TNOVA

NETWORK FUNCTIONS AS-A-SERVICE
OVER VIRTUALISED INFRASTRUCTURES

GRANT AGREEMENT NO.: 619520

Deliverable D6.3

Users Dashboard

Editor Evangelos K. Markakis (TEIC)

Contributors Evangelos Markakis, Athina Burdena, George Alexiou, Evangelos Pallis, Anargyros Sideris (TEIC), Aurora Ramos, Javier Melián (ATOS), Thomas Pliakas (CLDST),

Version 1.0

Date December 22nd, 2015

Executive Summary

The DoW describes this deliverable as: *"D6.3: User Dashboard (M24) – Report+Prototype – Design and implementation of the User Dashboard. Description of the implemented interfaces to the T-NOVA management modules. Presentation of the GUI and functionalities."*

The dashboard constitutes T-NOVA's system front-end. It enables the Service Providers (SPs) to create and publish their services, the Function Providers (FPs) to create and publish their Virtual Network Functions (VNFs) and both of them to participate in auctions for buying and selling, respectively, the published VNFs. In addition, the Dashboard allows the customers to discover (exploiting various criteria) and consume the offered services. Having in mind terminal neutrality and seamless upgradeability, whenever a new version is available, a web-based implementation of the Dashboard has been selected. Finally and towards maximising the users' Quality of Experience, T-NOVA's Dashboard allows personalization for a variety of settings such as interface, appearance and content according to each user's profile configuration.

In this context, this document presents an overview of the current technologies related to the development of a Dashboard. This includes the fronted (e.g. BootStrap), middleware (e.g. Angular) and backend (e.g. Django) web development frameworks. In the same direction, the existing web development and deployment approaches are surveyed focussing in the ones adopting a client-side and modular logic, for reasons of better scalability and interoperability—towards this, the notion of containerised micro-services is proposed and adopted for the Dashboard's implementation and deployment. In addition, the document describes the interfacing between the Users' Dashboard and the rest T-NOVA management modules, namely the Billing, UMAA (User Management, Authentication and Access Control), SLA management, Brokerage, Business Service Catalogue, Function Store and Orchestrator. Finally a step by step presentation of how to use the realised Dashboard is given in a series of screenshots.

Table of Contents

1. INTRODUCTION 8

1.1. DELIVERABLE'S STRUCTURE 10

2. USERS DASHBOARD DESIGN 11

2.1. AVAILABLE WEB DEVELOPMENT FRAMEWORKS 11
2.1.1. Frontend 11
2.1.2. Middleware 12
2.1.3. Back-End 13
2.1.4. Summary 15
2.2. DEVELOPMENT APPROACHES 15
2.2.1. Server-Centric Web Application (SCWA) 15
2.2.2. Browser-Centric Web Application (BCWA) 15
2.2.3. Summary 16
2.3. DEPLOYMENT ARCHITECTURES 16
2.3.1. Monolithic 16
2.3.2. Micro-services 17
2.3.3. Containerized micro-services 18
2.3.4. Summary 19

3. USERS DASHBOARD IMPLEMENTATION 20

3.1. ROUTING MODULE 20
3.2. BOOTSTRAP AUTHENTICATION FUNCTION 20
3.3. ROOT CONTROLLER 21
3.4. LOGIN CONTROLLER 22
3.5. REGISTER CONTROLLER 22
3.6. USER MANAGEMENT CONTROLLER 23
3.7. VNF MANAGEMENT CONTROLLER 24
3.8. NFSTORE IMAGES MANAGEMENT CONTROLLER 24
3.9. SERVICES MANAGEMENT CONTROLLER 25
3.10. BROKER CONTROLLER 25

4. USERS DASHBOARD INTERFACING WITH REST T-NOVA MARKETPLACE MODULES 26

4.1. APIS DEFINITION 26
4.1.1. Billing 26
4.1.2. User Management, Authentication and Access control Module 31
4.1.3. SLA Management 39
4.1.4. Billing 45
4.1.5. Brokerage 49
4.1.6. Function Store 51
4.1.7. Business Service Catalogue 54
4.1.8. Service Selection 58
4.1.9. Orchestrator 59

5. GRAPHICAL USER INTERFACE AND FUNCTIONALITIES 61

5.1. USER REGISTRATION 61
5.2. USER LOGIN 61
5.3. VNF PROVIDER 62
5.3.1. VNF Creation *62*
5.3.2. VNF Listing *66*
5.3.3. VNF Tools *67*
5.3.4. Images *69*
5.4. SERVICE PROVIDER 69
5.4.1. Service/NSD Creation *70*
5.4.2. NSD Listing *74*
5.4.3. NSD Tools *74*
5.5. CUSTOMER 75
5.5.1. Customer – Service Selection *75*
5.5.2. Customer - Service Purchase *76*
5.5.3. Customer Services *76*
5.6. ADMINISTRATOR 77
5.6.1. User Management *77*
6. VALIDATION 80
6.1. FUNCTIONAL VERIFICATION 80
6.2. REQUIREMENTS FULFILLMENT 82
7. CONCLUSIONS 84
8. REFERENCES 85
9. ANNEX A: 86

