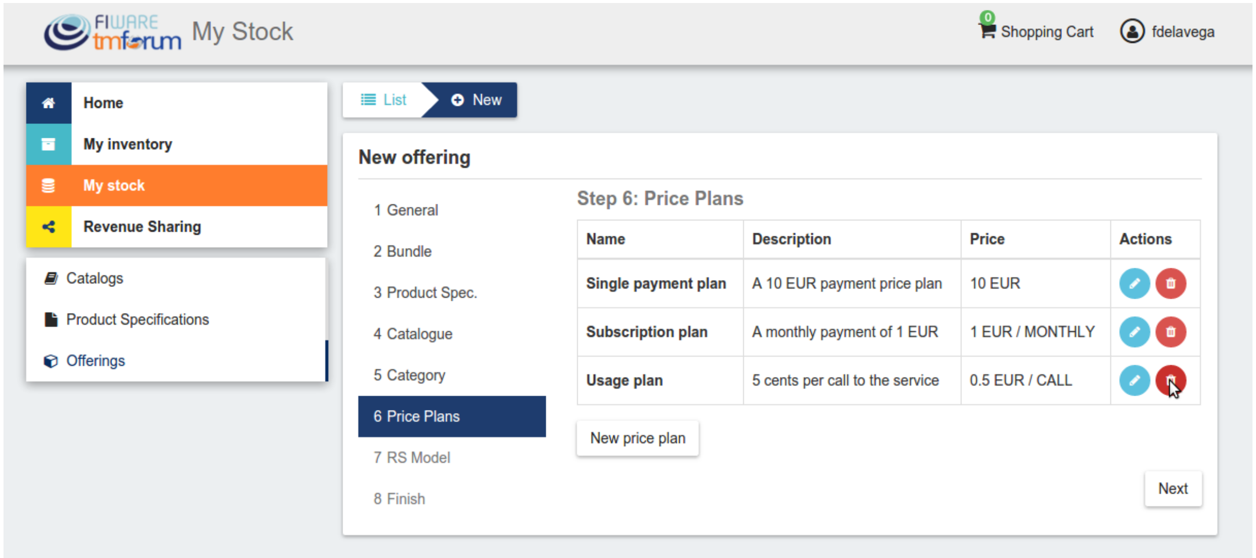
If you choose *recurring*, you have to provide the price, the currency, and the period between charges.



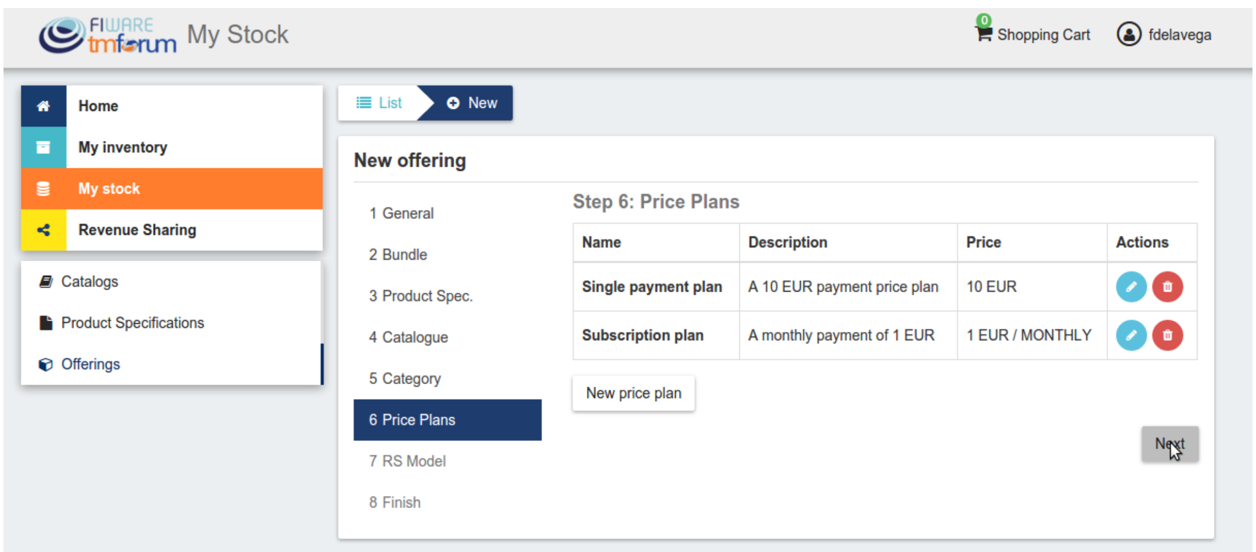
If you choose usage, you have to provide the unit to be accounted, the currency, and the price per unit



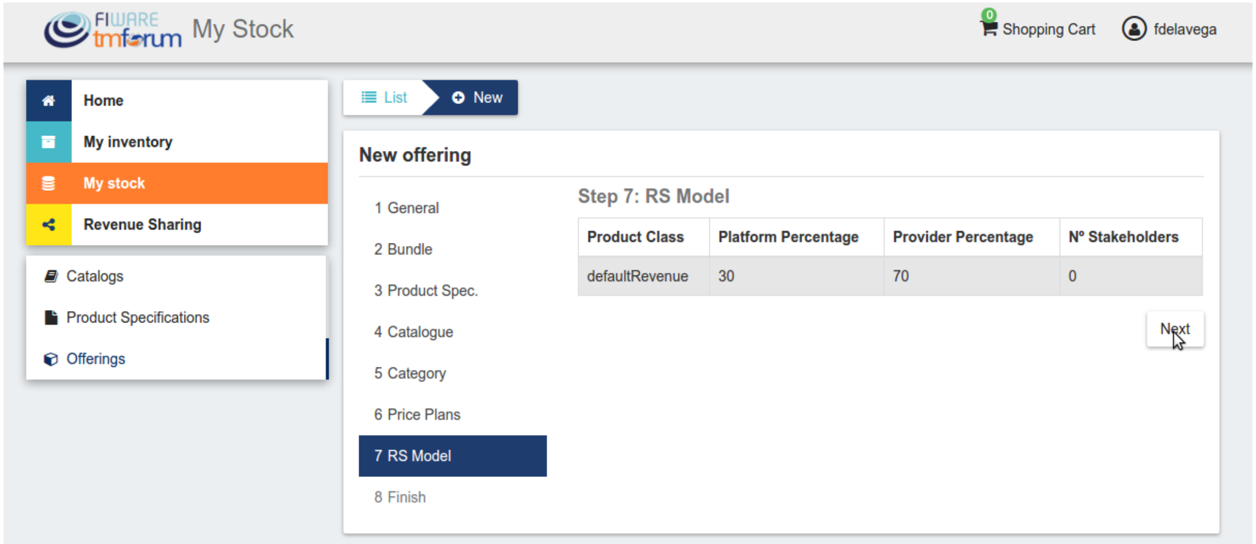
You can update or remove plans by clicking on the corresponding action button.

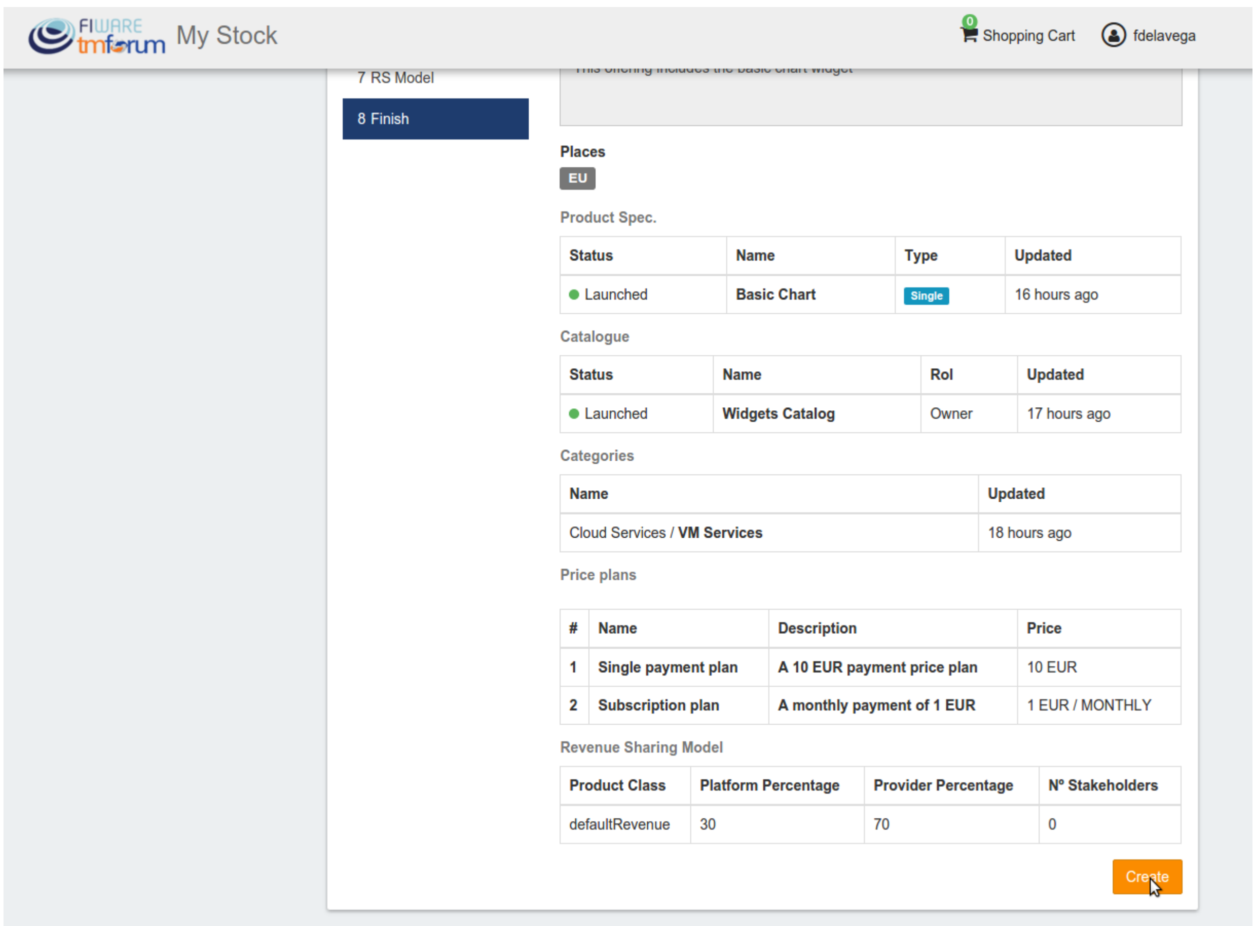


Once you have created you pricing model click on *Next*

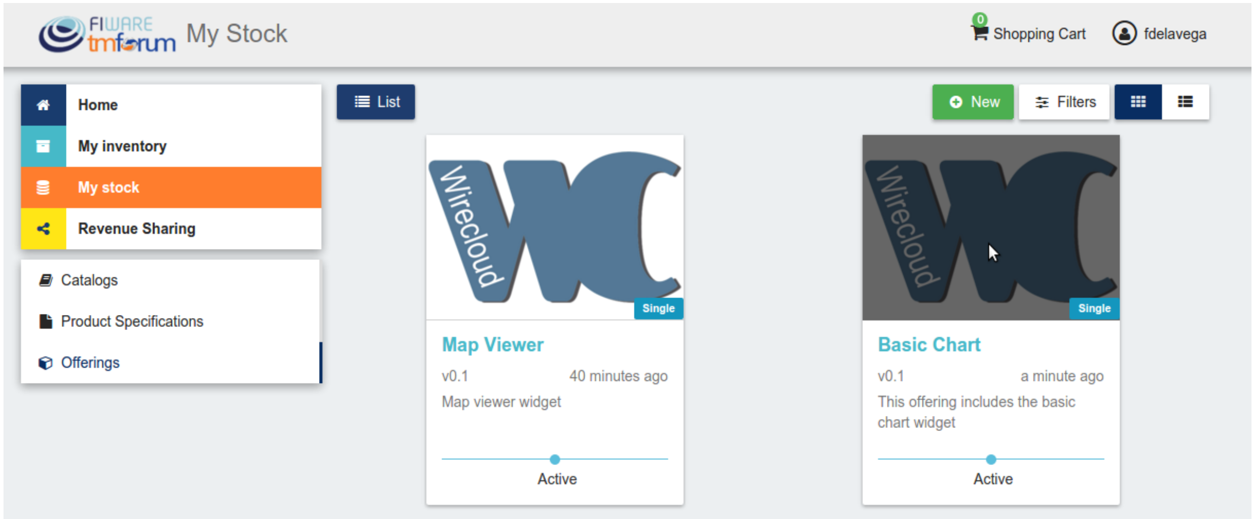


In the last step of the process, you have to choose the revenue sharing model to be applied to you offering between the available ones. Once done, click on *Next* and then on *Create*.

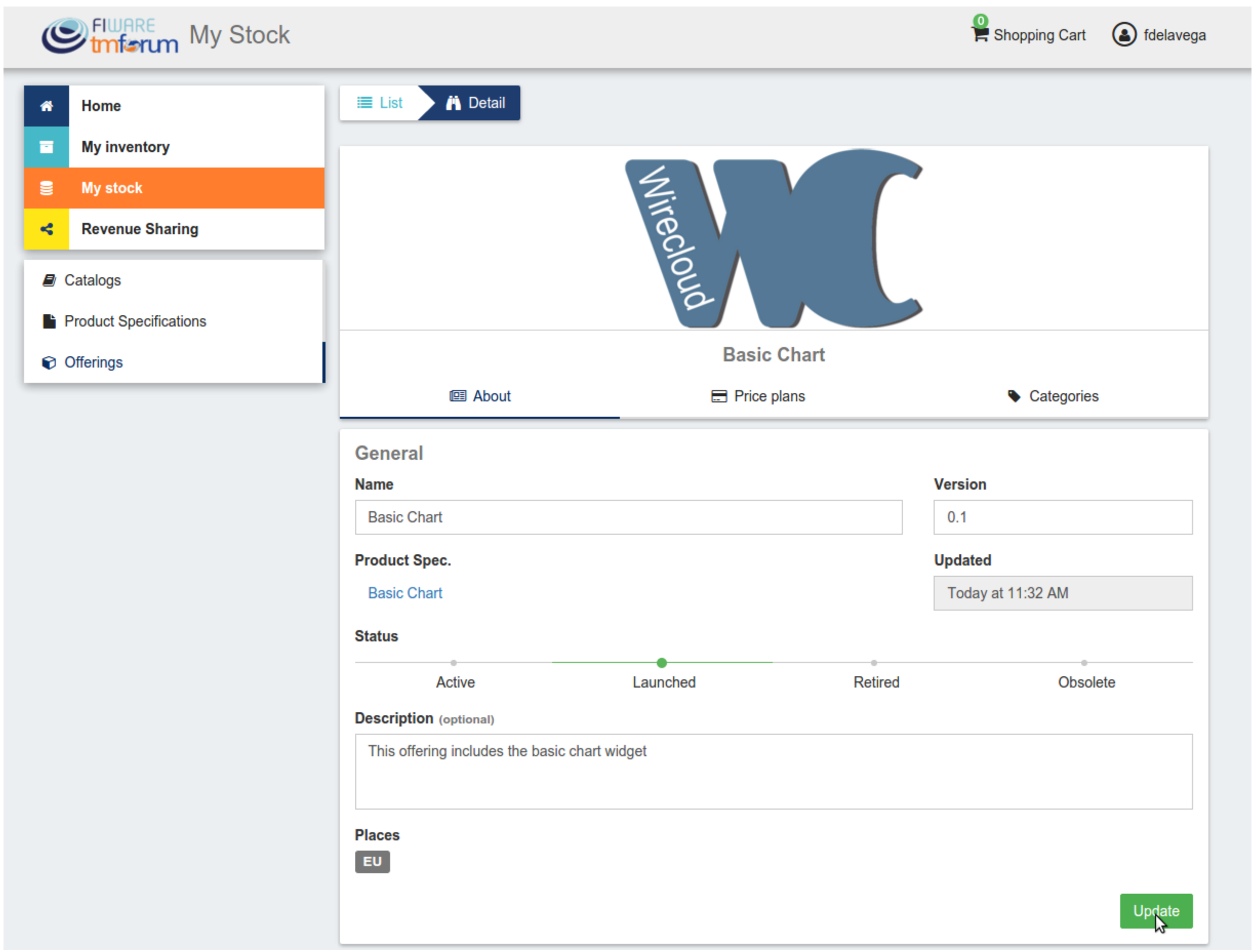




Sellers can also edit their offerings. To do that click on the offering to be updated.



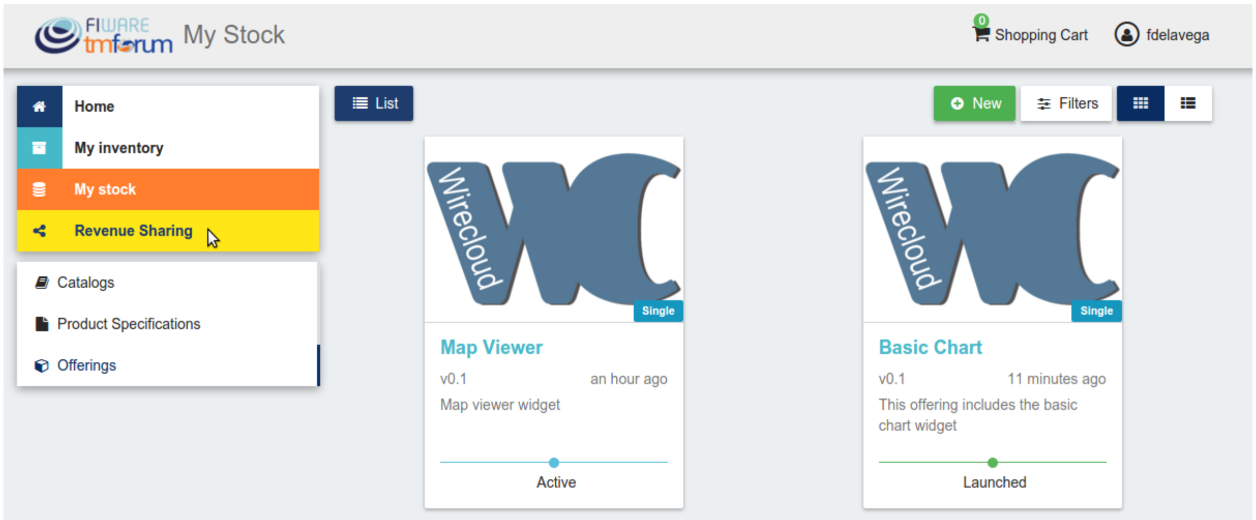
In the displayed form, change the fields you want to edit and click on *Update*. Note that for start selling you offering you have to update its status to *Launched*



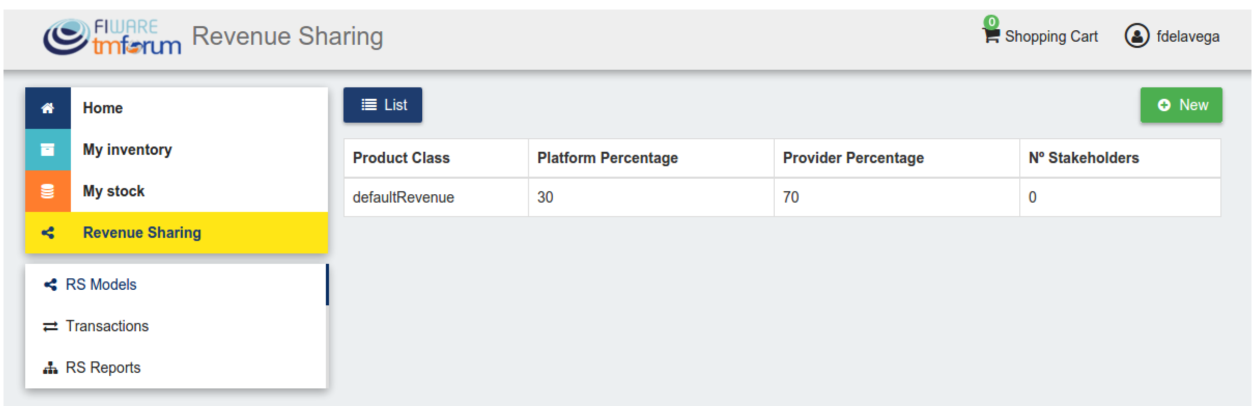
Manage Revenue Sharing Models

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.

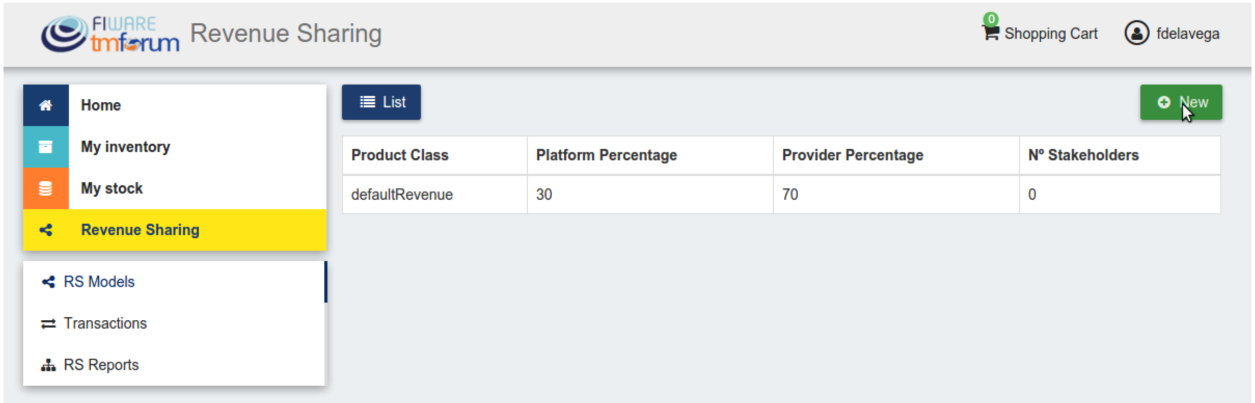
To manage RS models go to the *Revenue Sharing* section.



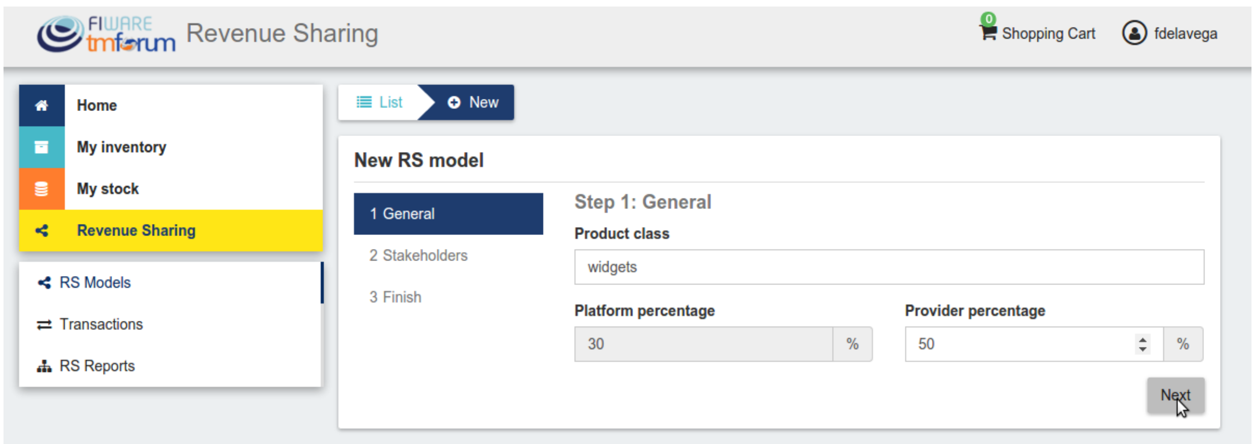
In this view, you can see the revenue sharing models you have available. By default it will appear the default RS model which establishes the revenue distribution between you and the Business API Ecosystem instance owner.



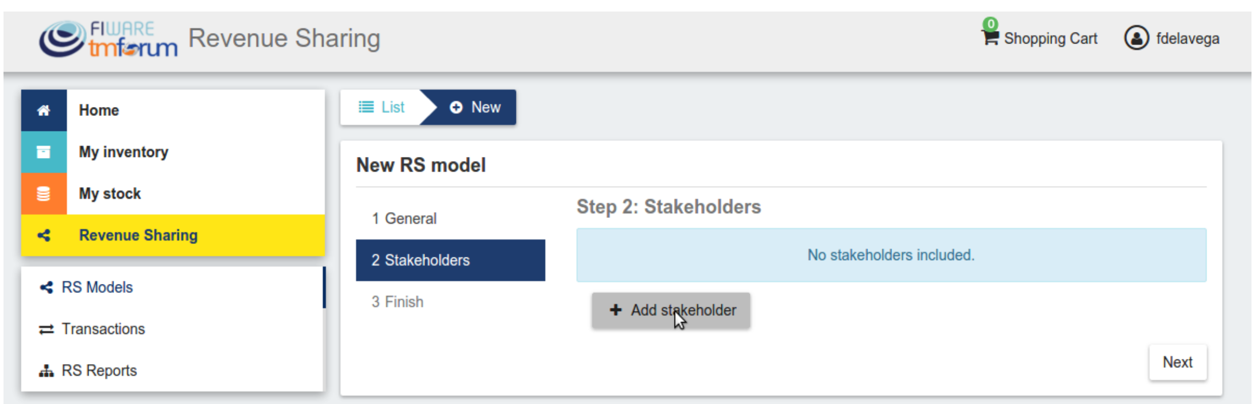
You can create a new RS model clicking on *New*



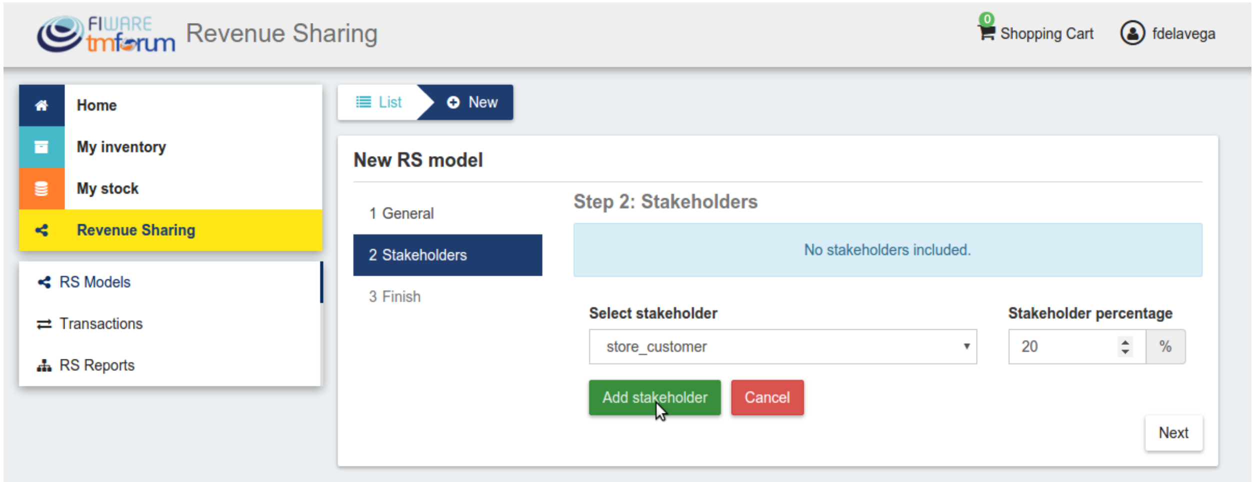
In the first step of the process you have to provide a product class, which identifies the RS model, and the percentage you want to receive. The platform percentage is fixed and cannot be modified. Once provided click on *Next*



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

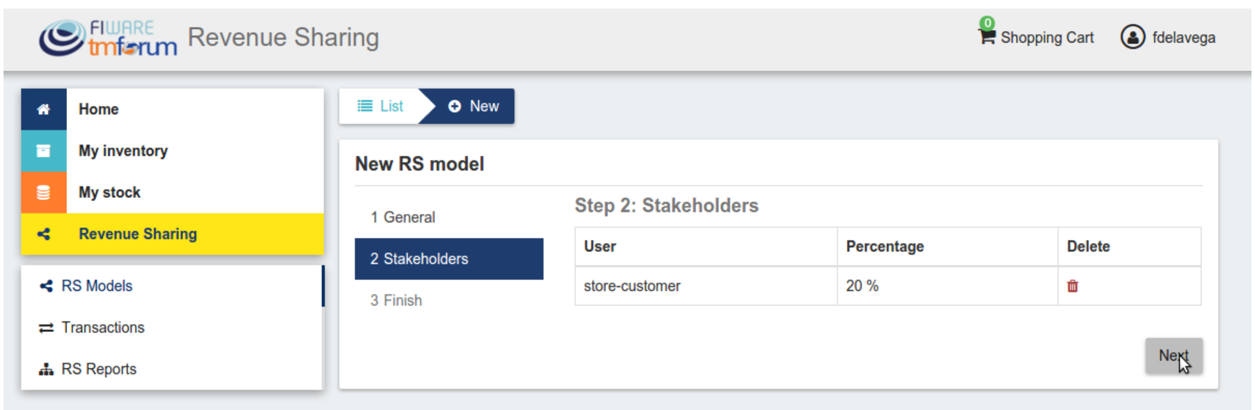


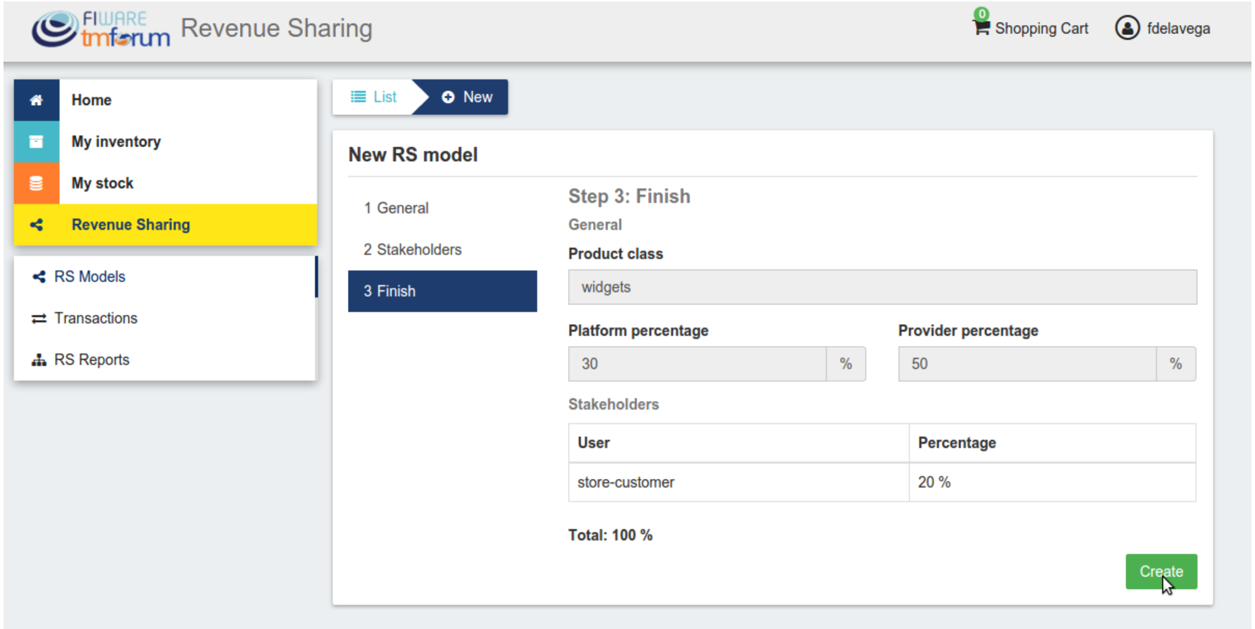
Then, select the Stakeholder between the available users, and provide its percentage. Finally, save it clicking on *Add Stakeholder*



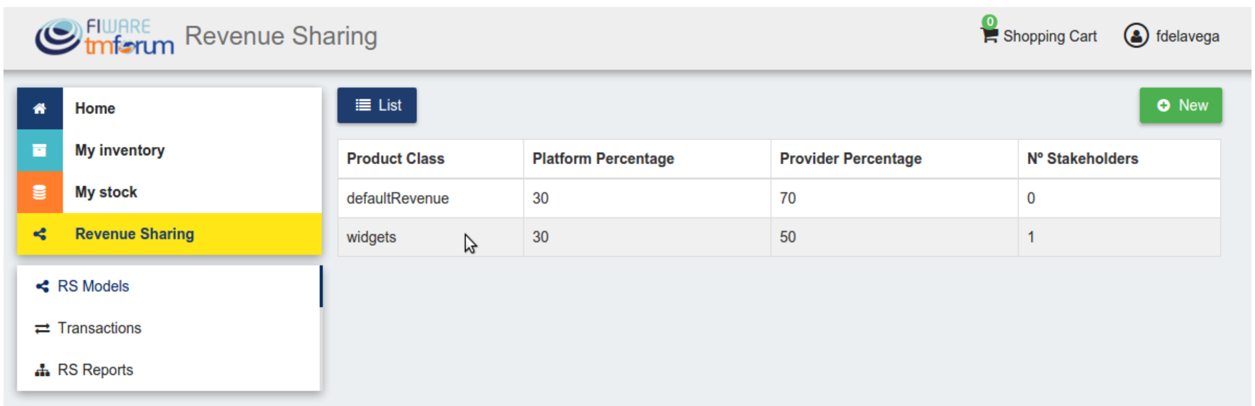
Note: The total percentage (provider + platform + stakeholders) must be equal to 100

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

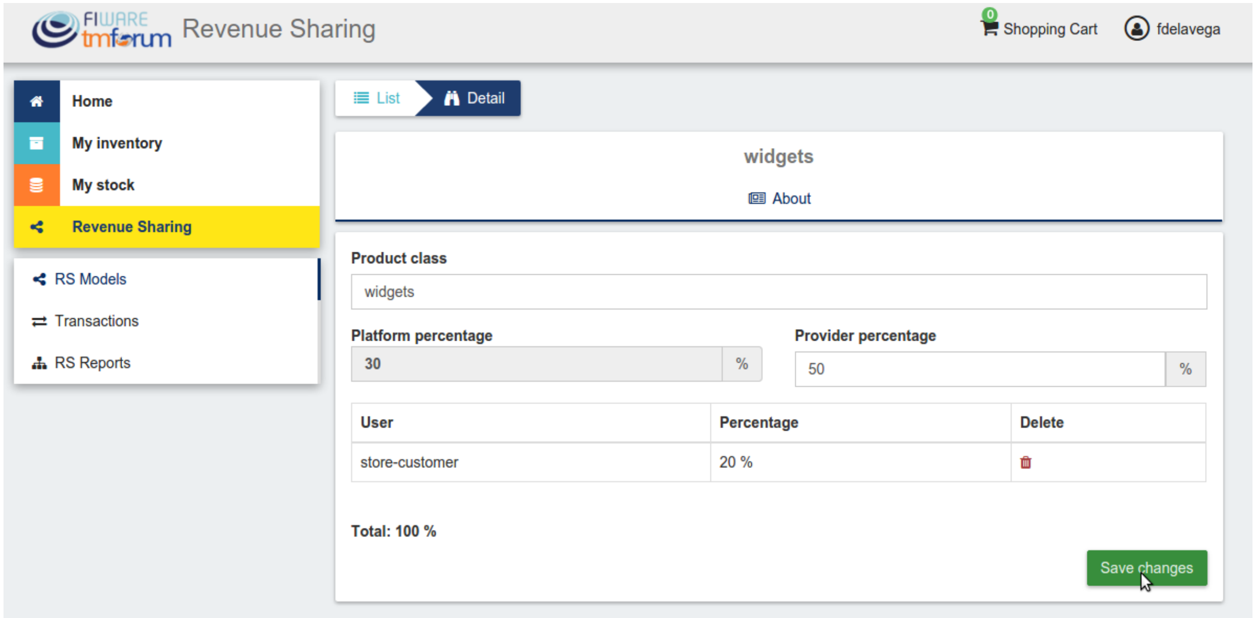




Sellers can also update their RS model. To do that click on the RS model to be updated.

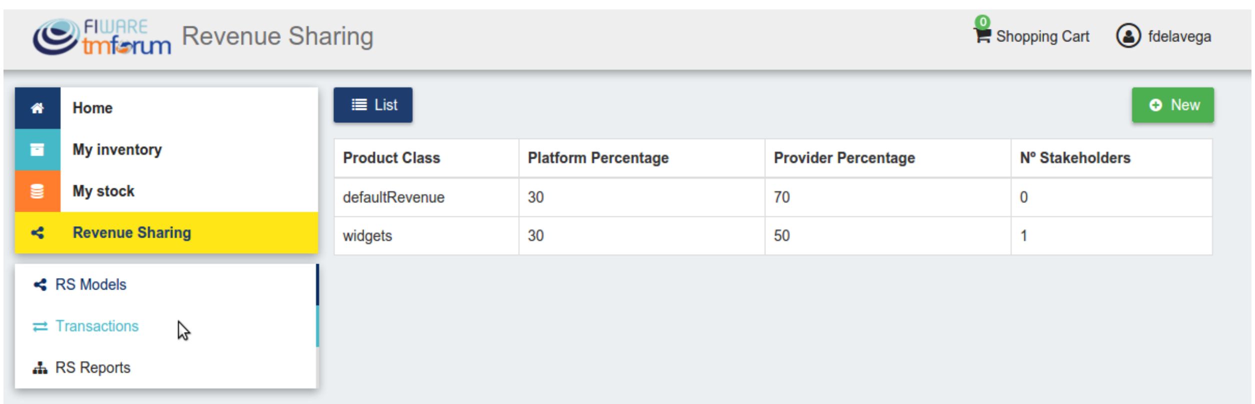


Then, update the required fields (including the stakeholders if you want), and click on *Save Changes*

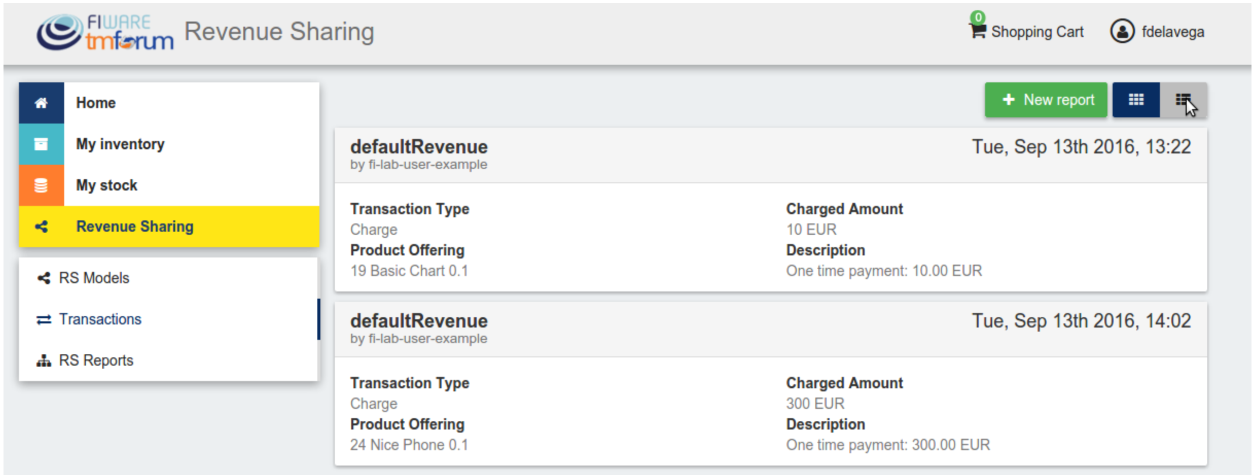


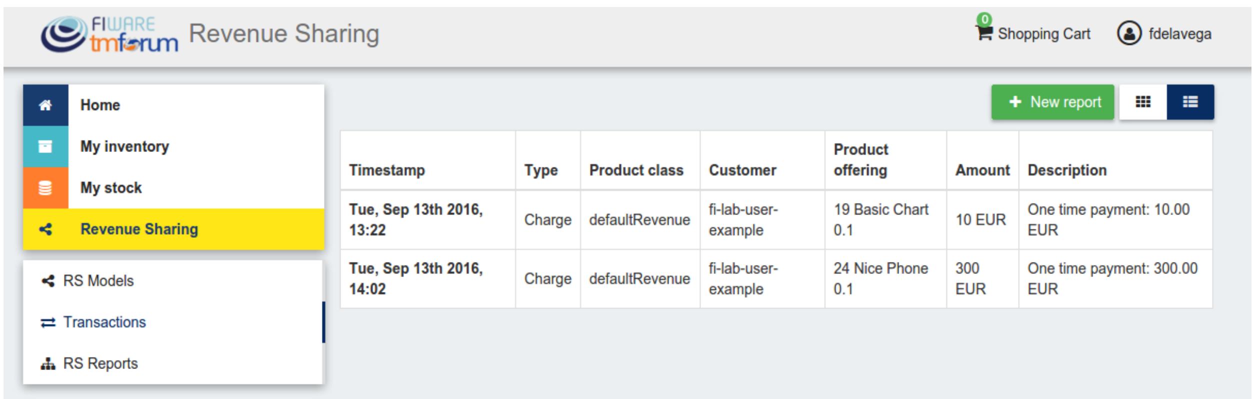
Manage Transactions

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*



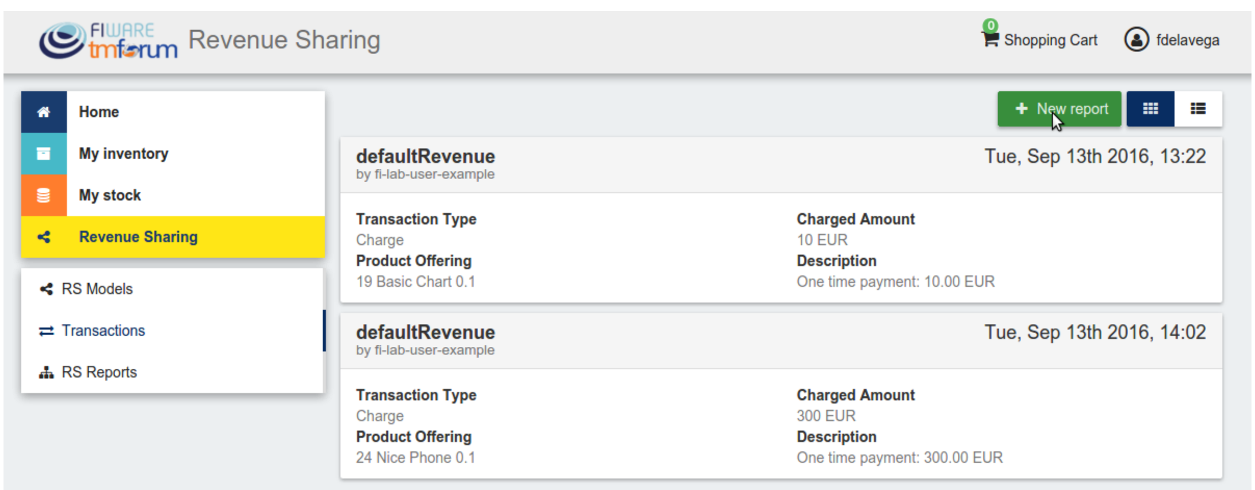
In the displayed view, you can see the transactions pending to be paid to you and your stakeholders. It is also possible to display the transactions in tabular way



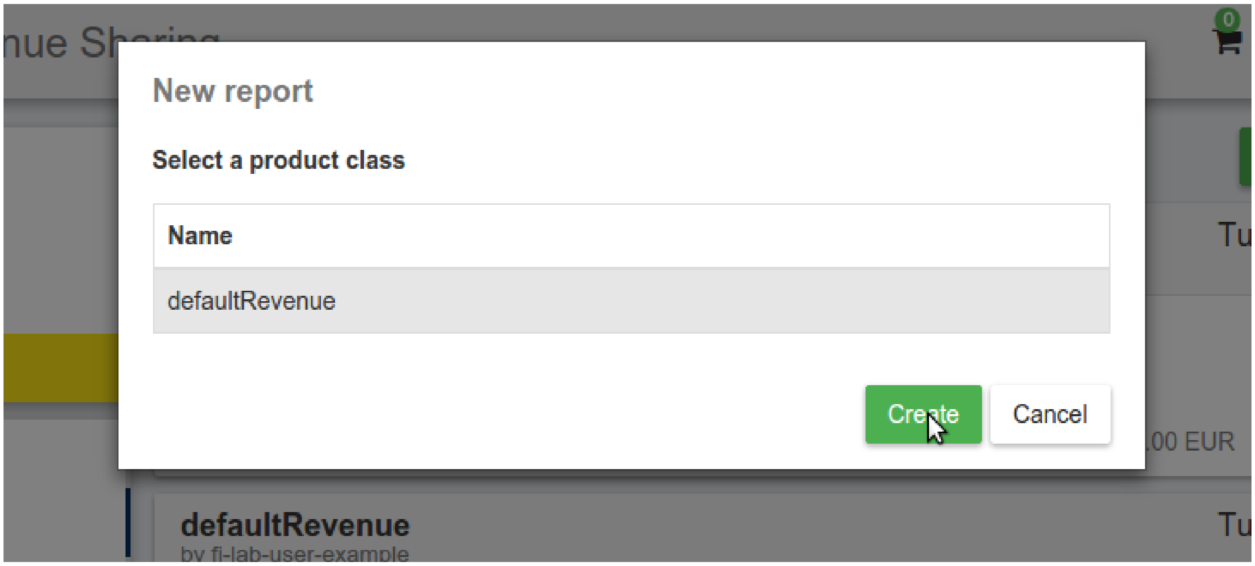


These transactions are aggregated and paid by the Business API Ecosystem periodically once a month. Nevertheless, if you need to be paid, you can force the revenue sharing calculus and payment of your pending transactions by manually generating a revenue sharing report.