Index of Figures

Figure 1-1 Dashboard views 8
Figure 2-1 Single Page Application 16
Figure 2-2 Dashboard in monolithic deployment 17
Figure 2-3 Dashboard deployed as a micro-service 18
Figure 2-4 Dashboard deployed as a containerised micro-service 19
Figure 3-1 Routing module 20
Figure 3-2 Bootstrap authentication function 21
Figure 3-3 Root controller 21
Figure 3-4 Login controller 22
Figure 3-5 Register controller 23
Figure 3-6 User management controller 23
Figure 1-1 VNF management controller 24
Figure 1-1 NFStore Images Management controller 24
Figure 1-1 Service Management controller 25
Figure 1-1 Broker controller 25
Figure 1-1 User registration 61
Figure 1-1 Dashboard's login page 62
Figure 1-1 VNF Providers home page 62
Figure 1-1 VNF Creation - Step 1: VNF Basic Information 63
Figure 1-1 VNF Creation - Step 2: VNF Composition, SLA Flavors 64
Figure 1-1 VNF Creation - Step 2: VNF Composition, Virtual Machines 64
Figure 1-1 VNF Creation - Step 2: VNF Composition, Virtual Links 65
Figure 1-1 VNF Creation - Step 3: SLA 65
Figure 1-1 VNF Creation - Step 3: SLA, Assurance Parameters 66
Figure 1-1 VNF Creation - Step 4: Billing 66
Figure 1-1 VNF Listing 67
Figure 1-1 Genereted VNFD Viewer 67
Figure 1-1 VNFD YAML Editor 68
Figure 1-1 VNF Diagram 68
Figure 1-1 Function Provider VNF Images 69
Figure 1-1 Service Provider 69
Figure 1-1 NSD Creation - Step 1: VNF Selection 70
Figure 1-1 NSD Creation - Step 1: VNF Selection, Trade Request 71
Figure 1-1 NSD Creation - Step 1: VNF Selection, Pending Trade Request 71
Figure 1-1 Trade Request FP View 71
Figure 1-1 Accepted Trade Offer 71
Figure 1-1 NSD Creation - Step 2: Basic Information 72
Figure 1-1 NSD Creation - Step 3: Service Composition and SLA 73
Figure 1-1 NSD Creation - Step 4: Assurance Parameters 73
Figure 1-1 NSD Creation - Step 4: Assurance Parameters 74
Figure 1-1 NSD Listing 74
Figure 1-1 Generated NSD Viewer 75
Figure 1-1 NSD YAML Editor 75
Figure 1-1 Customer – Service Selection 76
Figure 1-1 Customer - Service Purchase 76

Figure 1-1 Customer Services 77
Figure 1-1 Administrator 77
Figure 1-1 User Management 78
Figure 1-1 User Creation 78
Figure 1-1 Edit User Profile 79

Index of Tables

Table 1 SP dashboard view 9
Table 2 FP dashboard view 9
Table 3 Customer Dashboard view 9
Table 6-1 Dashboard functional verification 82
Table 6-2 Dashboard basic requirements 83

1. INTRODUCTION

As already stated at D2.42, the Dashboard constitutes the T-NOVA system's front-end and hosts the three views for the three basic stakeholders that will access the T-NOVA Marketplace: the Service Provider (SP), the Function Provider (FP) and the Customer. The main features of the Dashboard are presented in Figure 1-1.

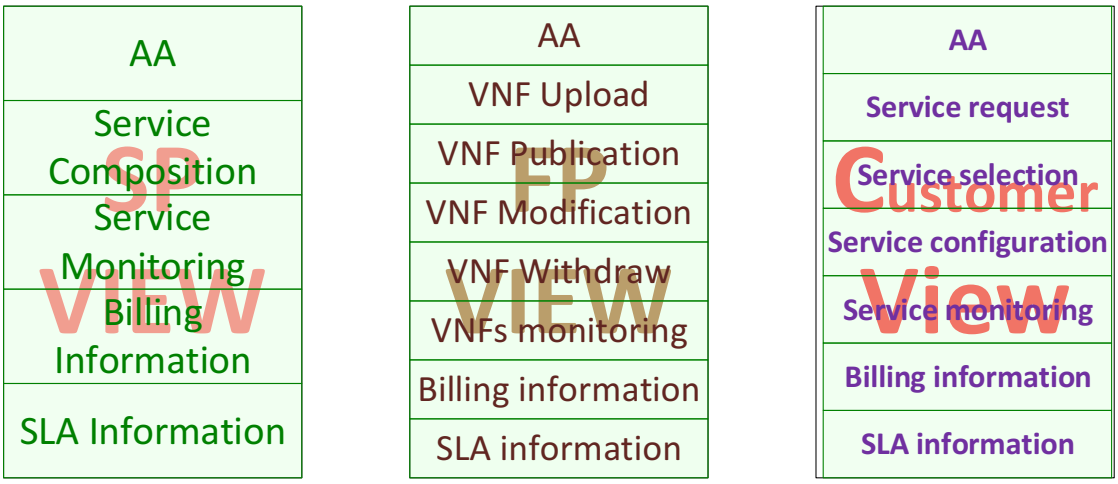

Figure 1-1 Dashboard views

In summary, the SPs', FPs' and Customers' views of the dashboard will allow them to provide the functionalities shown in Table 1, Table 2 and Table 3 respectively.

Functionality	Short Explanation
AA	Authorization and Authentication of the respective role into the T-NOVA Dashboard.
Service composition	Graphical wizard that will help the SP to compose a new Network Service (NS) starting from the brokerage among the FPs owing the available VNFs.
Service monitoring	Graphical representation of all monitoring data for a selected or "consumed" Service.
Billing information	Graphical representation of the billing outcomes of selected or "consumed" service. There will be two types of billing information for the SP: - Charges for the SP's customers (BSS functionality). - Invoices on behalf of its own suppliers, the FPs.
SLA information	Details of the selected or "consumed" service based on how they respect the agreed SLA. The SP will have access to two different kinds of SLA contract and SLA monitoring information: - SLA between SP and its customers (BSS) - SLA agreed with his its suppliers, the FPs

Table 1 SP dashboard view

Functionality	Short Explanation
AA	Authorization and Authentication of the respective role into the T-NOVA dashboard.
VNF Upload	Graphical wizard that will help the FP to upload his VNF with the necessary parameters.
VNF Publication	Graphical representation for the FP to provide the last check in order to publish the uploaded VNF
VNF Modification	Small graphical wizard that provides the ability to the FP to modify the uploaded VNF.
VNF Withdraw	Graphical representation that gives to the FP the ability to remove an already published or uploaded VNF
VNFs monitoring	Graphical representation of all monitoring data for a selected or "consumed" NF.
Billing information	Graphical representation of the Billing outcomes for a selected or "consumed" NF.
SLA information	Information of the selected or "consumed" NFs based on the agreed SLA and its fulfillment.

Table 2 FP dashboard view

Functionality	Short Explanation
AA	Authorization and Authentication of the respective role into the T-NOVA Dashboard.
Service request	Graphical representation of the Services/Functions returned by the T-NOVA business service catalogue.
Service Selection	Graphical representation assisted by a check box providing the ability to the customer to select a service for consumption.
Service configuration	Small Graphical wizard providing to the customer predefined parameters for defining the selected service.
Service monitoring	Graphical representation of the data gathered from the monitoring modules.
Billing information	Graphical representation of the billing outcomes of selected or "consumed" service.
SLA information	Details of the selected or "consumed" Service based on how they respect the agreed SLA.

Table 3 Customer Dashboard view

1.1. Deliverable's structure

The rest of the document is composed by the following sections:

Section 1 Introduction: Here, we give a short overview of the Dashboard and describe its functionalities.

Section 2 Requirements Overview: Here, we summarise the requirements the Dashboard must meet.

Section 3 Users Dashboard Design: This section discusses in brief the available web development frameworks and the existing software development and deployment approaches focusing in the ones exhibiting better scalability and interoperability.

Section 4 Users Dashboard Implementation: The implementation of each distinct functional part of the Users Dashboard is presented here.

Section 5 Users Dashboard Interfacing with T-NOVA Management Modules: In this section, we describe the interfaces of the Users Dashboard with the T-NOVA's management modules.

Section 6 Graphical User Interface and Functionalities: This section gives a step by step guide on how to use the implemented Dashboard.

Section 7 Conclusions: The conclusions of this document are given in this section.

2. USERS DASHBOARD DESIGN

2.1. Available Web development frameworks

2.1.1. Frontend

2.1.1.1. Semantic-UI

Semantic-UI [1] is a new front-end development framework rich in features and with a robust API. According to its development team, *"Semantic is a UI component framework based around useful principles from natural language."* To achieve this, Semantic-UI exploits a semantic, descriptive language for naming its classes, favoring the use of natural language words instead of abbreviations, as its counterparts mainly do.

This framework's uniqueness consists in its structured form, and in the provision of several components—namely the Feed, Comment, Shape and Sidebar—and a real-time debug feature not available in other frontend frameworks. Semantic-UI builds its structure around five descriptive categories (used for defining re-usable UI components) namely the UI Element, UI Collection, UI View, UI Module and UI Behavior. Additional advantages of Semantic UI are its minimal and neutral styling, leaving customization open to the developer, its components portability, its very good documentation and finally its website where many examples on how to use its different components can be found.

2.1.1.2. Bootstrap

"Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web." – Bootstrap [2]; a claim not far from truth. Bootstrap is a powerful, open-source front-end framework developed by a Twitter team. It combines HTML5/HTML, CSS3/CSS, and Javascript code towards enabling developers to easily build user interface components. It comes with a free set of tools and data APIs that allow the creation of flexible and responsive web layouts and common interface components (e.g. Scrollspy, Typeaheads) without even writing a single line of JavaScript code. Bootstrap's popularity builds on the following advantages:

- Faster development cycle: There is a plethora of predefined design templates and classes available to the developer.
- Responsiveness: Bootstrap's responsive features enable for the proper display of the web pages on heterogeneous devices and screen resolutions without any change in markup.
- Consistency: The same design templates and styles are provided to all Bootstrap components through a central library.
- User-friendly: A basic working knowledge of HTML and CSS suffices for starting development with Bootstrap.
- Compatibility: Bootstrap is compatible with all modern browsers such as Mozilla Firefox, Google Chrome, Safari, Internet Explorer, and Opera.

- Open Source: It is completely free to download and use.

2.1.2. Middleware

2.1.2.1. AngularJS

AngularJS [3] is an open-source web application framework, adhering to the Model-View-Controller (MVC) architecture, designed for creating single page web applications. In this direction, it exploits declarative programming for creating user interfaces and for connecting the software components to each other. AngularJS most notably feature is the two-way data binding where changes in any model are automatically reflected to the related views and, vice versa, any modification in the views leads to the respective models update. This feature along with advantages like the inherently support of MVC, the use of conventional HTML as the declarative language for defining interfaces and the use of directives for bringing additional functionality to HTML have made AngularJS a very popular choice amongst web developers.