To create a new report click on *New Report*

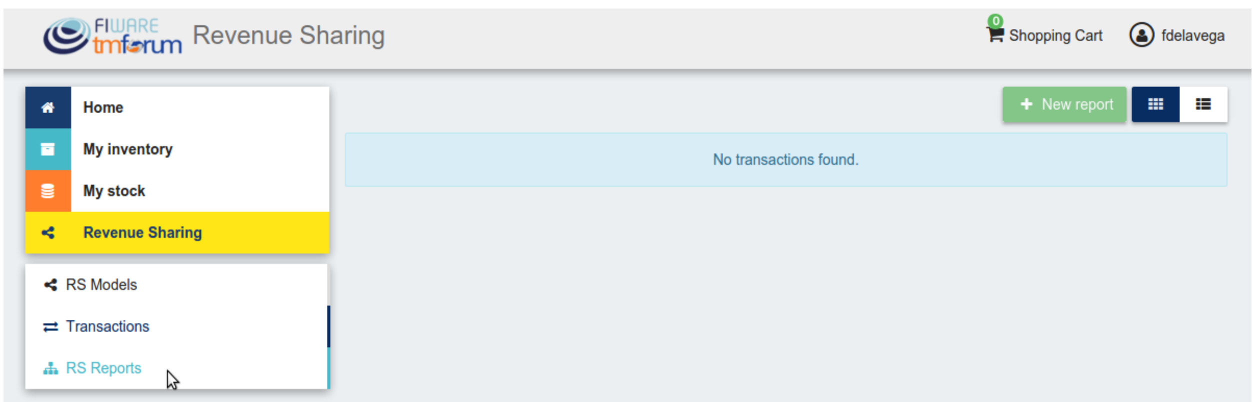


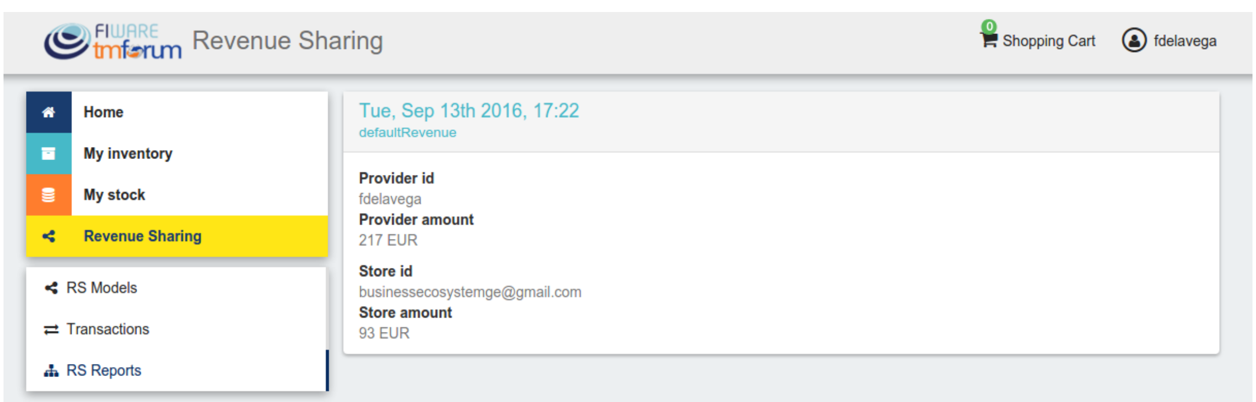
In the displayed modal, choose the product classes to be calculated and click on *Create*



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*



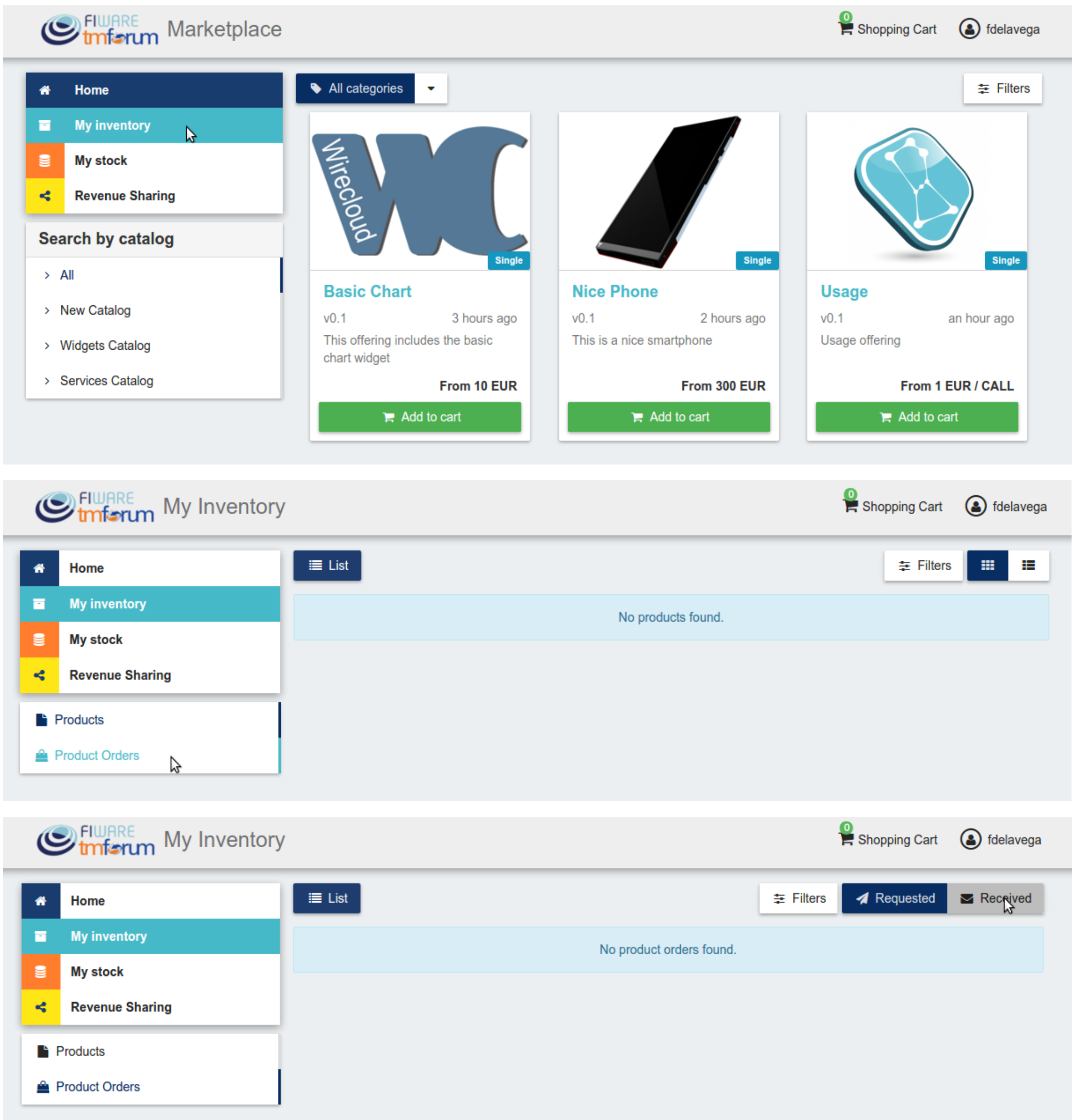


Note: Sellers would need to have a PayPal account associated to the email of their FIWARE IdM account in order to be paid for their products

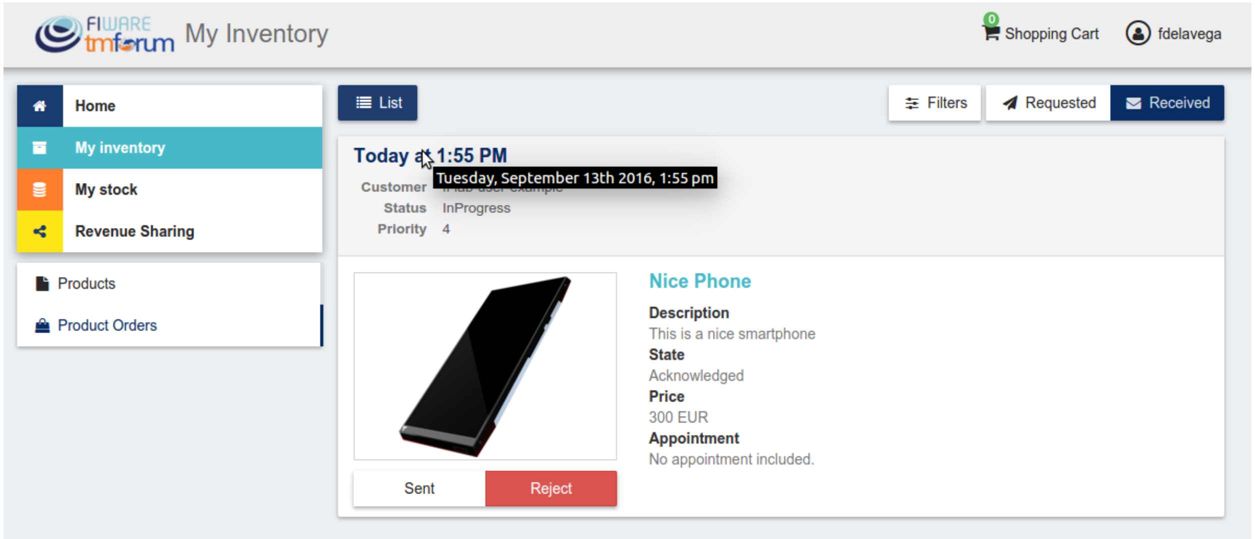
Manage Received Orders

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.

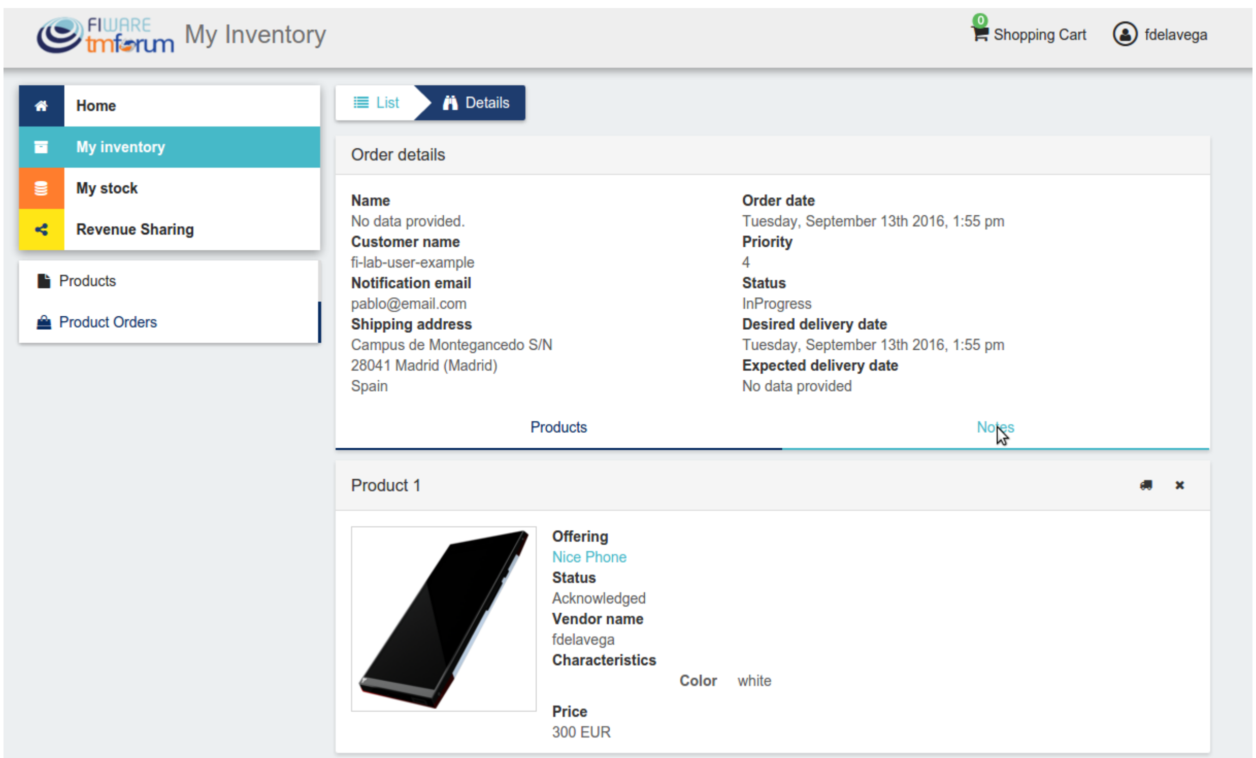


You can view the details of a received order clicking on the order date

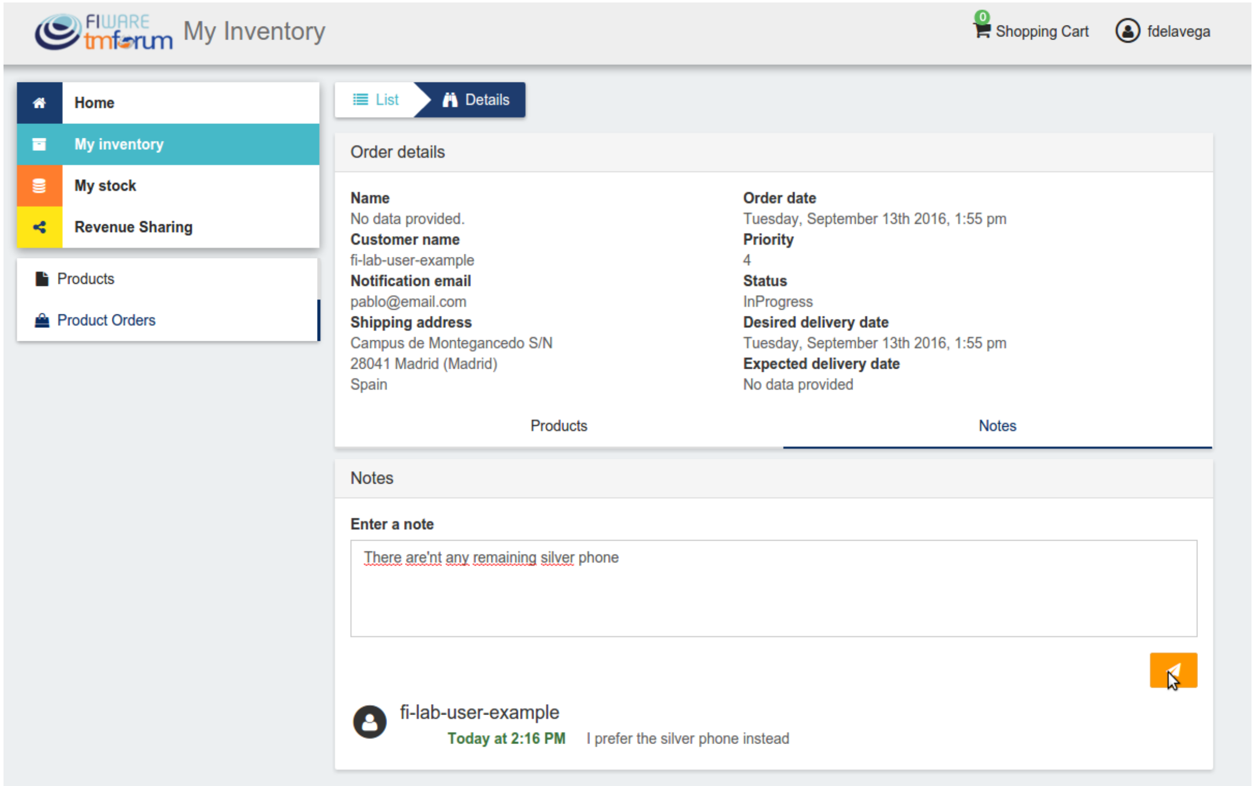


In the displayed view you can review the details of the order and the details of your products acquired by the customer, including the chosen characteristics.

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

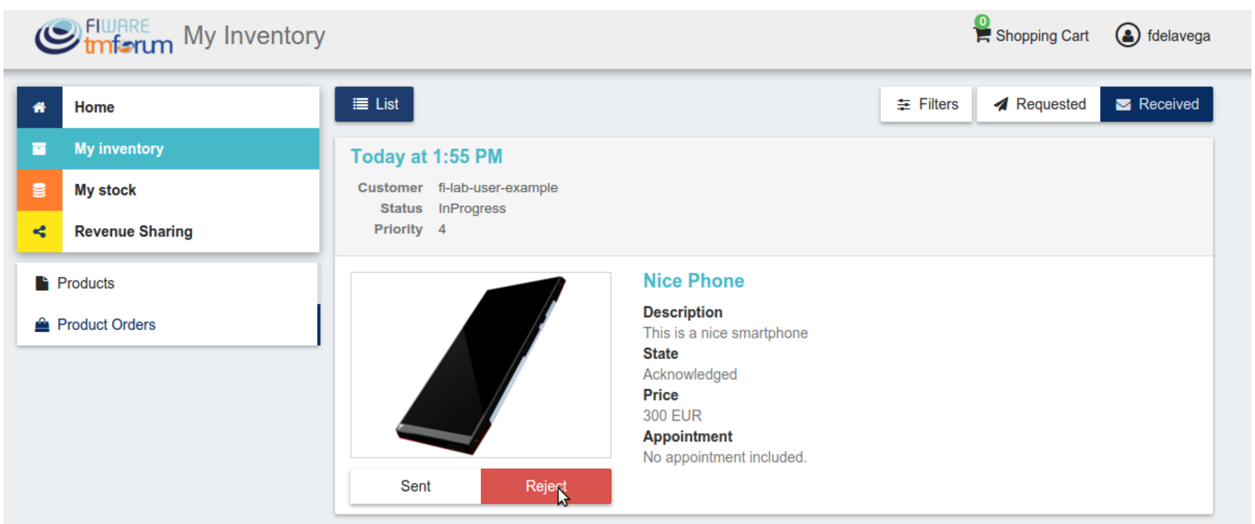


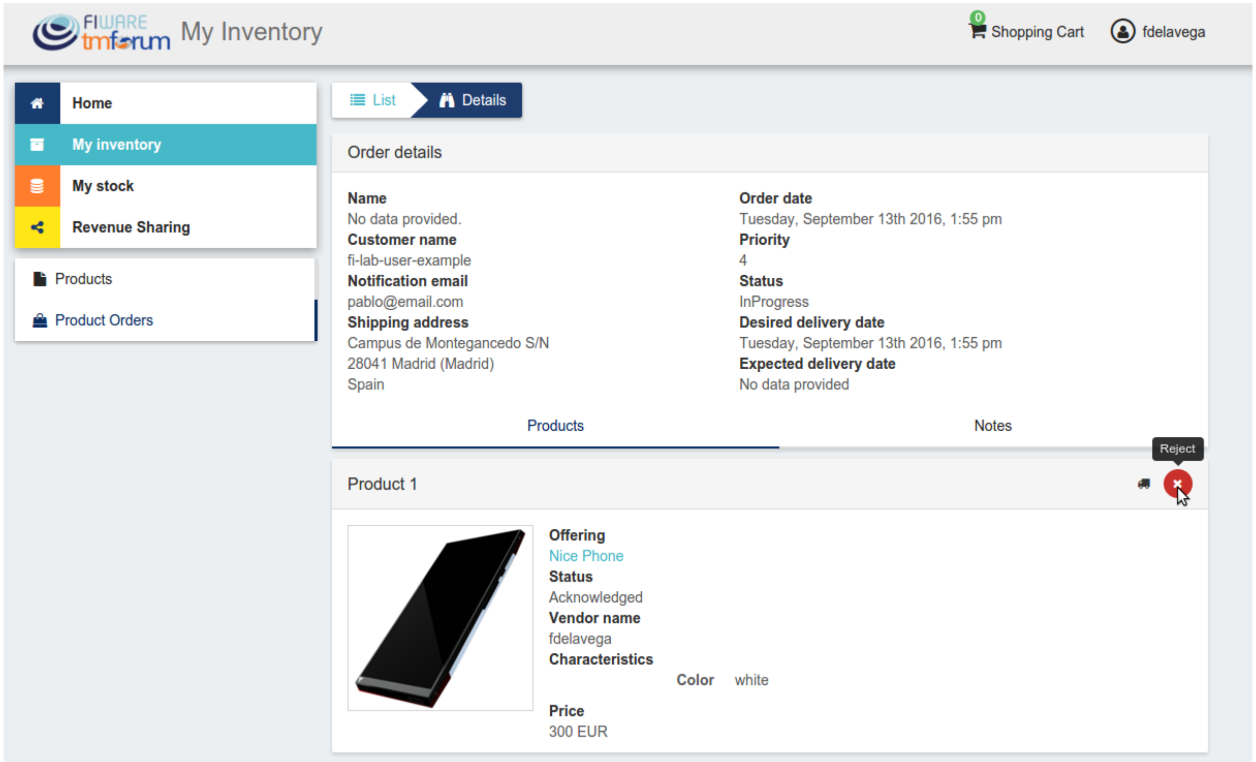
You can also give a reply to customer notes including it in the text area and clicking on the send button



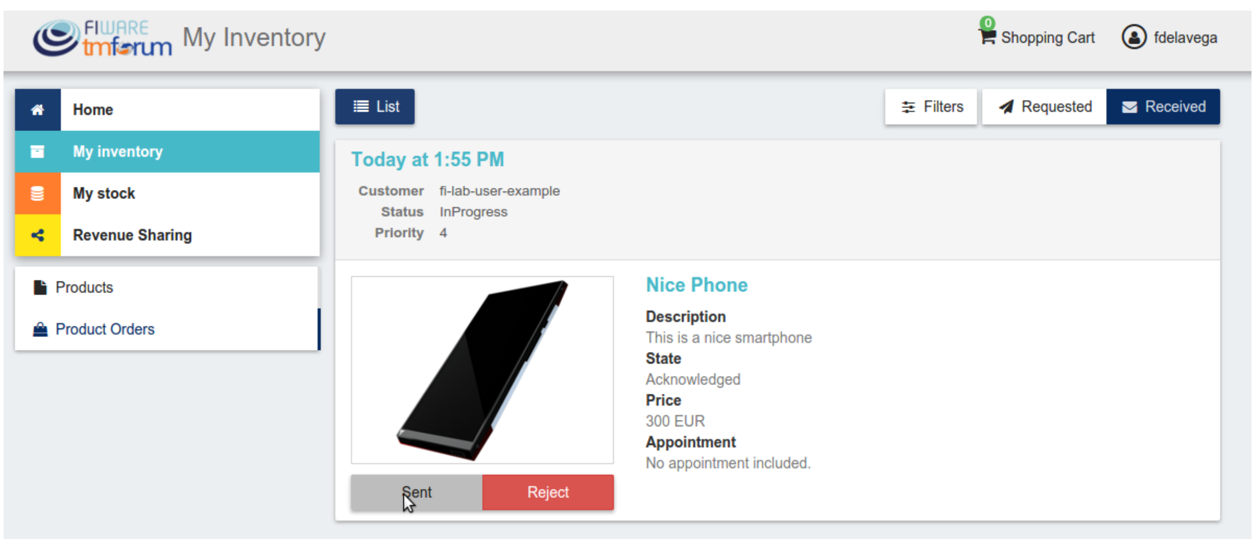
If the acquired product is not digital, the order needs to be processed manually by the seller, in the sense that the seller will have to send the acquired product to the customer. To deal with this situation, the order details view allows sellers to manually change the status of the order.