2.1.2.2. jQuery

jQuery [4] is a JavaScript library that simplifies operations like HTML document traversal and manipulation (Document Object Model elements manipulation), event handling, animation, and Ajax by offering an easy-to-use API that is compatible with a multitude of browsers. Among its most notable features are the event assignment and the event callback function definitions, both done once in the code. The main advantages of using this library are:

- JavaScript and HTML separation: Javascript is use for adding event handlers to DOM, rather than adding HTML event attributes to call JavaScript functions.
- Brevity and clarity: Features like chainable functions and shorthand function names enable for brevity and clarity.
- Cross-browser compatible: jQuery provides one consistent interface that handles the interoperability amongst the browsers' different javaScript engines,
- Extensible: New events, elements, and methods can be easily added and then reused as a plugin.

2.1.2.3. Ember.js

Ember.js [5] is an open source, javaScript application framework, adhering to the Model-View-Controller (MVC) model, used for, as its developers' state, "*creating ambitious web applications*". It follows the Convention over Configuration (CoC), and the Don't Repeat Yourself (DRY) principles. Ember.js provides a series of tools and incorporates common idioms and best practices into the framework for reducing the amount of written code. Although it is a web framework, it is also possible to exploit it for building desktop and mobile applications. Ember was designed around several key ideas:

- Focus on ambitious web applications: Ember supports full MVC in contrast with most of its counterparts focusing in the V (view) part of MVC.
- Production friendly: Enables the developer to be productive immediately. In this direction, Ember utilises a pluggable architecture and offers a Command Line Interface (CLI) that provides a standard application structure and build pipeline.
- Stability without stagnation: Assures backward compatibility while still innovating and evolving the framework.

Ember 2.0 was released August 13, 2015 and the most important changes occur in the view layer and include the one way data flow by default, the "Just refresh it" when something changes, the introduction of standard lifecycle hooks for components and the exploitation of Glimmer rendering engine aiming to improve re-render performance.

2.1.2.4. React

React [6] is an open-source JavaScript library and is commonly exploited as the V in MVC. The library in highly interoperable as it makes no assumptions about the rest of the used technology stack. React abstracts the DOM, thus offering a simpler programming model and better performance. Contrary to its counterparts, React can also render on the server using Node. React aims to help developers building large applications that use time varying data. Its main features are:

- Use of vanilla JavaScript: React uses JavaScript's features for most of its operations (this explains React's lightweight API).
- One Way Data Flow: React utilises the simpler one way data flow than the more complex two way data flow. In this context, when the properties on a React component are updated, that component is re-rendered.
- Virtual DOM: React maintains its own virtual DOM, rather than using solely the browser's DOM. This allows the library to determine more efficient which parts of the browser's DOM need an update.
- Server-Side Rendering: Server-side rendering is a unique feature of React, especially important for high-traffic websites where the user experience (e.g. web page speed loading) must be excellent.

2.1.3. Back-End

2.1.3.1. Django

Django [7] is a free, open source, MVC aware, high-level Python Web framework promoting rapid development and clean, pragmatic design. As its developer states "*Django makes it easier to build better Web apps more quickly and with less code.*" Django eases the creation of complex, database-driven websites and emphasizes reusability and "pluggability" of components, adhering to the principle of DRY (Don't Repeat Yourself). It uses an object-relational mapper (ORM) that mediates between data models (defined as Python classes) and a relational database ("Model"); a system

for processing HTTP requests with a web templating system ("View") and a regular-expression-based URL dispatcher ("Controller"). Django is highly extensible and rich in features, some of them are:

- a lightweight and standalone web server for development and testing
- a form serialization and validation system which can translate between HTML forms and values
- a template system that utilises the concept of inheritance
- a caching framework
- support for middleware classes
- an internal dispatcher system
- an internationalization system
- a XML/JSON serialization system
- a system for extending the capabilities of the template engine
- an interface to Python's built in unit test framework
- an extensible authentication system
- a dynamic administrative interface
- tools for generating RSS and Atom syndication feeds and Google Sitemaps
- built-in mitigation for cross-site request forgery, cross-site scripting, SQL injection, password cracking and other typical web attacks
- a framework for creating GIS applications

2.1.3.2. Flask

Flask [8] is a BSD licensed, Python micro web application framework built with a small core and easy-to-extend philosophy. Flask is based on the Werkzeug WSGI (Web Server Gateway Interface) toolkit and Jinja2 template engine but it does not because it does not presume the use of a particular tool or library—It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, it is easy to add extensions to Flask and there are already a number of them providing object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools. Some of its features include:

- Provision of a development server and debugger
- Integrated support for unit testing
- RESTful request dispatching
- Jinja2 templating
- Support for secure cookies
- 100% WSGI 1.0 compliant
- Unicode-based
- Extensive documentation
- Google App Engine compatibility
- Extensions available to enhance features desired

2.1.3.3. Symfony

According to its developers, "*Symfony is a set of PHP Components, a Web Application framework, a Philosophy, and a Community — all working together in harmony.*" Symfony [9], an open source project, relies on PHP, the largest web development and has a vibrant and growing community (e.g. eZ Platform uses Symfony's Full Stack, Drupal 8 uses many Symfony Components). The framework introduces a truly unique HTTP and HTTP cache handling system by being an HTTP-centric request/response framework. In addition, it supports the use of advance features like ESI (Edge Side Includes) for separating the different parts of the web application.