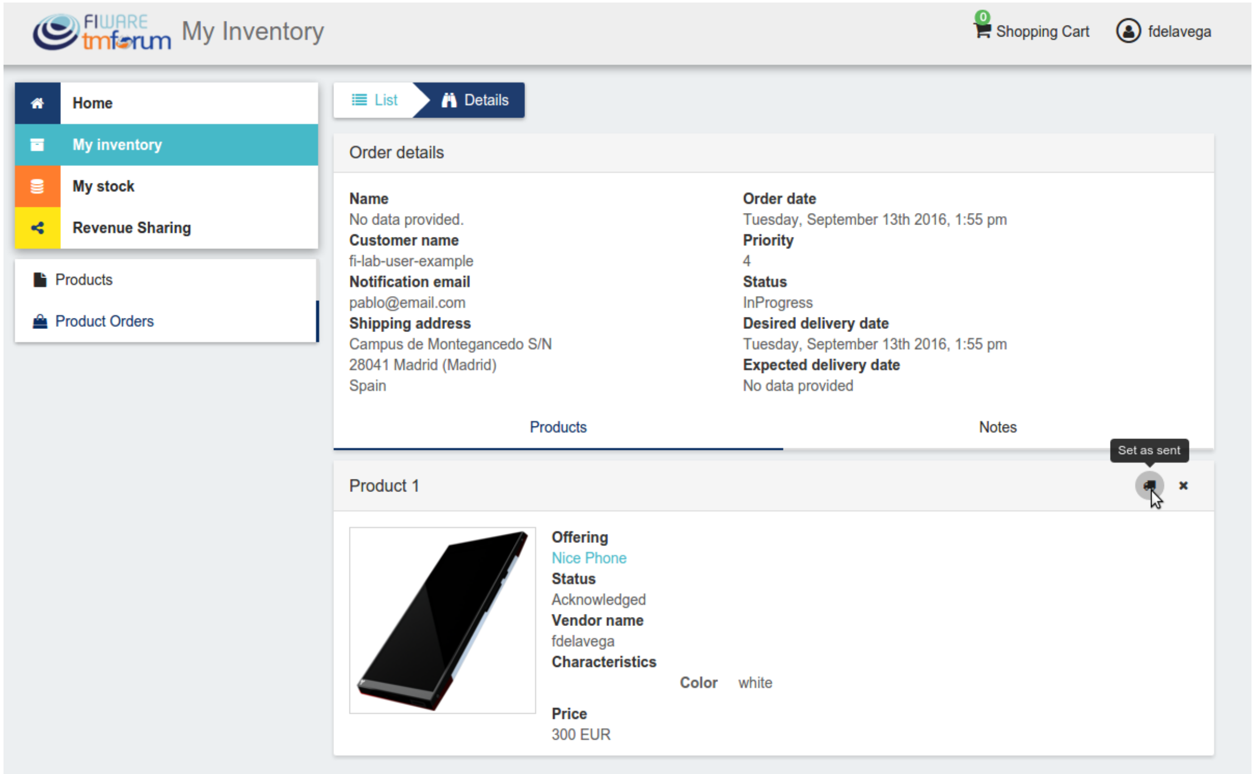
To reject a received order you have to click in the *Reject* button located in the search or in the details view of the order.



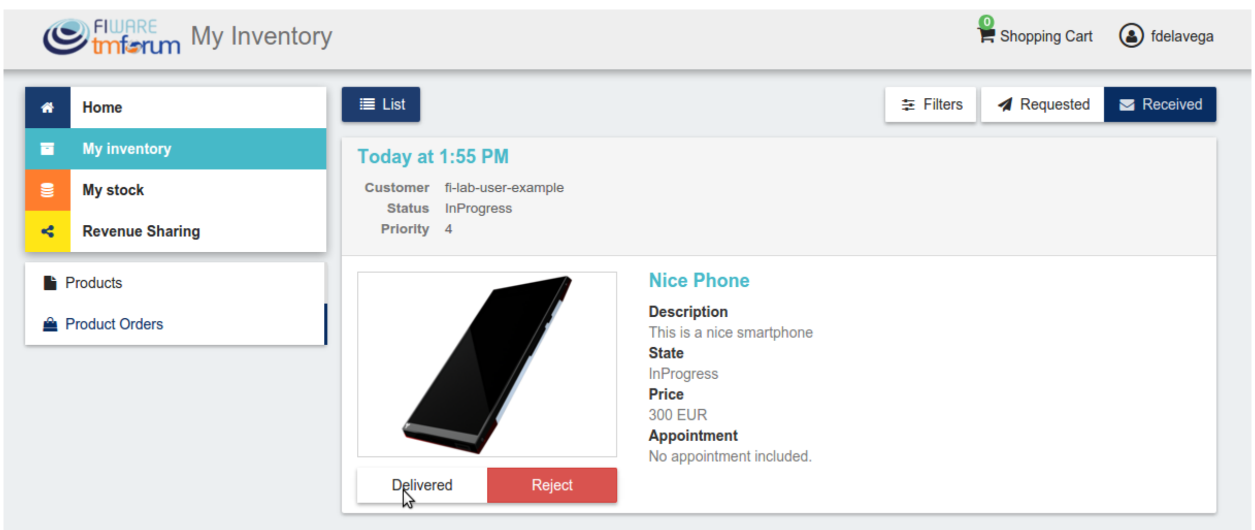


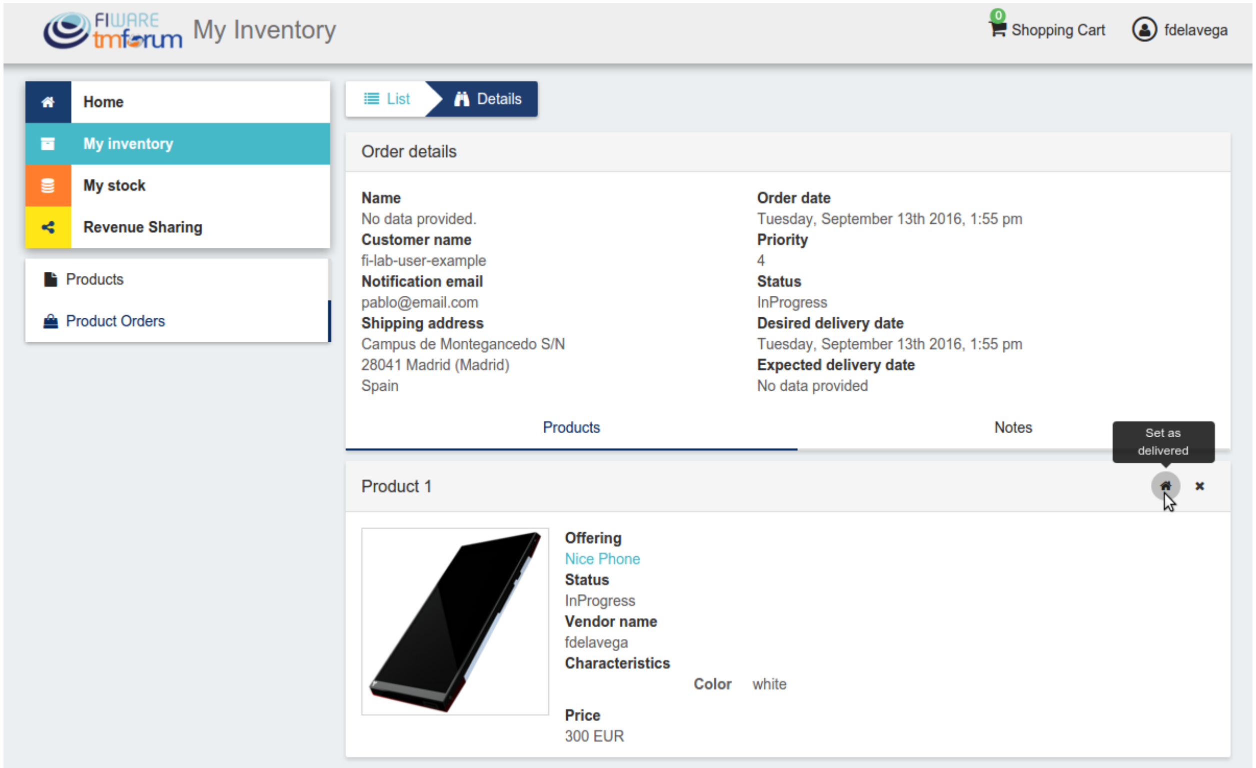
In case you accept the order and send the product to the customer, you have to put it as *inProgress* clicking on the *Sent* button





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



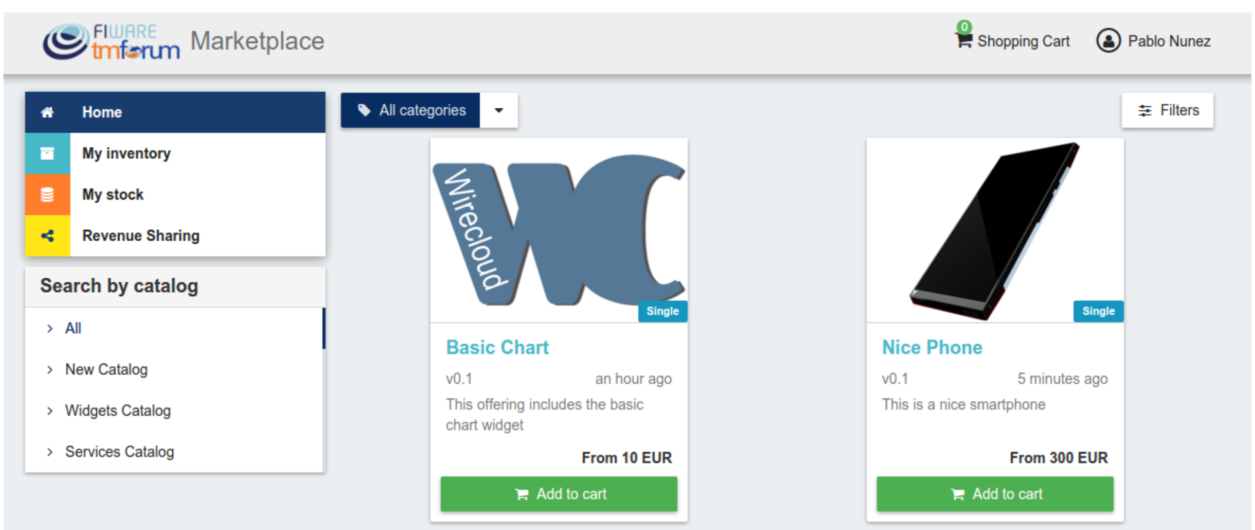


Customer

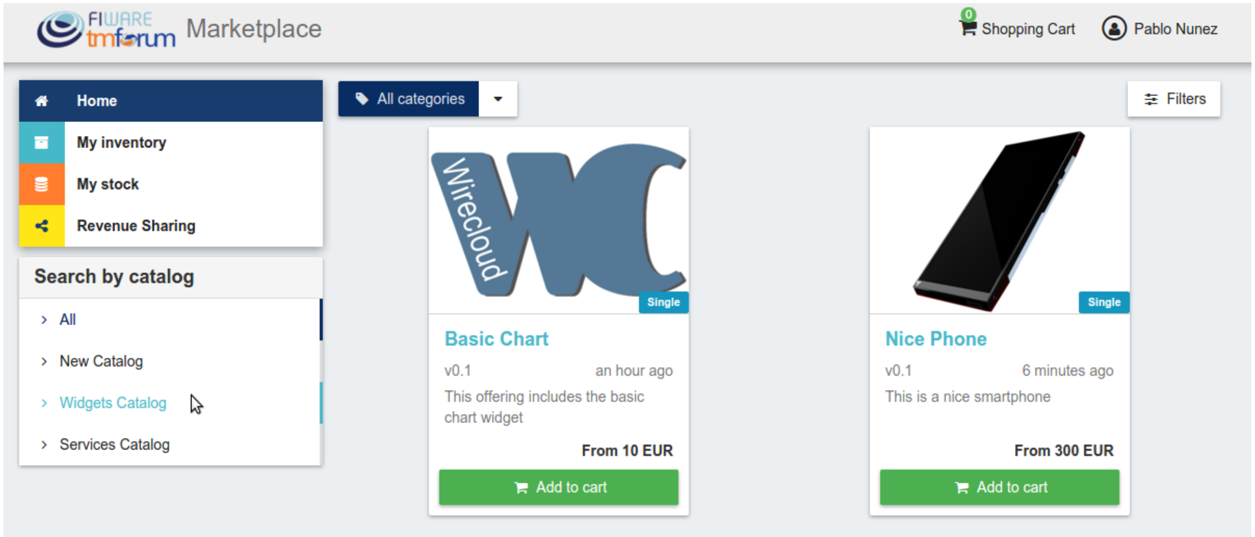
All of the users of the system have by default the *Customer* role. Customers are able to create orders for acquiring offerings.

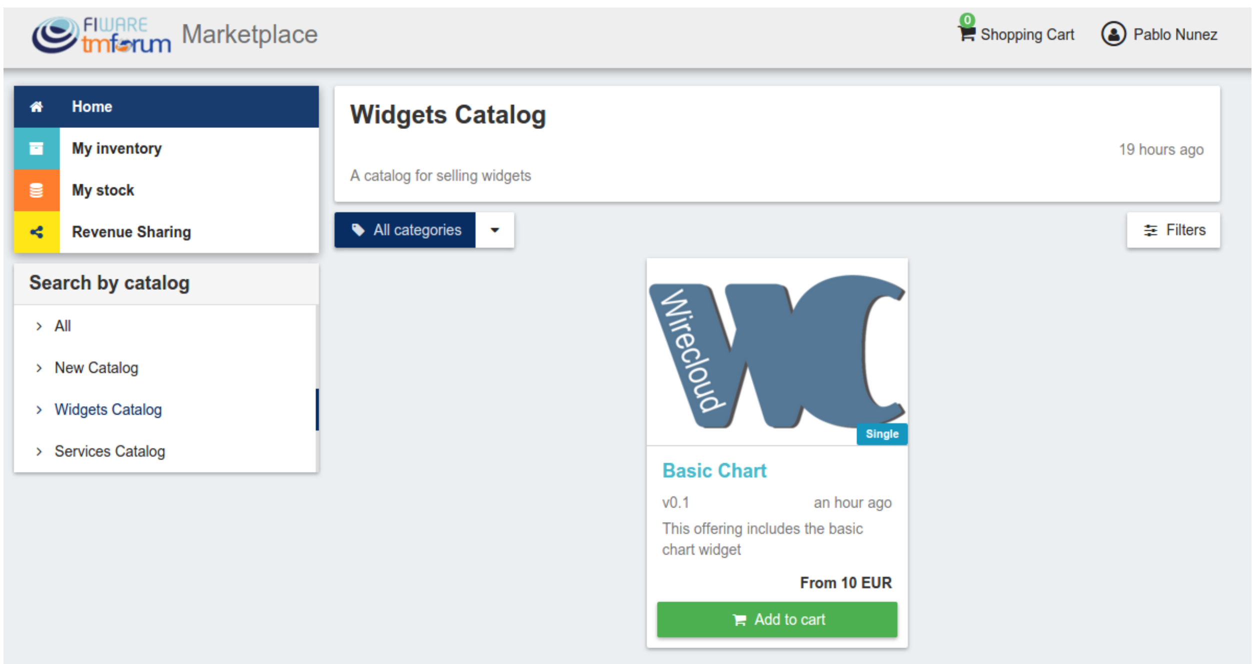
List Available Offerings

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

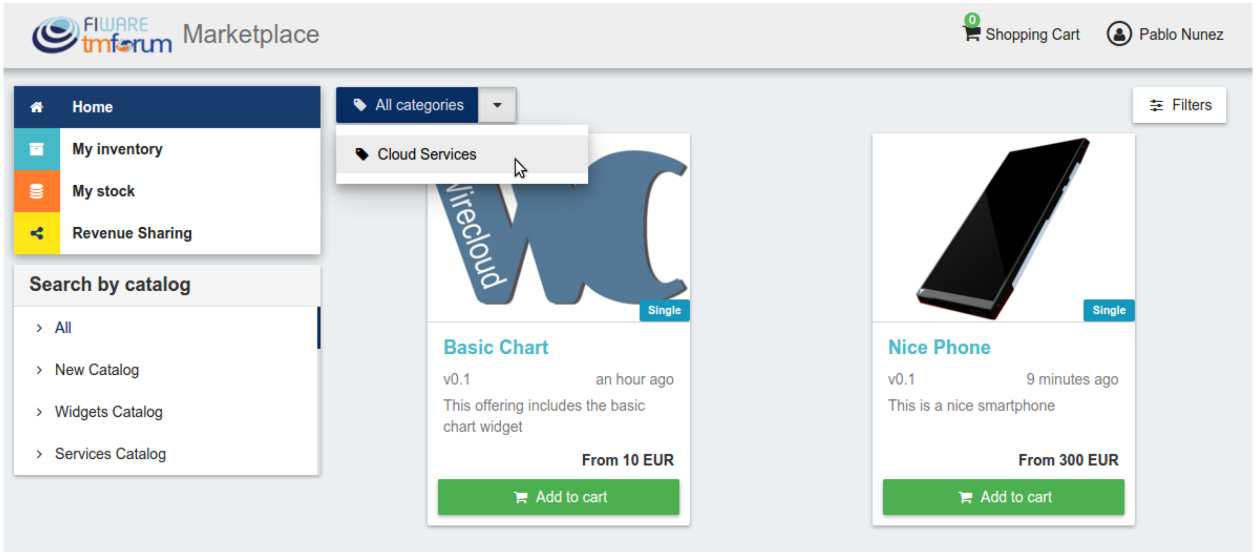


Additionally, customers can select an specific catalog of offerings by clicking on it.

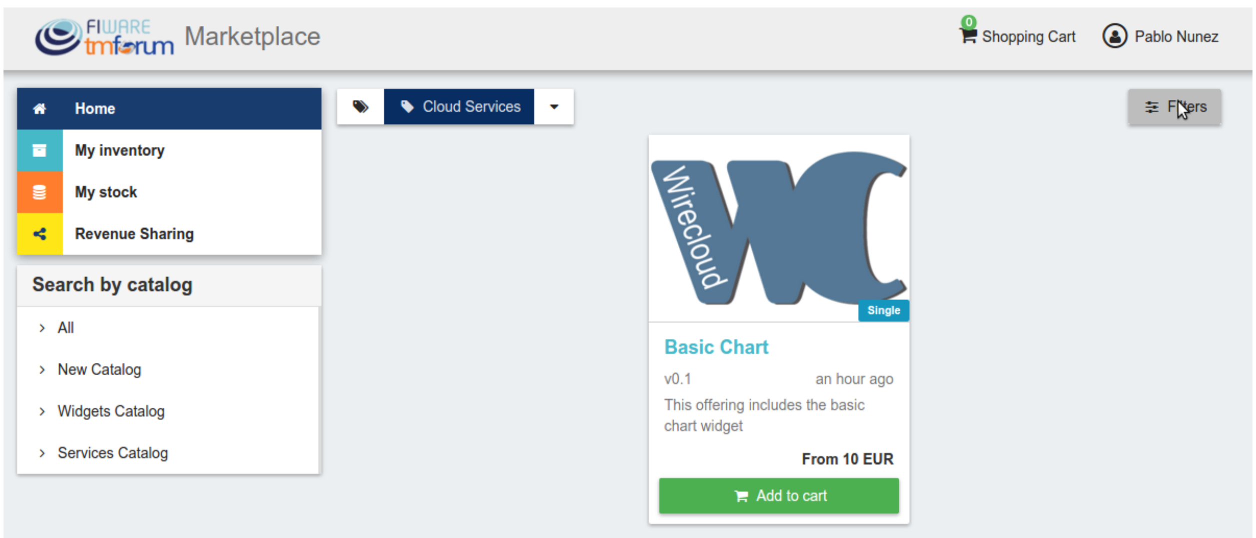


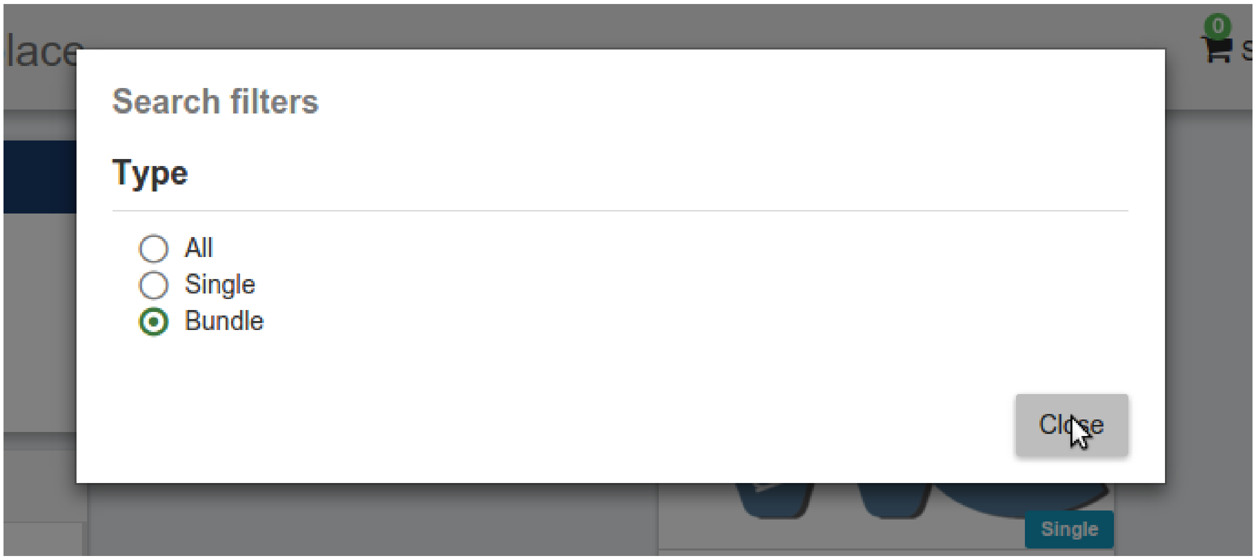


Moreover, customers can filter the shown offerings by category using the categories dropdown and choosing the wanted one.