2.1.3.4. YII

Yii [10] is an open-source, MVC aware, PHP framework used for developing Web 2.0 applications, promoting clean, DRY (Don't Repeat Yourself) design, and supporting rapid development. It offers excellent documentation and has a supportive community. Features like Database Access Objects (DAO), Active Record, and programmatic Database migrations ease the effort for developing database-powered web applications. In addition, it has built in support for form input, validation, Ajax, and built-in authentication. Yii also provides a built-in code generation tool, named Gii, speeding application development. It also integrates well with other frameworks (e.g. Zend, PEAR)and supports I18N for providing localized versions of the developed applications. Finally, Yii supports PSR-4 class auto-loading, provides a RESTful API framework and a documentation generator.

2.1.4. Summary

Based on the factors, of maturity, community support, number of supported features, extensibility and interoperability, we selected Bootstrap, Angular and Django as frontend, middleware and backend Web development frameworks respectively.

2.2. Development Approaches

2.2.1. Server-Centric Web Application (SCWA)

Server-Centric Web Application (SCWA) uses a Server to collect data, process them and serve the resulting HTML page to the client's browser. The disadvantage of this approach is that the page is posted back to the server; this introduces communication and processing overhead that can decrease the overall performance as it forces the user to wait for the page to be processed and recreated.

2.2.2. Browser-Centric Web Application (BCWA)

Browser-Centric Web Application (BCWA) approach embeds all the functional parts (e.g. Scripts) on the client's side and executes them on the client's Internet browser.

The advantage of BCWA architecture is the faster response time and less overhead (e.g. Data, Processing Power) on the web server. Additionally, BSWA is ideal when the page elements need to be changed without the need to contact the database.

A very popular technology belonging to the BCWA category is Single Page Application (SPA). Essentially SPA is a web application that fits in a single page and loads on the initial page request. SPA is suitable for web-centric applications that handle a large amount of data by exploiting, when needed, asynchronous download of features (HTML templates/JSON data) and by re-rendering locally any part of the interface without requiring from the server to re-render the complete HTML page. Finally, SPA allows web developers to give a "native-application" like experience to the end-user.

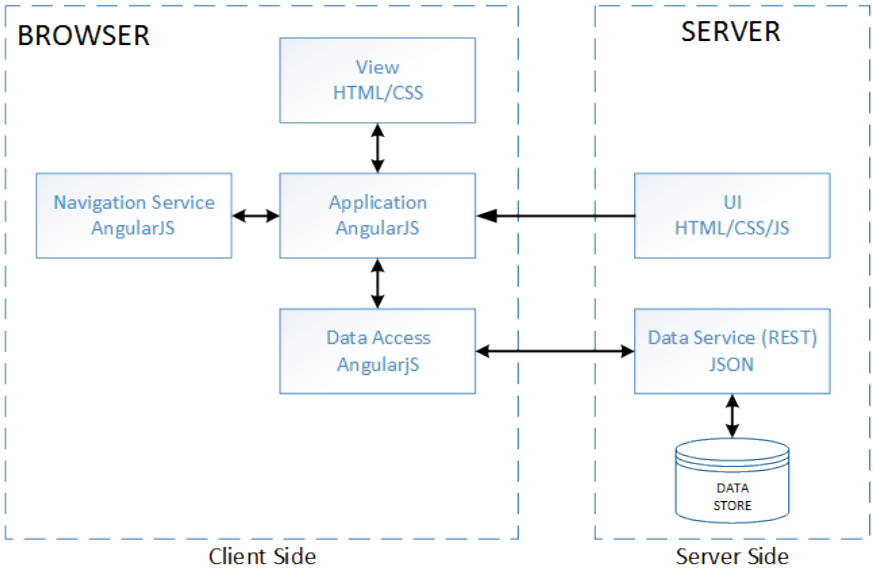

Figure 2-1 Single Page Application

2.2.3. Summary

For the Dashboard's implementation, the Browser-Centric Web Application architecture was chosen for the following two reasons: scalability and faster response time (excluding the initial load time). For the former, BCWA is more suitable as it diffuses the processing load from the server to the clients; in other words all the code is downloaded, from the server, at the session's beginning and then it runs inside the clients' browsers. For the latter, as all (or almost all) of the data are inside the browser the time for getting, processing and displaying them is usually far less than the time needed to request and get the data from the server (SCWA paradigm).

2.3. Deployment Architectures

2.3.1. Monolithic

A monolithic architecture is defined in the traditional client-server model. In this architecture all the functionalities are contained into a single service, which is the most

common way to develop and package web applications. A monolithic web application typically composed of the client, the server and the database. In this context, the Users' Dashboard along with the other modules are developed as a single application, exploiting one common web development framework (Django in Figure 2-2), and are deployed in one virtual machine.

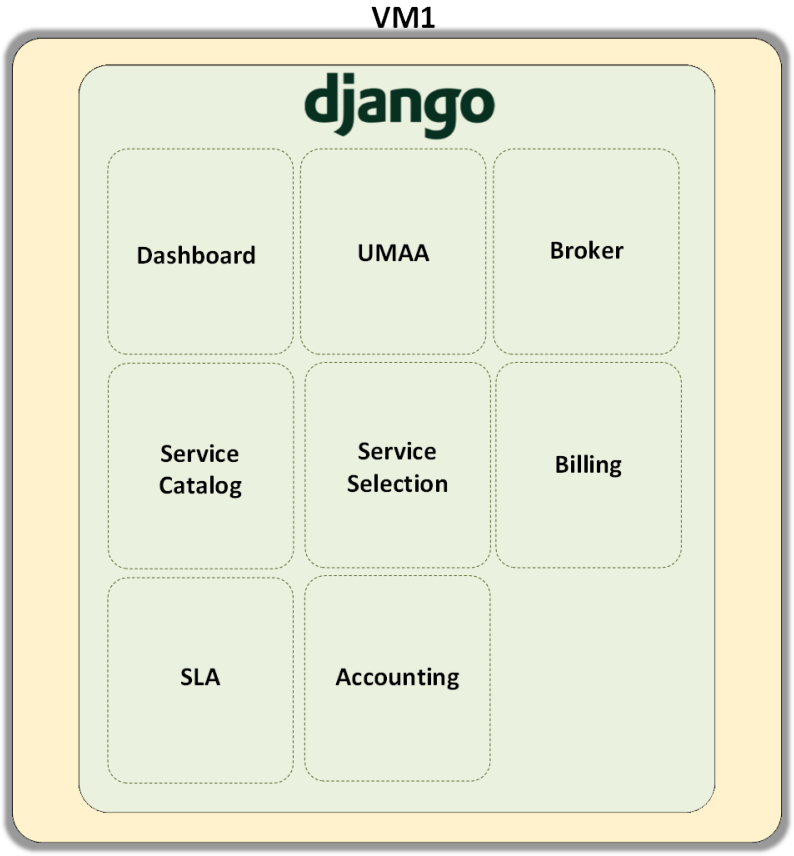

Figure 2-2 Dashboard in monolithic deployment

2.3.2. Micro-services

Micro-services architectural paradigm allows for the decomposition of one heavy application in several smaller components, named micro-services, which can be distributed across multiple locations. Each micro-service focuses in the execution of one or a small set of tasks, is independently deployable and uses, usually, an HTTP API for communicating with the rest micro-services. In this context, and as Figure 2-3 depicts, each T-NOVA module may be deployed as a distinct micro-service, exploiting a different development framework. Moreover, and for avoiding library versioning incompatibilities and conflicts, each micro-service can be deployed in a distinct virtual machine.

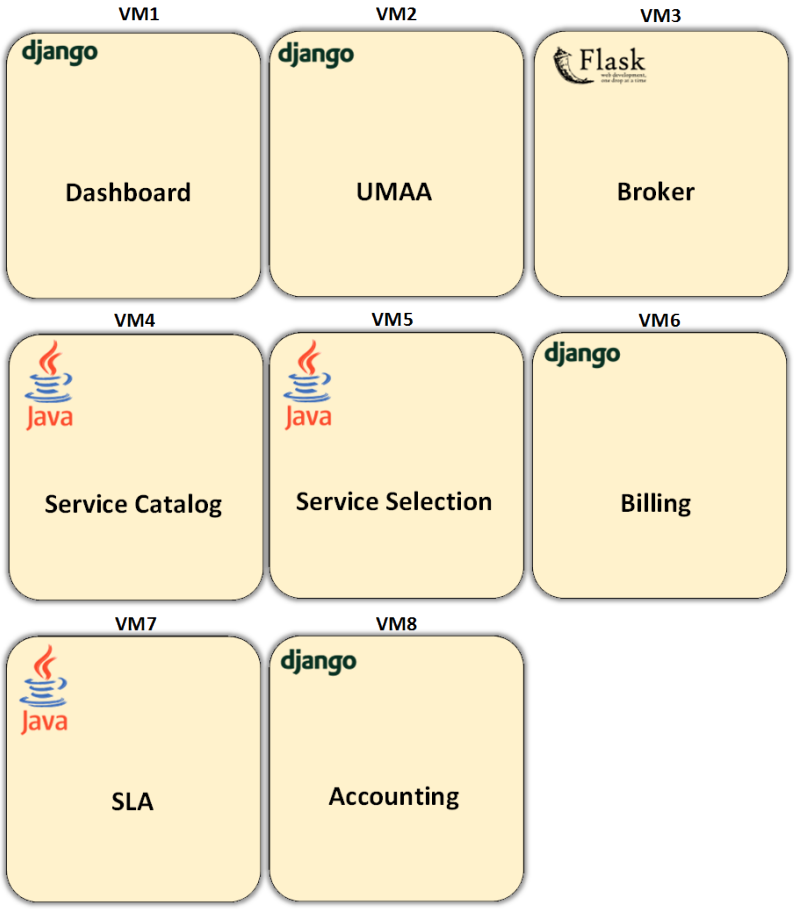

Figure 2-3 Dashboard deployed as a micro-service

2.3.3. Containerized micro-services

As Figure 2-4 depicts containerised micro-services refer to the deployment of each distinct micro-service inside a docker. Docker [11] is a container that packages all the software (e.g. code, libraries, third-party tools, etc.) needed to implement and run the micro-service. This allows the deployment of the micro-service, on any computer, on any infrastructure and in any cloud. In this context, each T-NOVA Marketplace module may be deployed as a distinct micro-service, exploiting a different development framework as in the "plain" micro-service case; however, with dockers there are no library versioning incompatibilities and conflicts and therefore all micro-services can be deployed in one virtual machine.