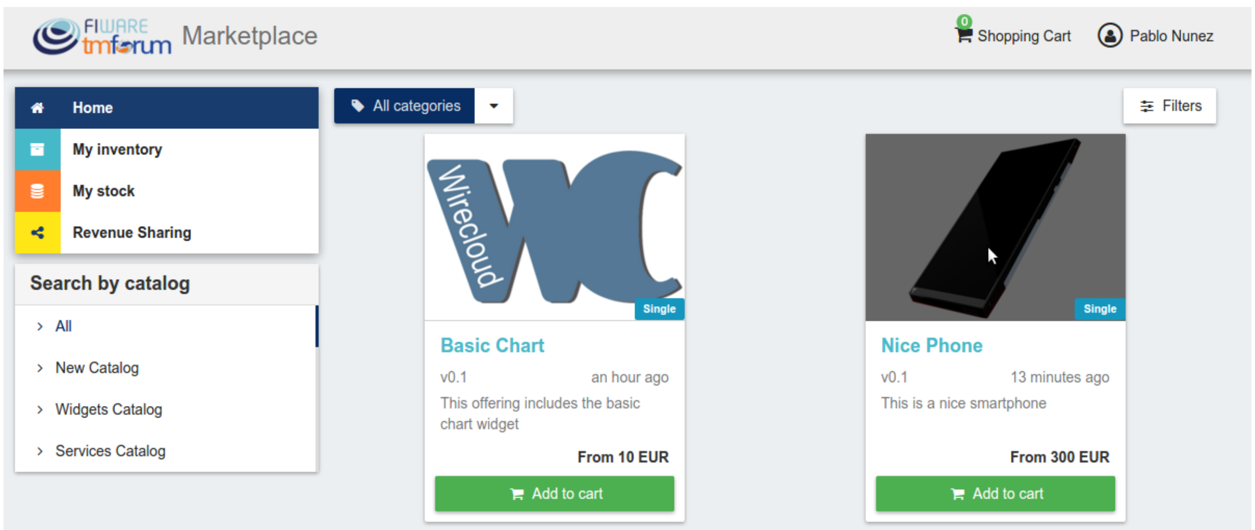


Finally, customers can also filter bundle or single offerings using the *Filters* modal.

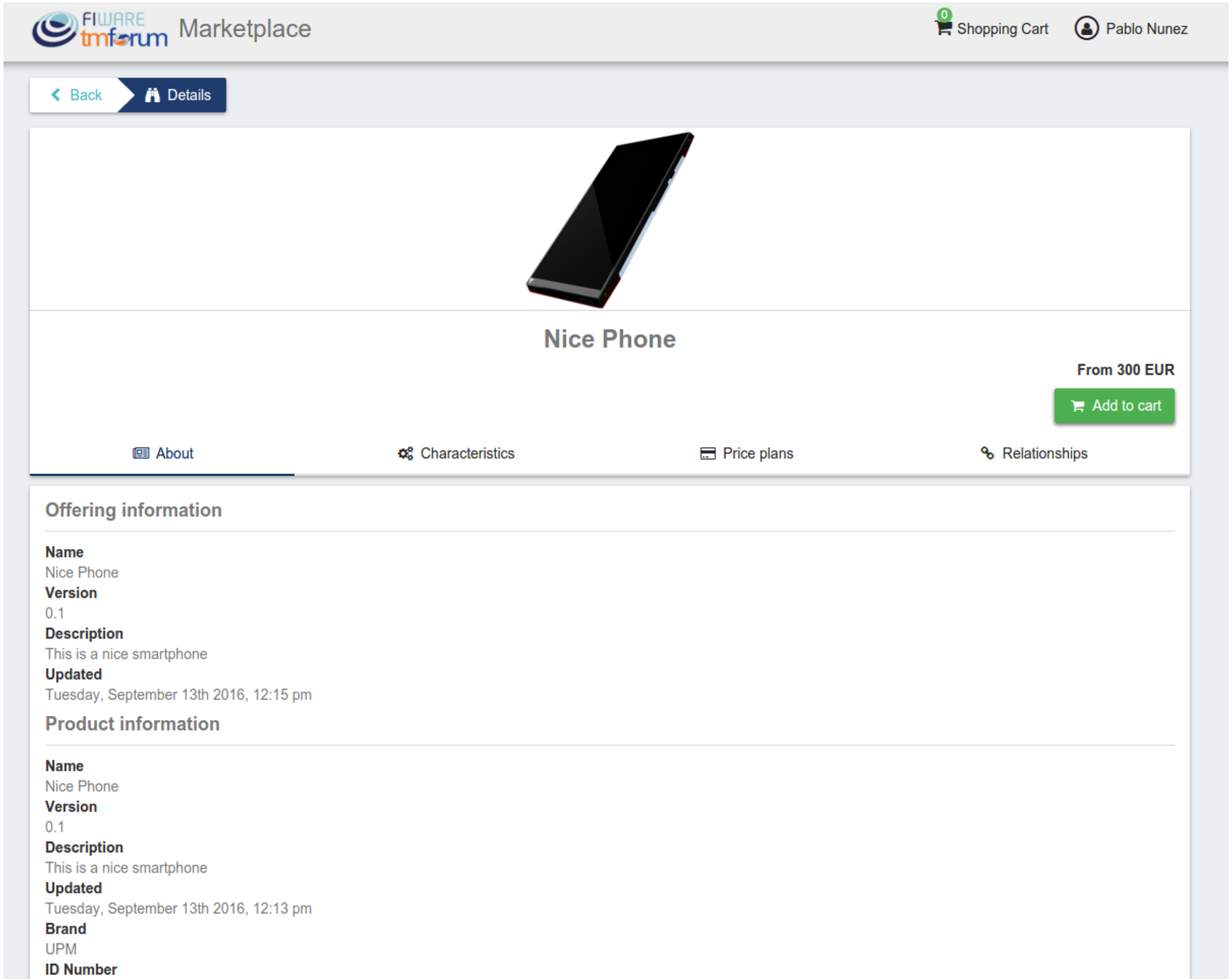


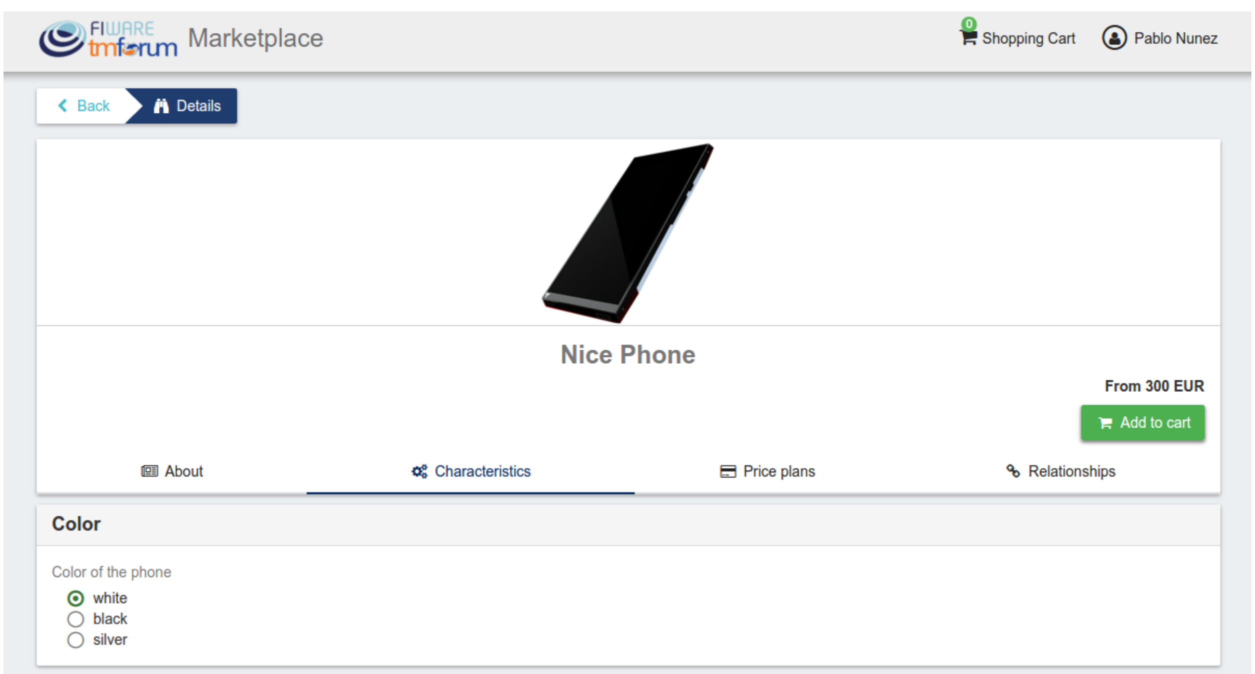


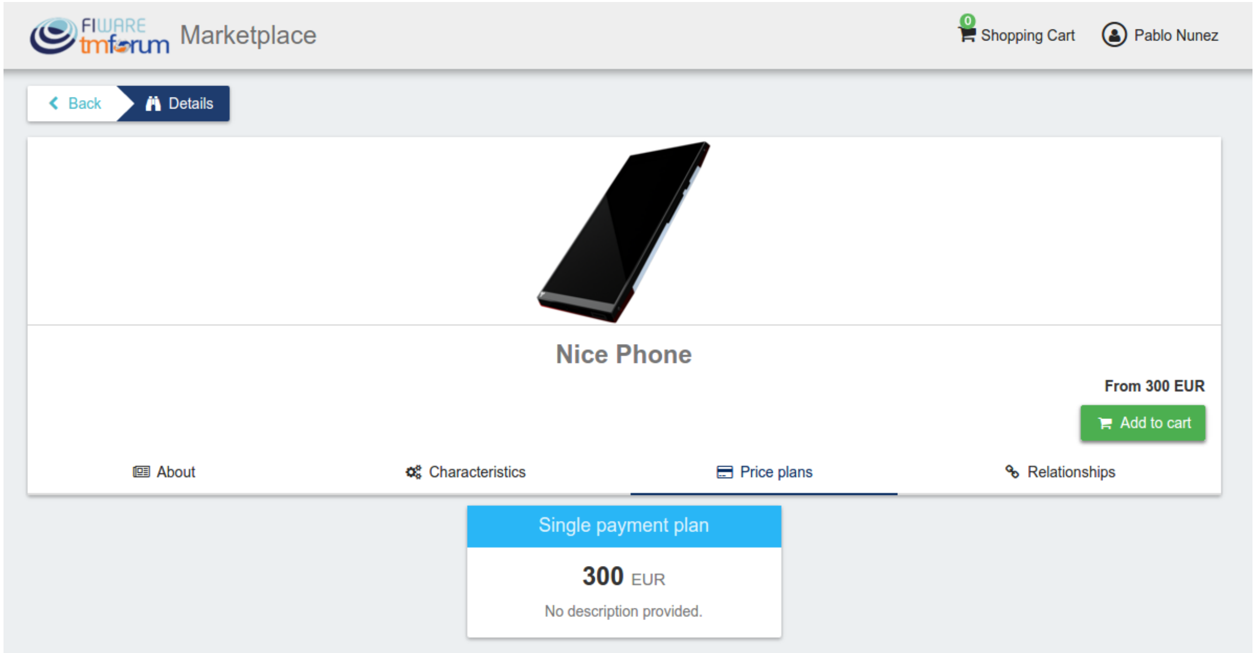
Customers can open the details of an offering by clicking on it



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


Marketplace
Shopping Cart
Pablo Nunez
Back
Details
Nice Phone
From 300 EUR
Add to cart
About
Characteristics
Price plans
Relationships
Offering information
Name
Nice Phone
Version
0.1
Description
This is a nice smartphone
Updated
Tuesday, September 13th 2016, 12:15 pm
Product information
Name
Nice Phone
Version
0.1
Description
This is a nice smartphone
Updated
Tuesday, September 13th 2016, 12:13 pm
Brand
UPM
ID Number

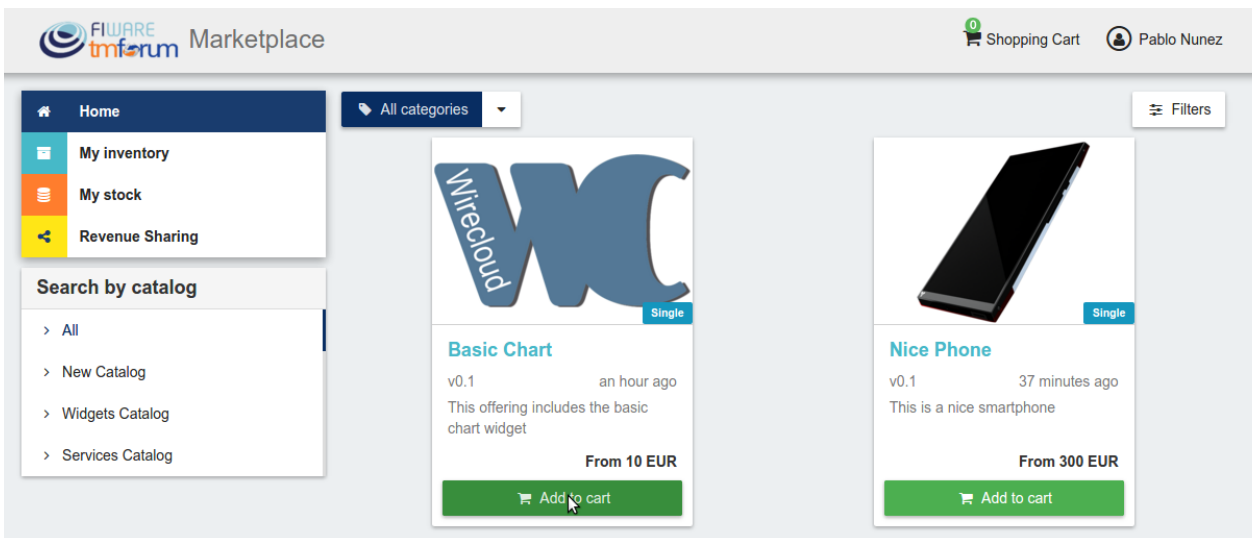

Marketplace
Shopping Cart
Pablo Nunez
Back
Details
Nice Phone
From 300 EUR
Add to cart
About
Characteristics
Price plans
Relationships
Color
Color of the phone
white
black
silver

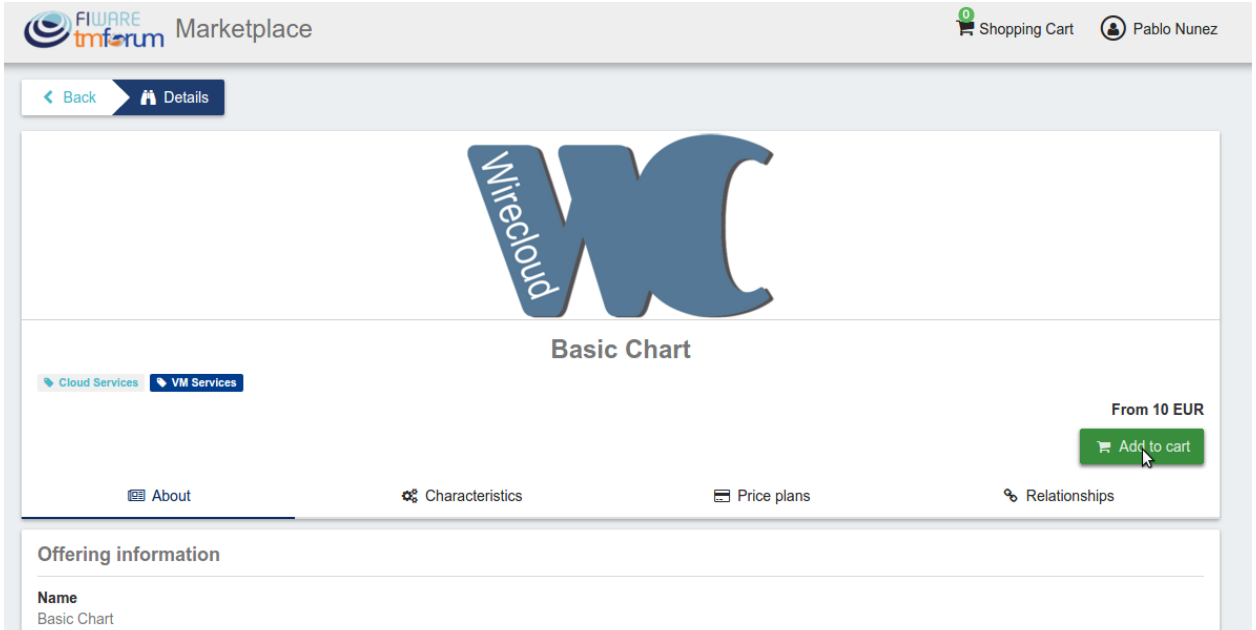


Create Order

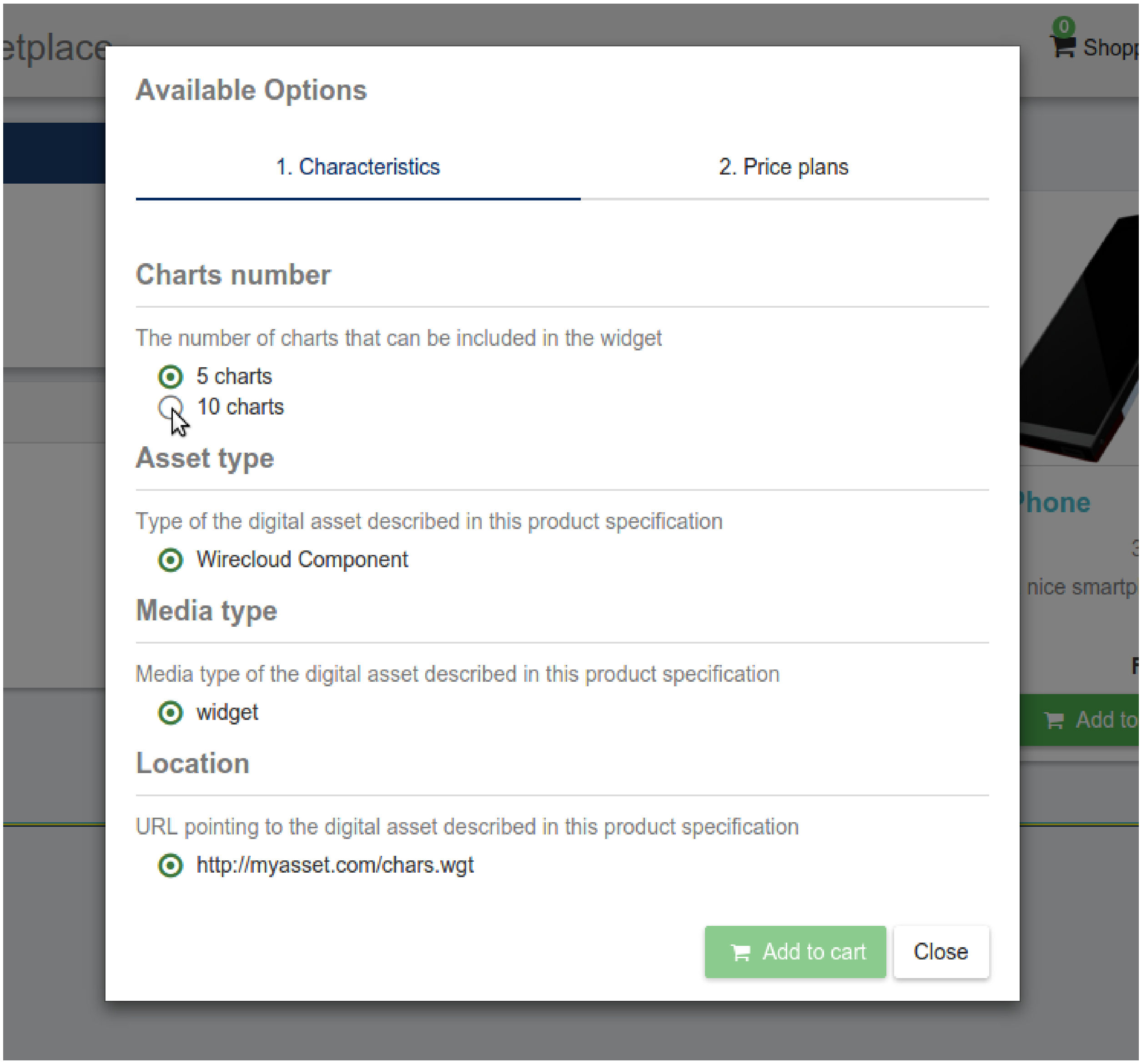
Customers can create orders for acquiring offerings. The different offerings to be included in an order are managed using the *Shopping Cart*.