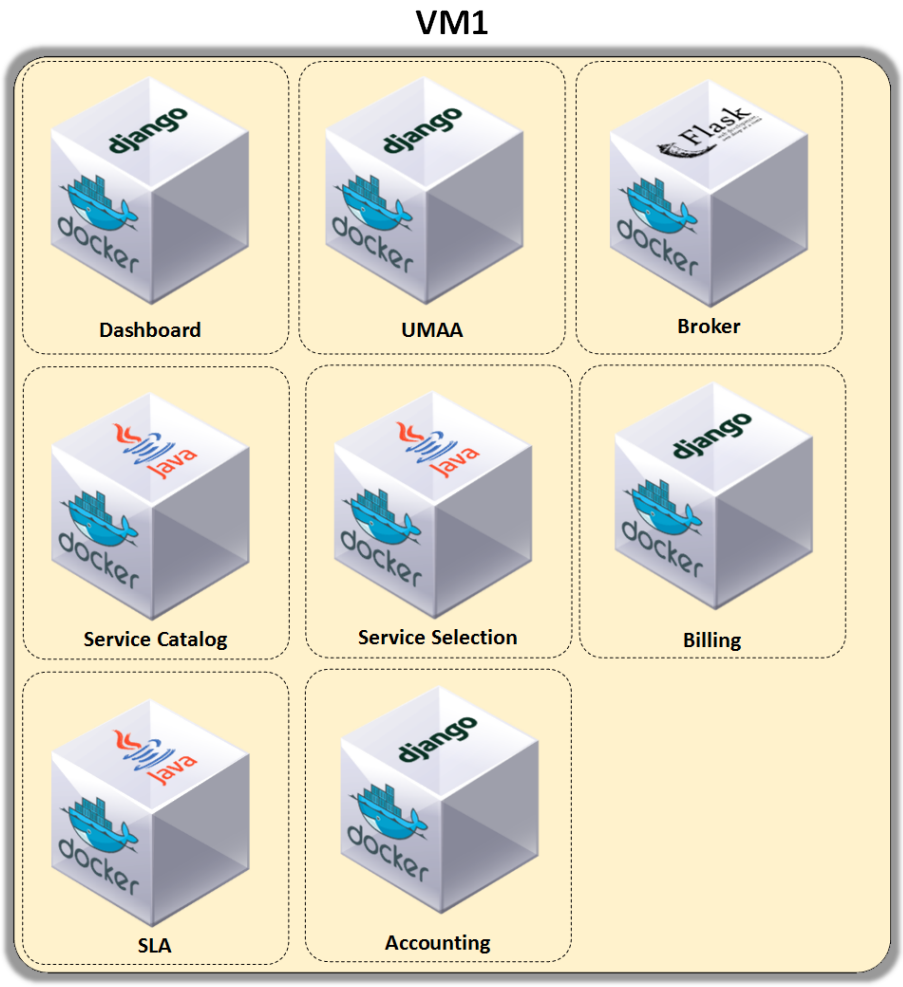

Figure 2-4 Dashboard deployed as a containerised micro-service

2.3.4. Summary

For the Dashboard's deployment, the containerised micro-service approach was selected. This will enable each developer assigned with the task of building a T-NOVA module to use its preferable development framework without having to take into account the selections of the other modules' developers (this may increase the development speed). Furthermore, the exploitation of docker technology enables for the deployment of all the modules in one virtual machine (VM) without having to worry about libraries incompatibilities and conflicts thus reducing the managerial and processing overhead needed to run a distinct VM for each micro-service.

3. USERS DASHBOARD IMPLEMENTATION

This section presents the implementation of the users' Dashboard in a series of code snippets. Each code snippet provides a snapshot of the code written to realise the Dashboard's functionalities.

3.1. Routing module

This module routes the users' requests (e.g. login, register) to the controller responsible for handling them.

```
// states definition
angular.module('dashboard').config(function ($stateProvider, $urlRouterProvider) {

    $urlRouterProvider.otherwise("/index/dashboard");

    $stateProvider
        .state('login', {
            url: "/login",
            templateUrl: "/static/dashboard/templates/login.html"
        })
        .state('register', {
            url: "/register",
            templateUrl: "/static/dashboard/templates/register.html"
        })
        .state('index', {
            url: "/index",
            templateUrl: "/static/dashboard/templates/index.html"
        })
        .state('index.profile', {
            url: "/profile",
            templateUrl: "/static/dashboard/templates/index.profile.html"
        })
        .state('index.dashboard', {
            url: "/dashboard",
            templateUrl: "/static/dashboard/templates/index.dashboard.html"
        })
        .state('index.users', {
            url: "/users",
            templateUrl: "/static/dashboard/templates/index.users.html"
        })
        .state('index.users.list', {
            url: "/list",
            templateUrl: "/static/dashboard/templates/index.users.list.html"
        })
        .state('index.users.new', {
            url: "/new",
            templateUrl: "/static/dashboard/templates/index.users.new.html"
        })
        .state('index.users.edit', {
            url: "/{userID:int}/edit",
            templateUrl: "/static/dashboard/templates/index.users.edit.html",
            controller: function ($scope, $stateParams) {
                $scope.userID = $stateParams.userID;
```