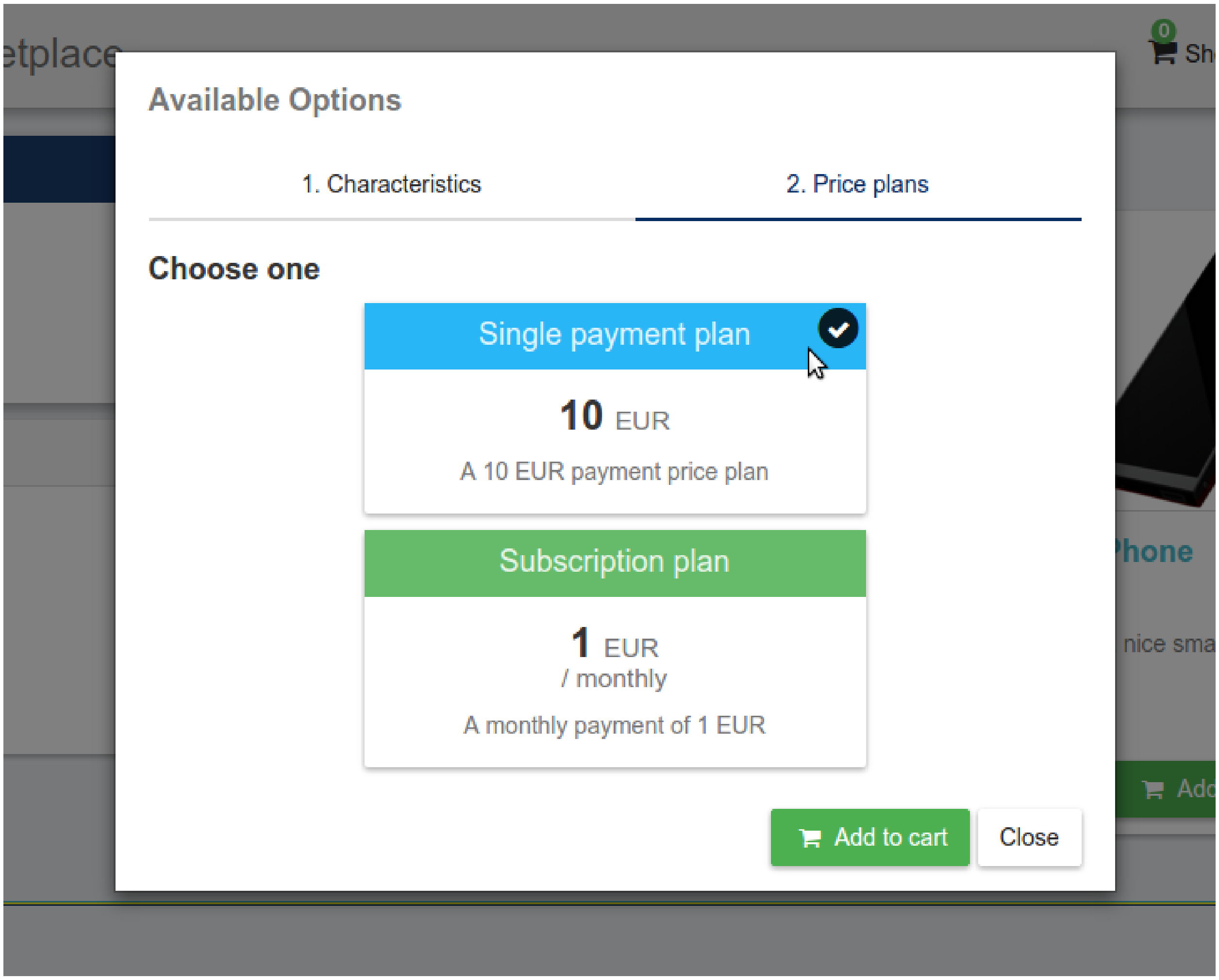
To include an offering in the shopping cart there are two possibilities. You can click on the *Add to Cart* button located in the offering panel when searching, or you can click on the *Add to Cart* button located in the offering details view.



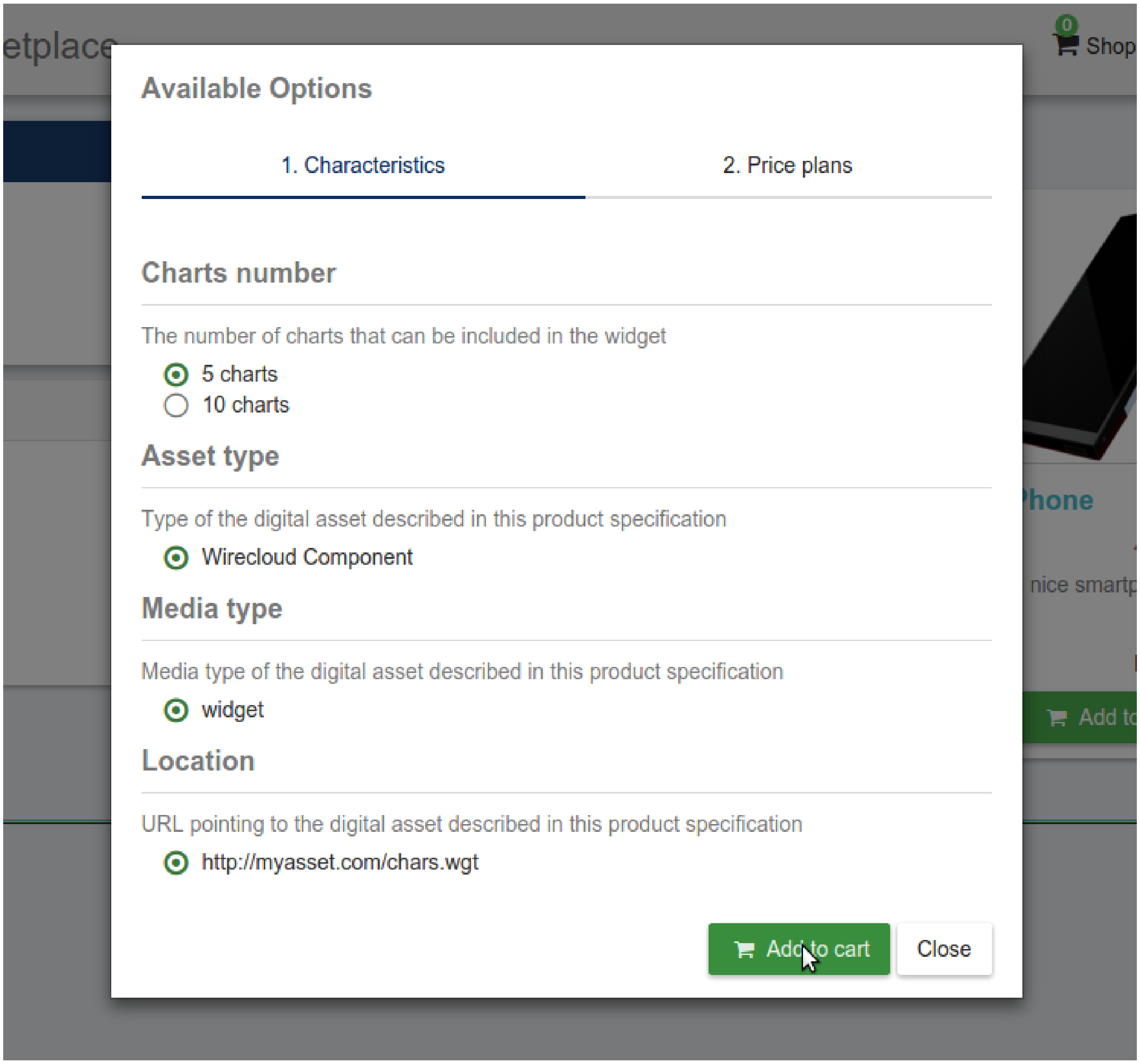


If the offering has configurable characteristics or multiple price plans, a modal will be displayed where you can select your preferred options

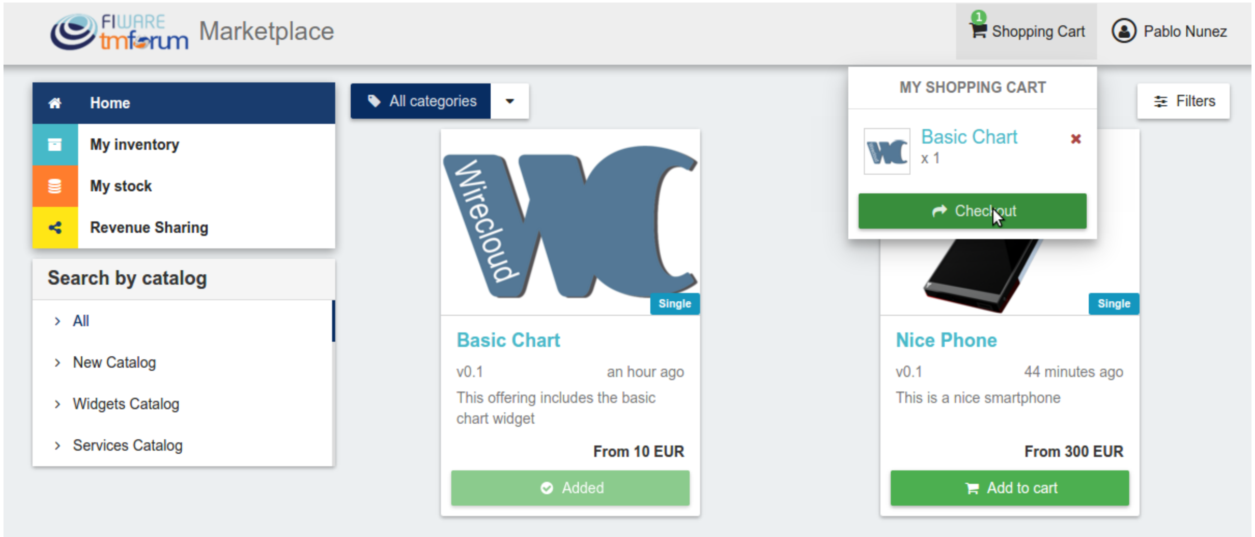

etplace
0
Shopp
Available Options
1. Characteristics
2. Price plans
Charts number
The number of charts that can be included in the widget
5 charts
10 charts
Asset type
Type of the digital asset described in this product specification
Wirecloud Component
Media type
Media type of the digital asset described in this product specification
widget
Location
URL pointing to the digital asset described in this product specification
http://myasset.com/chars.wgt
Add to cart
Close
hone
nice smartp
Add to



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



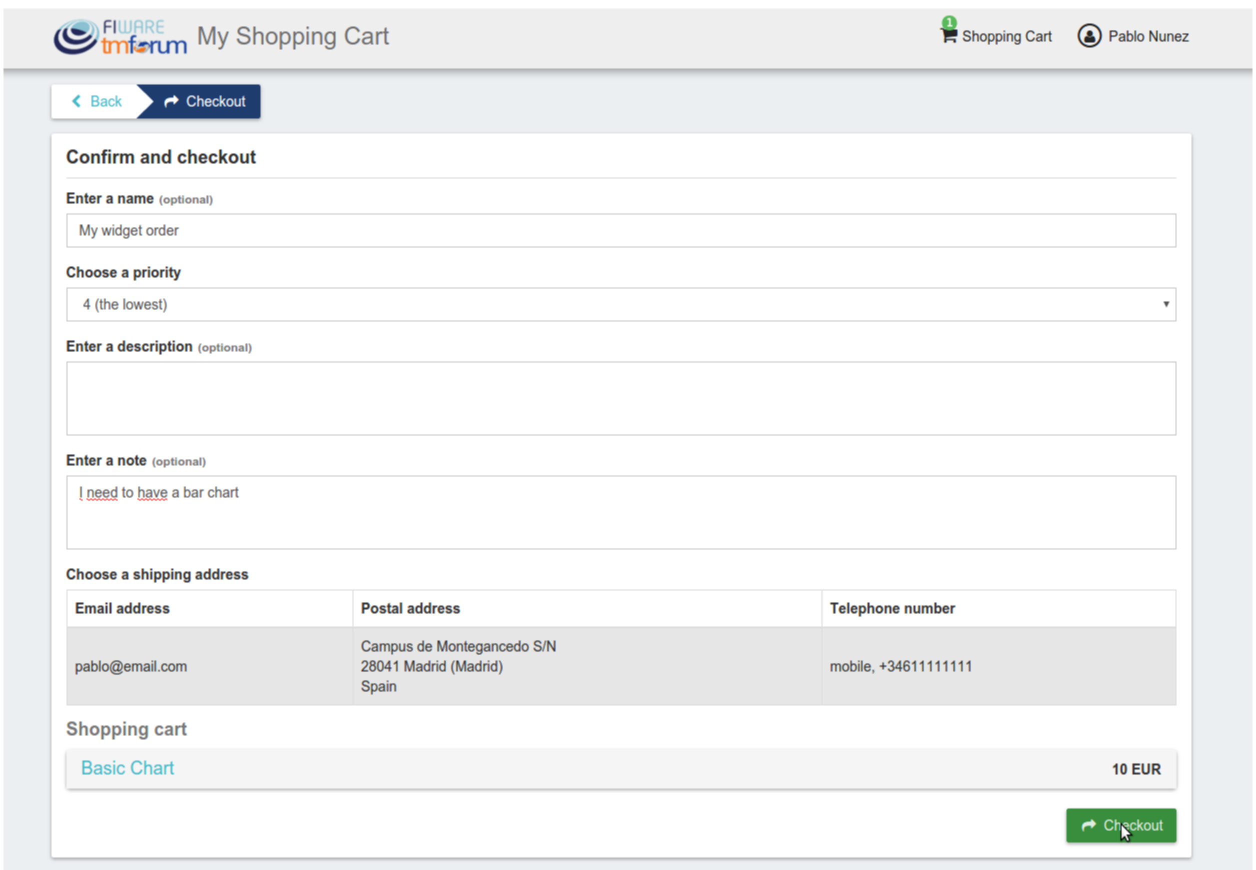
Once you have included all the offerings you want to acquire to the shopping cart, you can create the order clicking on *Shopping Cart*, and then on *Checkout*



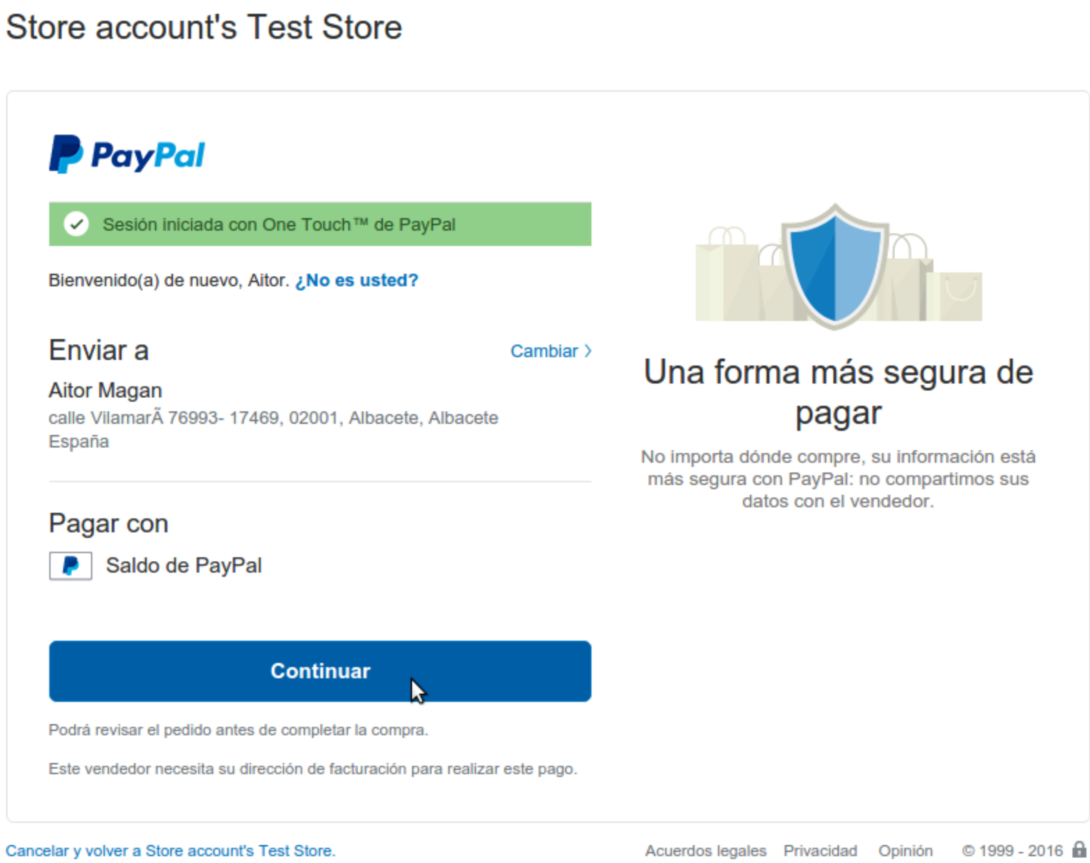
In the displayed form, you can include an optional name, an optional description, or an optional note. Notes can include any additional information you want to provide to the sellers of the acquired offerings.

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

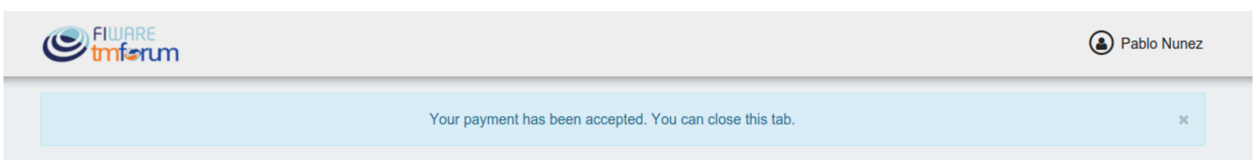
Once you have provided all the required information you can start the order creation clicking on *Checkout*



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

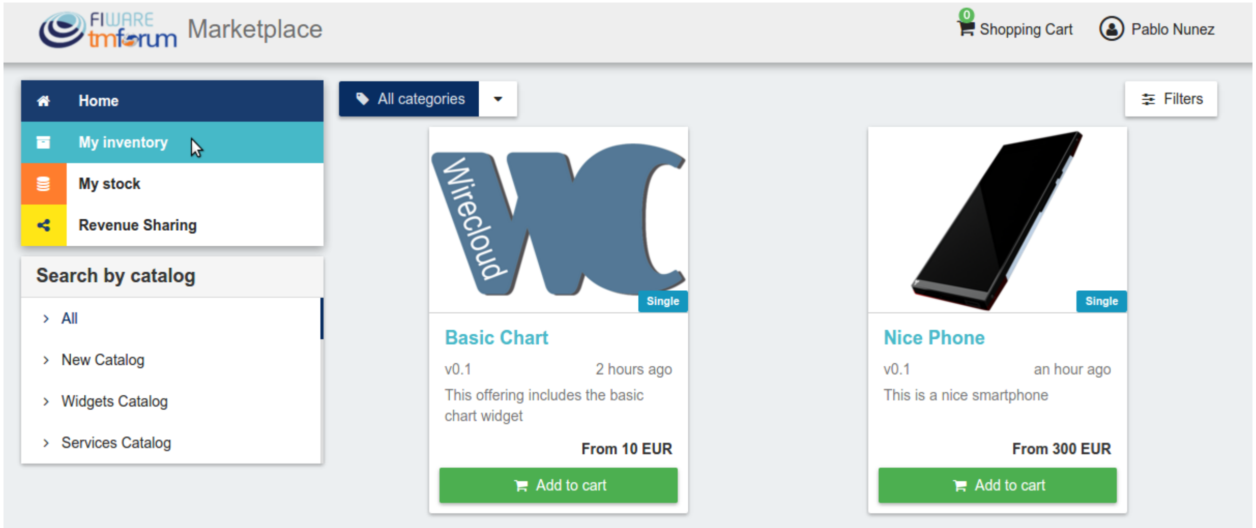


Finally, you will see a confirmation page

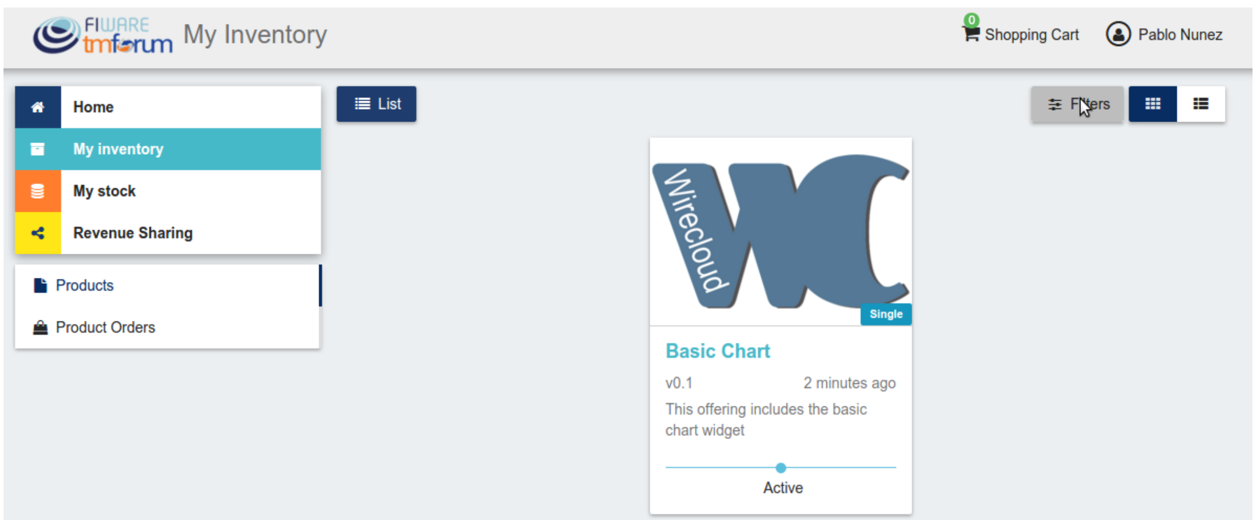


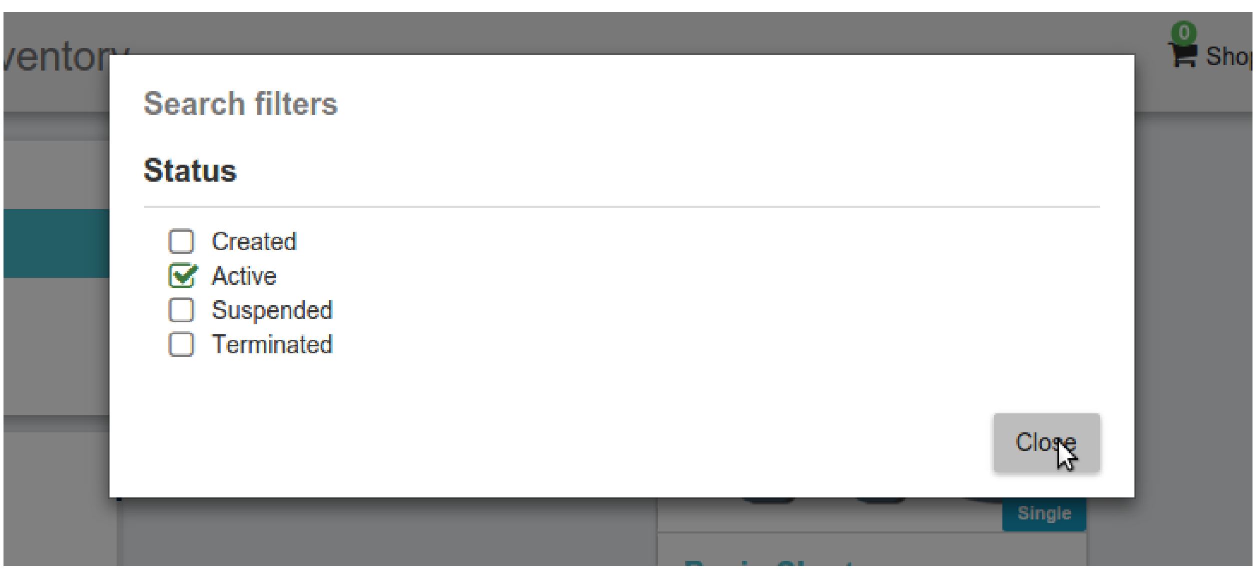
Manage Acquired Products

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

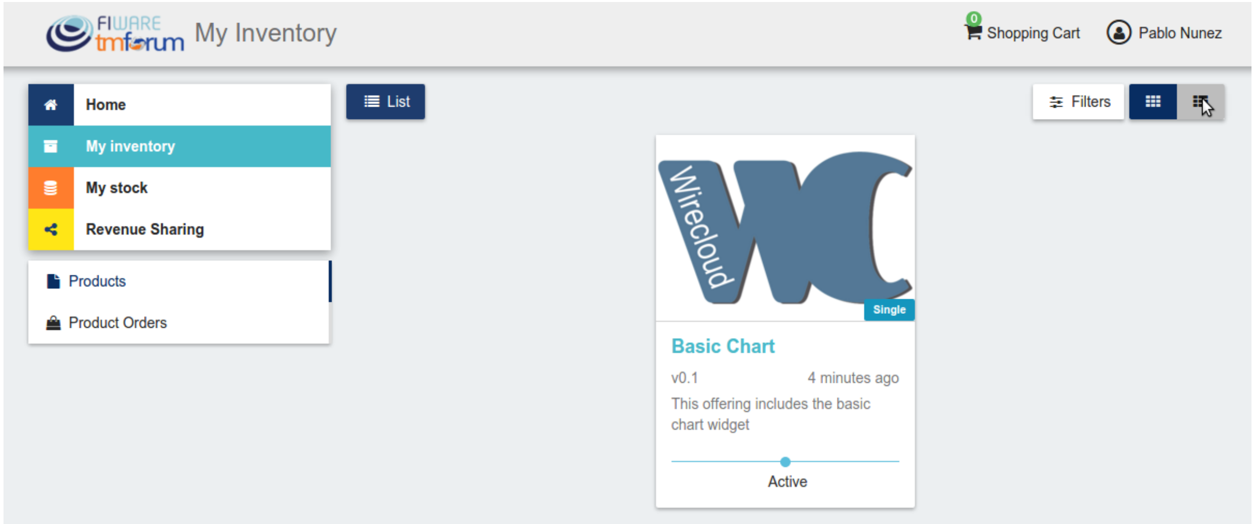


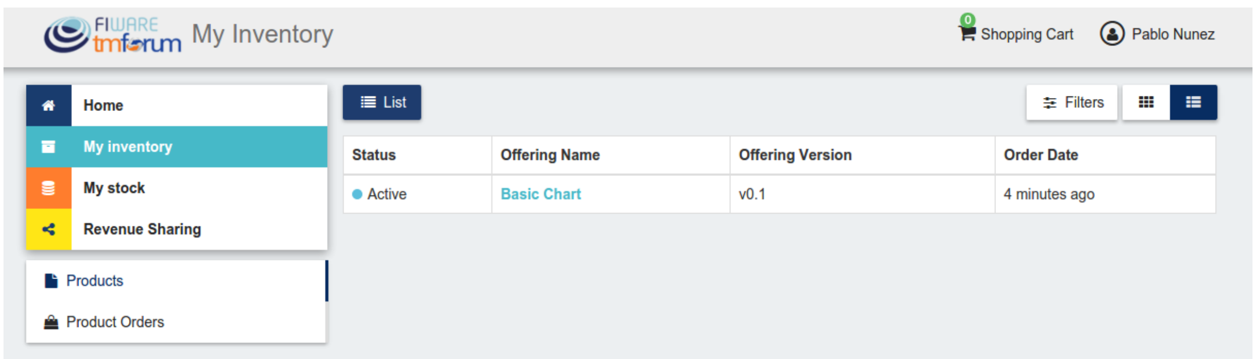
In this view, it is possible to filter you products by its status. To do that click on *Filters*, select the related statuses, and click on *Close*



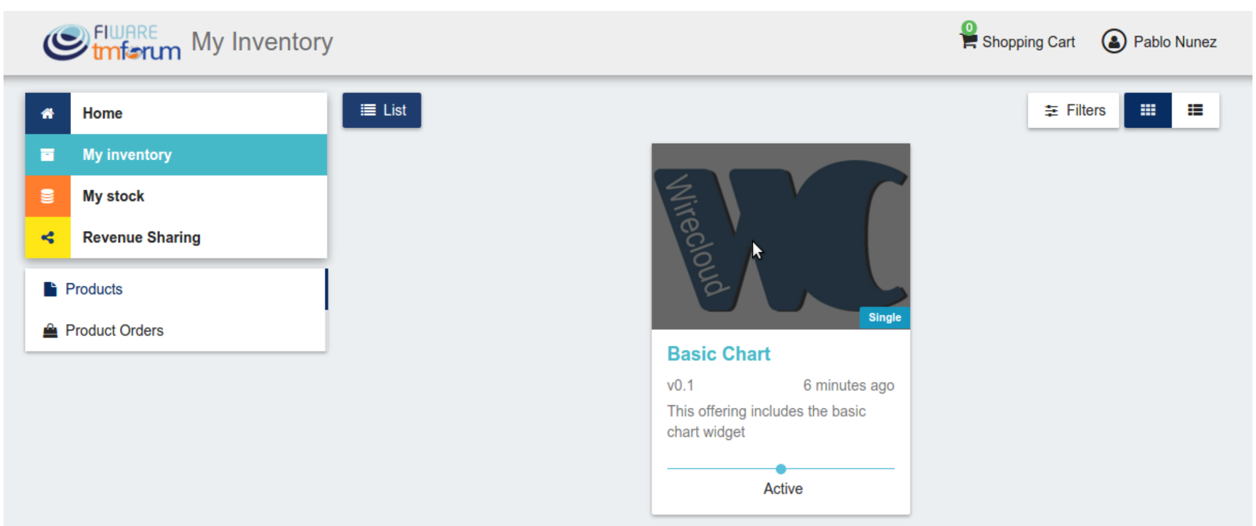


It is also possible to switch between the grid and tabular views using the related buttons

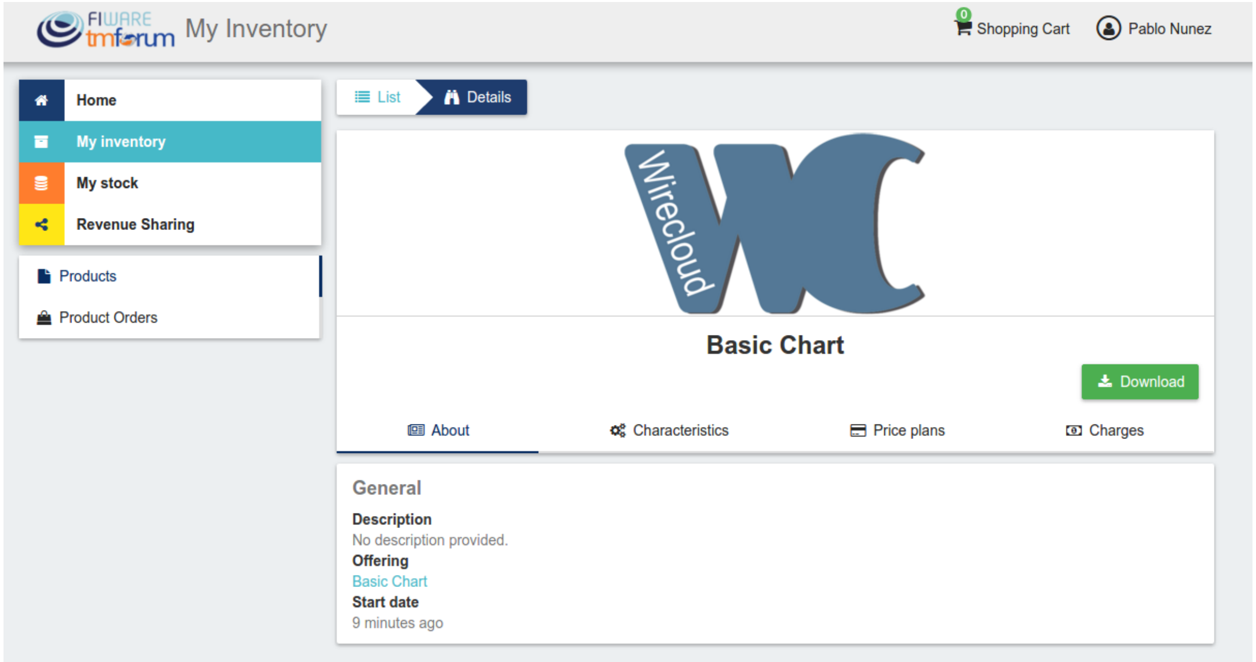


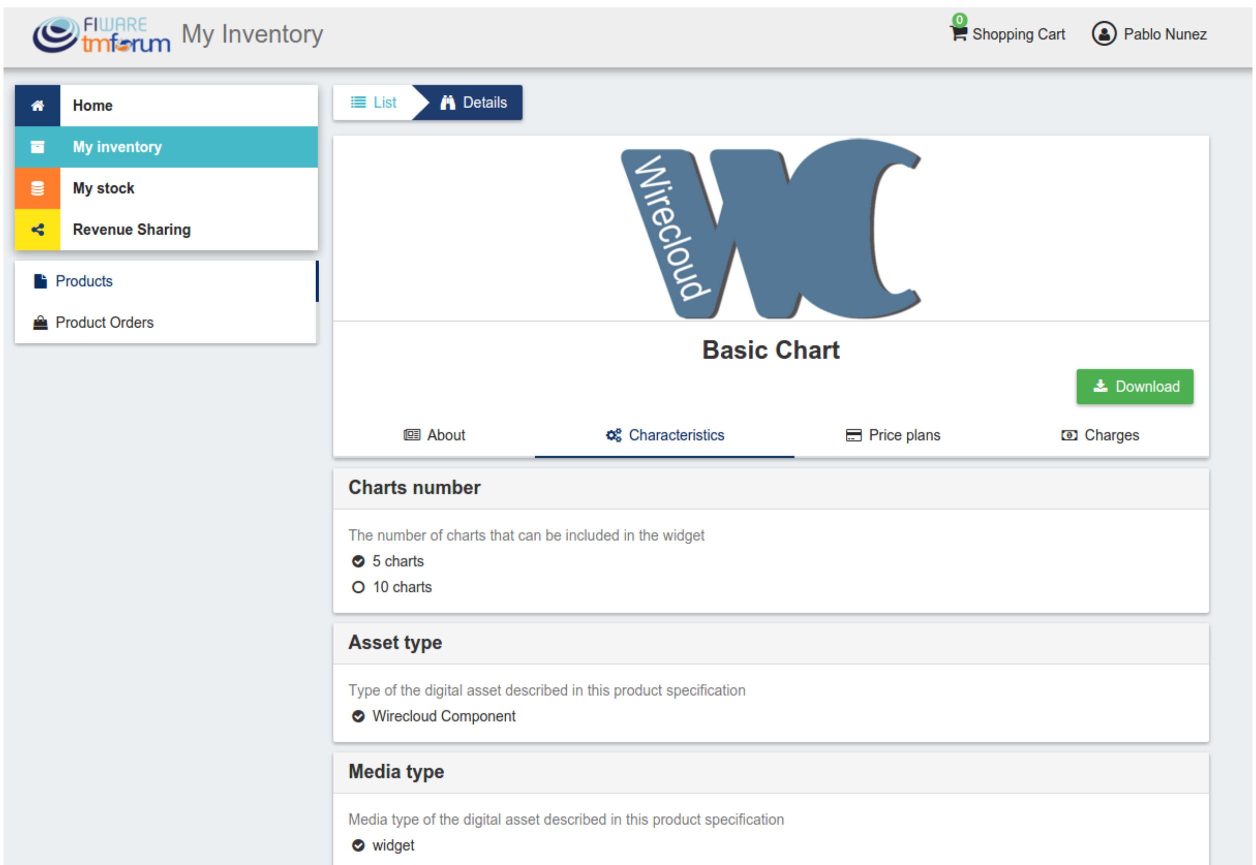


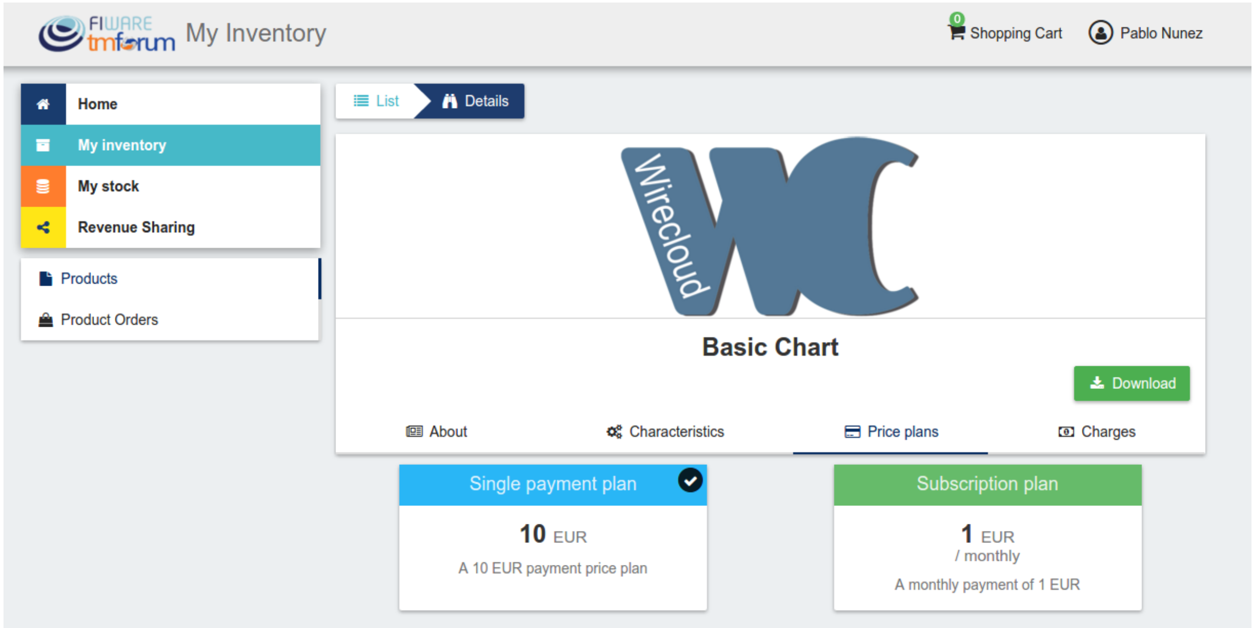
You can manage a specific acquired product clicking on it



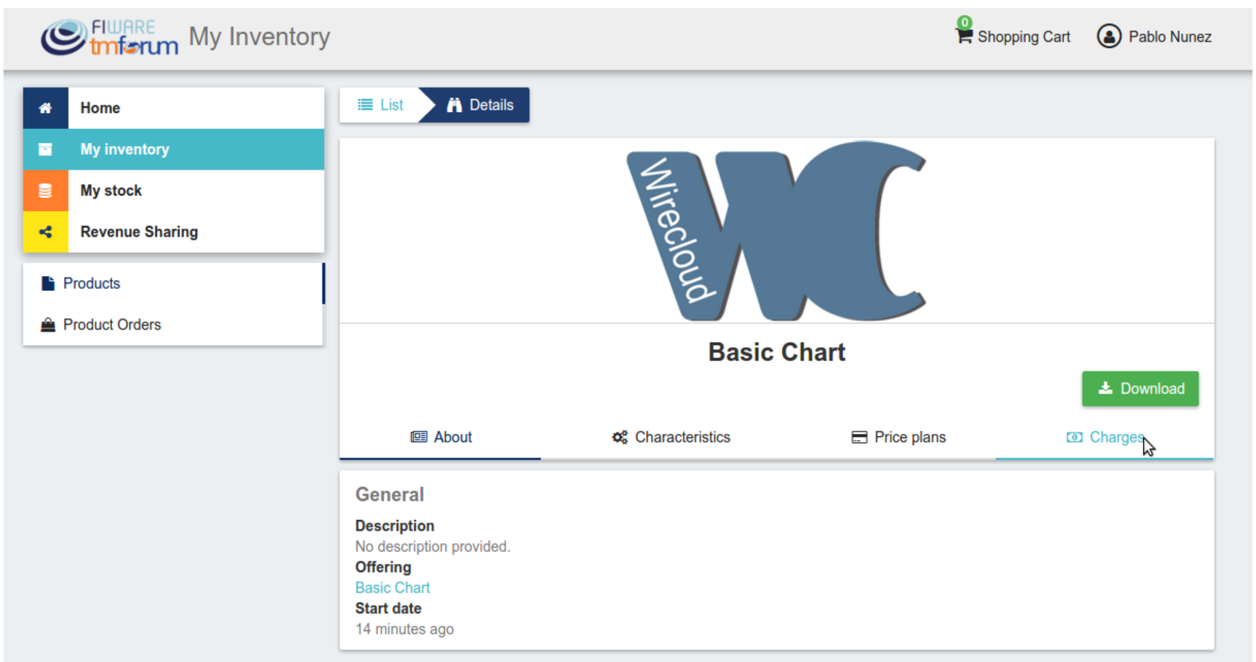
In the displayed view, you can see the general info of the acquired product, and the characteristics and pricing you have selected.


My Inventory
Shopping Cart
Pablo Nunez
Home
My inventory
My stock
Revenue Sharing
Products
Product Orders
List
Details
Basic Chart
Download
About
Characteristics
Price plans
Charges
General
Description
No description provided.
Offering
Basic Chart
Start date
9 minutes ago

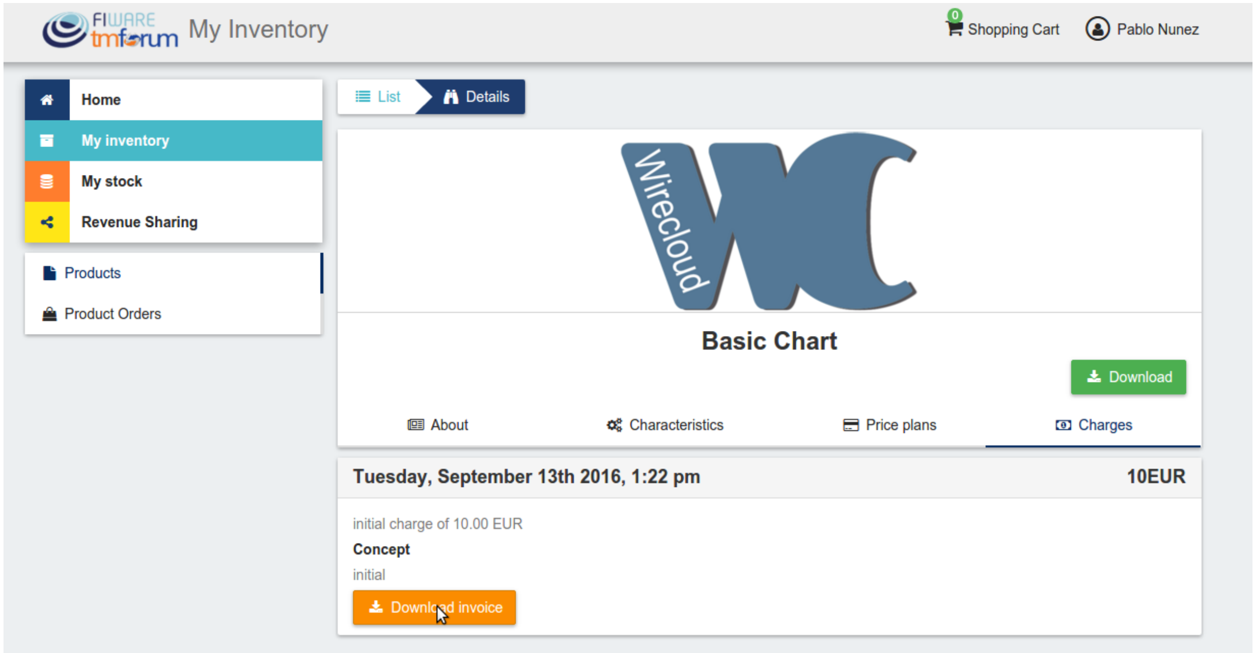

My Inventory
Shopping Cart
Pablo Nunez
Home
My inventory
My stock
Revenue Sharing
Products
Product Orders
List
Details
Basic Chart
Download
About
Characteristics
Price plans
Charges
Charts number
The number of charts that can be included in the widget
5 charts
10 charts
Asset type
Type of the digital asset described in this product specification
Wirecloud Component
Media type
Media type of the digital asset described in this product specification
widget



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



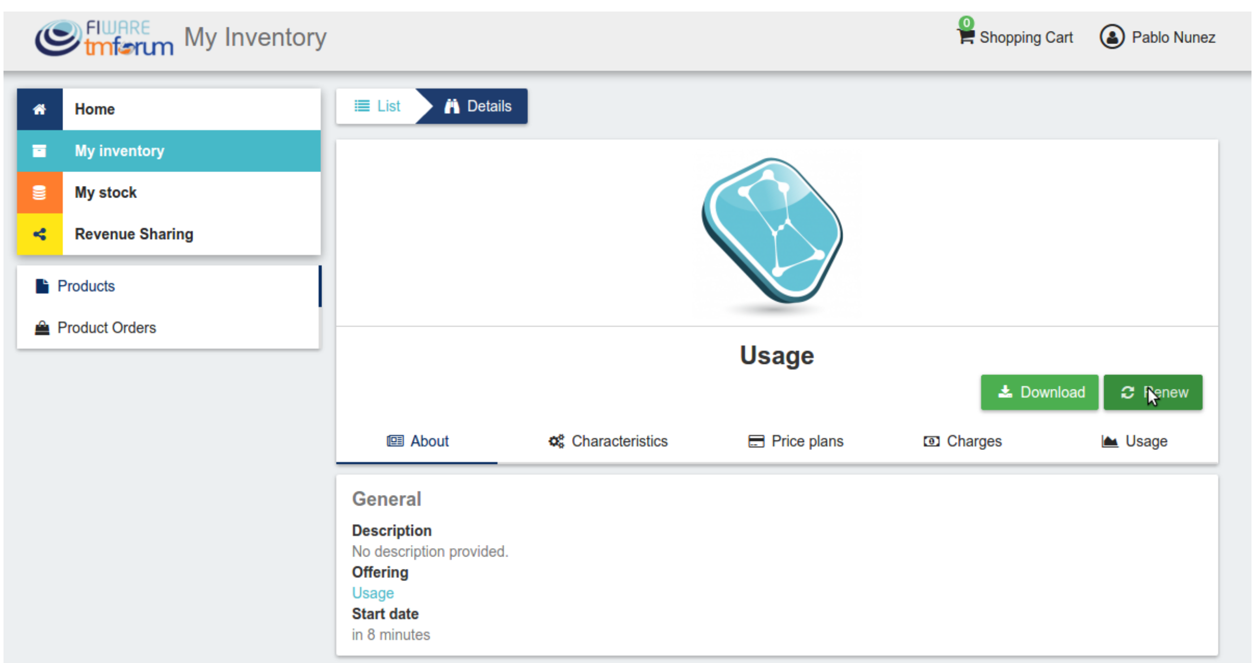
In this tab, you will find detailed information of the different charges and you will be able to download the related invoice clicking on *Download Invoice*



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

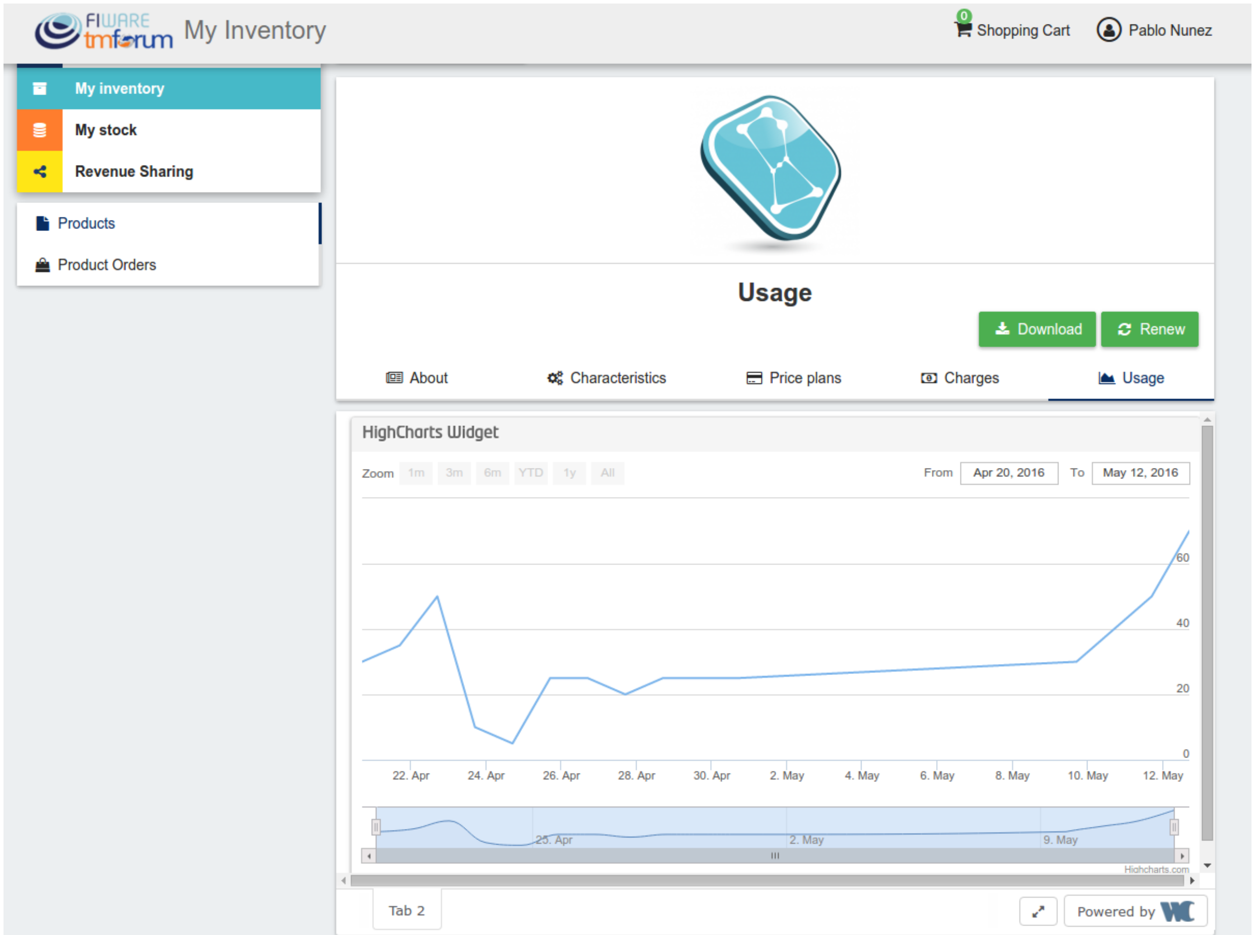
images/user/inv12.png

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.



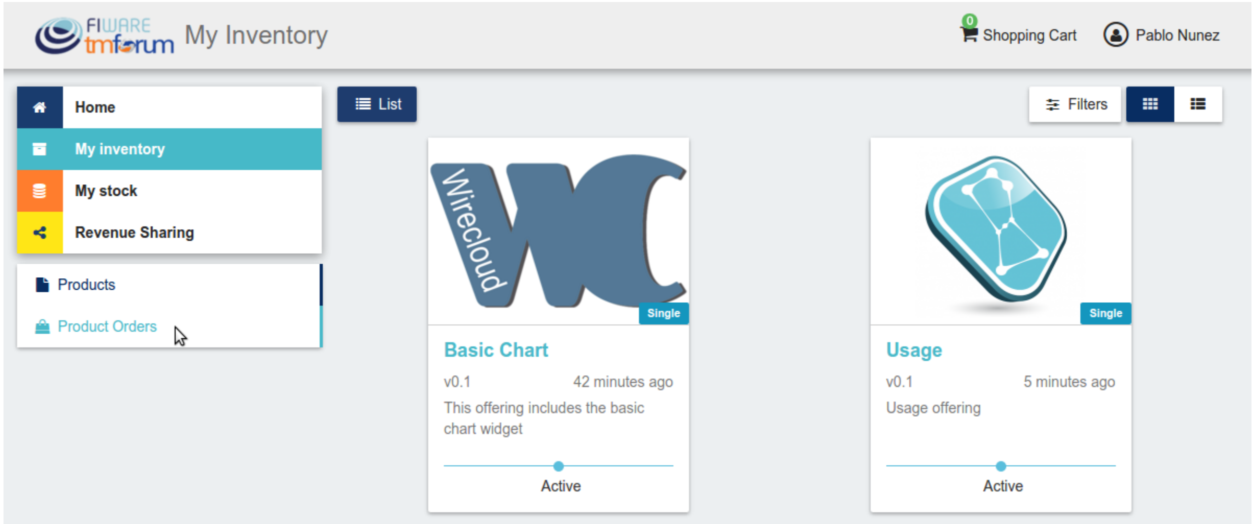
Note: If you product has expired and you do not renew it, it will be suspended, which means you will not have access to the acquired service until you pay

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

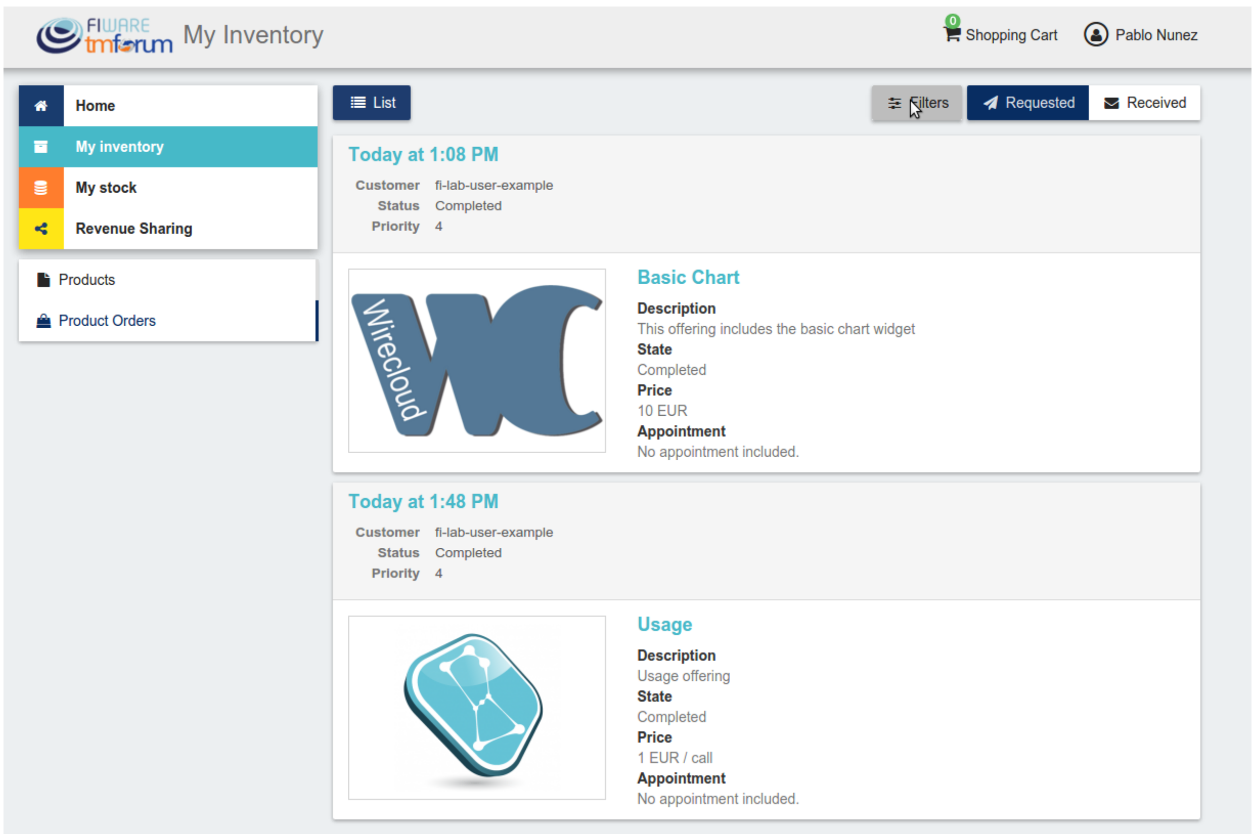


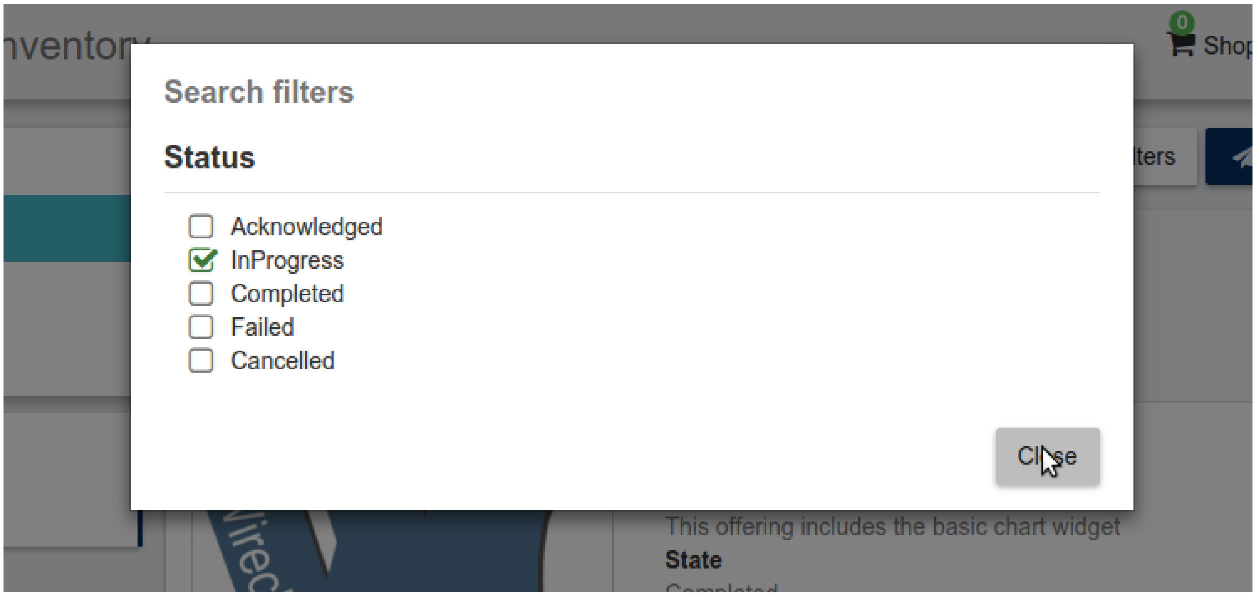
Manage Requested Orders

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*

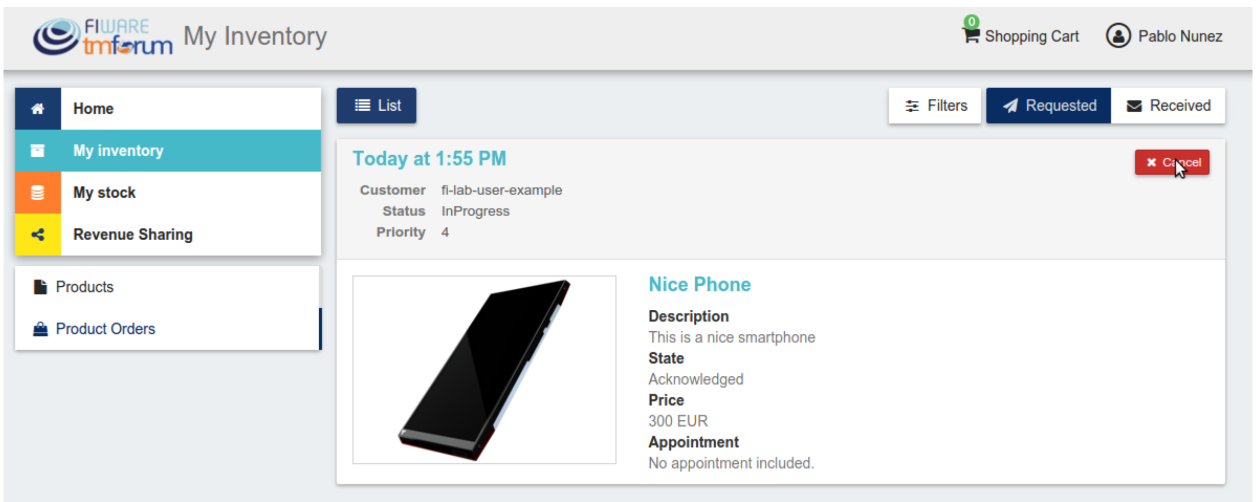


In the displayed view, you can see the orders you have created, which can be filtered by its status. To do that, click on *Filters*, select the wanted statuses, and click on *Close*

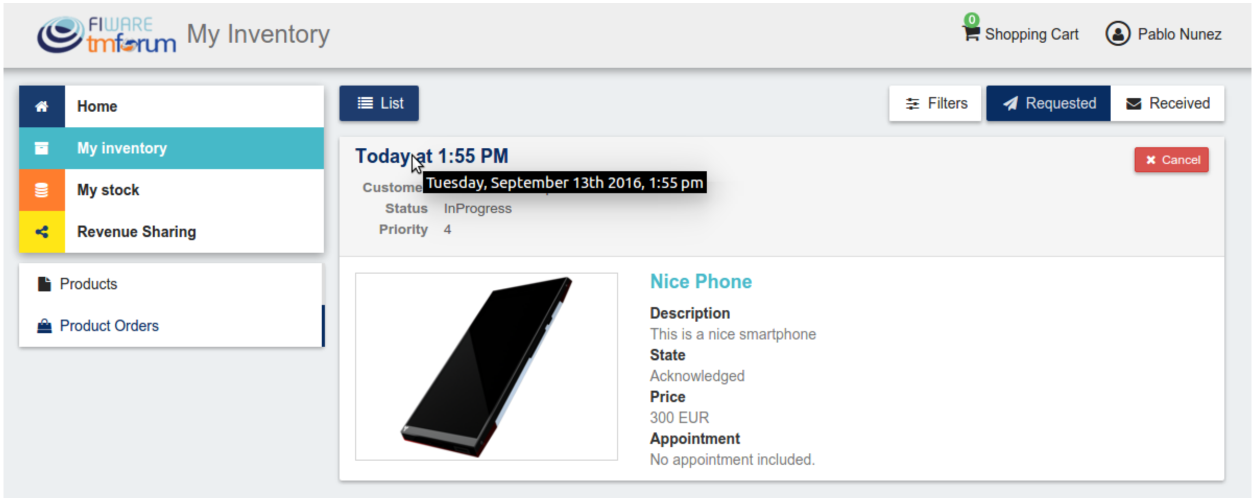




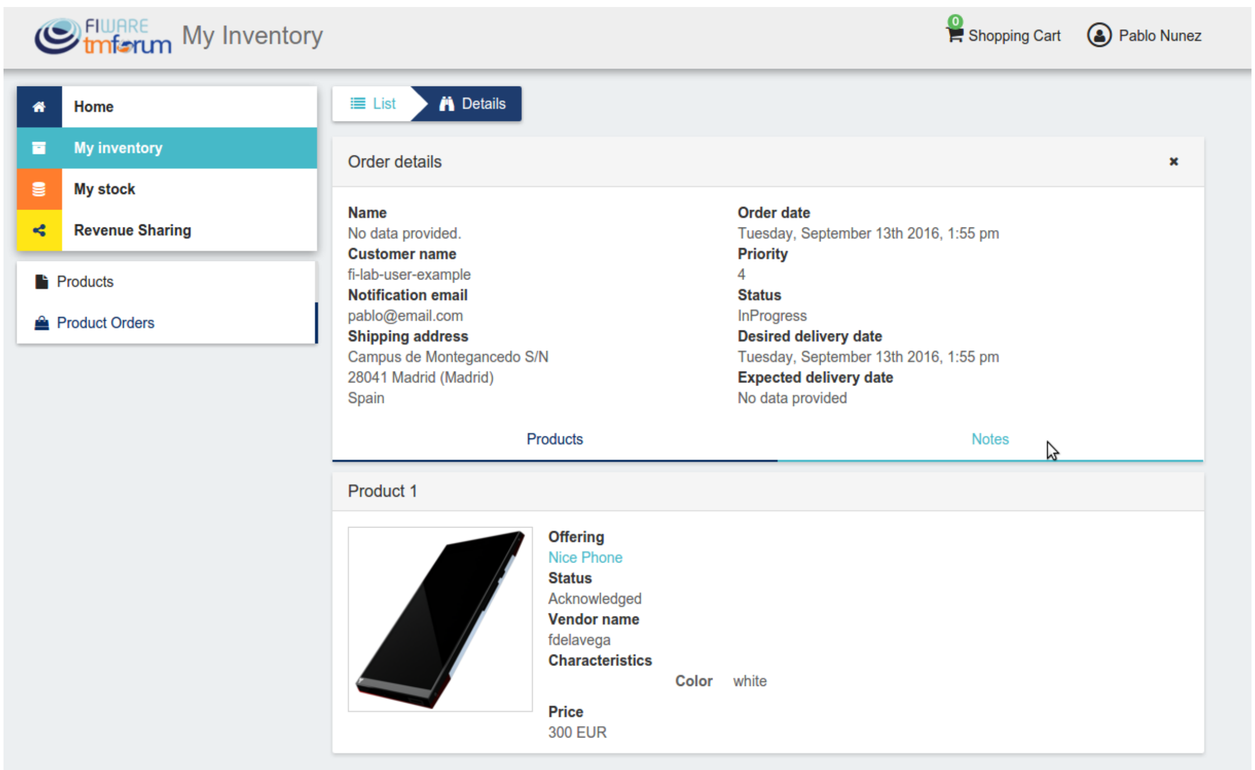
For those orders that include offerings of non digital products, you will be able to cancel them if the seller has not yet started the process. To do that, locate the order to be canceled and click on *Cancel*



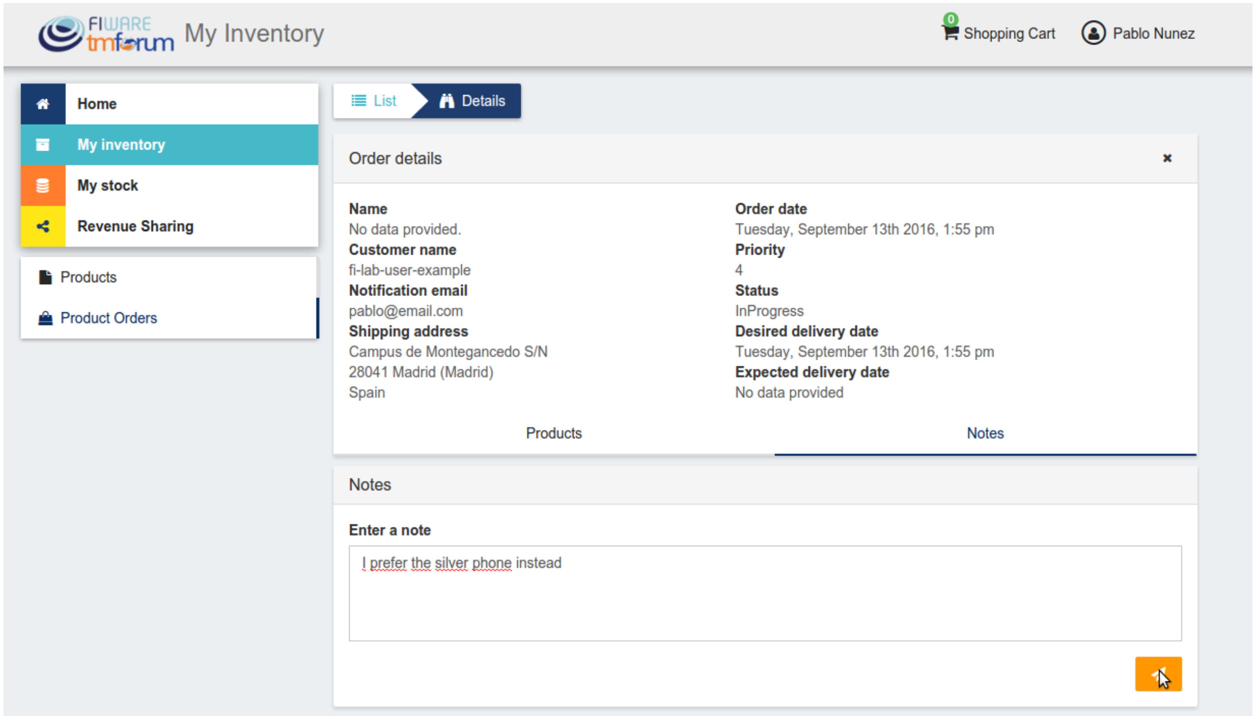
Moreover, you can review the details of the order. To do that click on the date of the order.



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



1.2.3 Programmer Guide

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

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

- Apiary
- GitHub Pages

Plugin Package

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

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

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

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

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

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

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

    def on_post_product_spec_validation(self, provider, asset):
        pass

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

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

    def on_pre_product_offering_validation(self, asset, product_offering):
        pass

    def on_post_product_offering_validation(self, asset, product_offering):
        pass

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

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

Implementing Event Handlers

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

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

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

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

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
```