Figure 3-1 Routing module

3.2. Bootstrap authentication function

This function checks if a token exists in the cookies of the HTTP session and if yes it uses it for authenticating the user during the ongoing HTTP session; if not it redirects the user to the login page.

```
angular.module('dashboard', ['ui.bootstrap', 'ngSanitize', 'angularModalService', 'ui.select', 'ui.checkbox', 'ui-rangeSlider

angular.module('dashboard').run(['$state', '$rootScope', '$timeout', '$cookies', 'Restangular', function ($state, $rootScope
    $rootScope.isAuthorized = false;
    $rootScope.$state = $state;

    var appStarted = 0; // flag to redirect only once when app is started
    $rootScope.$on('$stateChangeStart',
        function (event, toState, toParams, fromState, fromParams) {
            if (appStarted) return;
            appStarted = 1;
            event.preventDefault(); //prevents from resolving requested url

            var token = $cookies.get('token');
            if (token) {
                console.log("JWT token exists");
                Restangular.setDefaultHeaders({Authorization: 'JWT ' + token});
                $rootScope.isAuthorized = true;
                $state.go('index.dashboard');
            } else {
                console.log("JWT not found");
                $state.go('login');
            }

        });
}]);

// change {{ }} default template tags to [[ ]] because conflict with django's template tags
angular.module('dashboard').config(function ($interpolateProvider) {
    $interpolateProvider.startSymbol('[[');
    $interpolateProvider.endSymbol(']]');
});

// setup restangular base url
angular.module('dashboard').config(function (RestangularProvider) {
    RestangularProvider.setBaseUrl('http://marketplace.t-nova.eu');
    RestangularProvider.setRequestSuffix('\/');
});
```

Figure 3-2 Bootstrap authentication function

3.3. Root controller

This controller contacts the UMAA module for checking the requesting user's privileges and based on them renders only the authorised content.

```
function RootCtrl(Restangular, $scope, $rootScope, $cookies, $state, $interval) {



    // check if is not authorized
    if (!$rootScope.isAuthorized) {
        $rootScope.$state.go('login');
    }

    $scope.available_groups = [];
    $scope.available_countries = [];
    $scope.user_permissions = [];
    $scope.user_profile = {};

    $scope.getGroupName = function (group_id) {
        var group_name = '';

        angular.forEach($scope.available_groups, function (value, key) {
            if (value.id == group_id) group_name = value.name;
        });

        return group_name;
    };

    $scope.getGroupID = function (group_name) {
        var group_id = -1;

        angular.forEach($scope.available_groups, function (value, key) {
            if (value.name == group_name) group_id = value.id;
        });

        return group_id;
    };

    $scope.hasPerm = function (perm) {
        var has_perm = false;

        angular.forEach($scope.user_permissions, function (value, key) {
            if (value == perm) has_perm = true;
        });

        return has_perm;
```

Figure 3-3 Root controller

3.4. Login controller

This controller contacts the UMAA module for authenticating the users.

```
function LoginCtrl(Restangular, $scope, $rootScope, $cookies, alertService) {

    $scope.init = function () {
        // check if is already authorized
        if ($rootScope.isAuthorized) {
            $rootScope.$state.go('index.dashboard');
        }
    };

    $scope.init();

    $scope.login = {};
    $scope.doLogin = function () {
        Restangular.one('auth').customPOST({username: $scope.login.username, password: $scope.login.password}).then(
            function (response) {
                Restangular.setDefaultHeaders({Authorization: 'JWT ' + response.token});

                var exp_date = new Date();
                exp_date.setDate(exp_date.getDate + 15); //15 days exp date
                $cookies.put('token', response.token, {expires: exp_date});

                $rootScope.isAuthorized = true;
                $rootScope.$state.go('index.dashboard');

            }, function (response) {
                console.log("Authentication error with status code " + response.status);
                if (response.status == 400) {
                    var error_message = response.data.non_field_errors[0];
                    console.log("Authentication error message: " + error_message);
                    alertService.add('danger', error_message);
                    //$alert({title: '',content: error_message,placement: 'top-right',type: 'danger',show: true,duration: 3});
                }
            });
    };

}
```

Figure 3-4 Login controller

3.5. Register controller

This controller contacts the UMAA module for registering the new Dashboard's users, namely the new Customers, Service and Function Providers.

```
306
307 function RegisterCtrl(Restangular, $scope, $rootScope, alertService) {
308
309     $scope.init = function () {
310         // check if is already authorized
311         if ($rootScope.isAuthorized) {
312             $rootScope.$state.go('index.dashboard');
313         }
314     };
315
316     $scope.init();
317
318     $scope.user = {};
319     $scope.user.group = '2';
320
321     $scope.doRegister = function () {
322
323         console.log($scope.user);
324         $scope.user.groups = [$scope.user.group];
325
326         Restangular.one('/user-management/register/').customPOST($scope.user).then(
327             function (response) {
328
329                 alertService.add('success', 'You have successfully registered!');
330                 $rootScope.$state.go('login');
331
332             }, function (response) {
333                 console.log("Registration error with status code " + response.status);
334                 if (response.status == 400) {
335                     alertService.add('danger', 'Registration failed.');
336
337                     var html_error = '<ul>';
338                     angular.forEach(response.data, function (value, key) {
339                         html_error = html_error+ '<li> Invalid '+key+'</li>';
340                     });
341                     html_error = html_error+ '</ul>';
342
```

Figure 3-5 Register controller

3.6. User management controller

This controller allows the Dashboard's administrator to manage (create, modify, delete) the user accounts.

```
1  angular.module('dashboard').controller('UsersListCtrl', ['Restangular', '$scope', '$rootScope', UsersListCtrl]);
2  angular.module('dashboard').controller('UserCreateCtrl', ['Restangular', '$scope', '$rootScope', '$state', UserCreateCtr
3  angular.module('dashboard').controller('UserEditCtrl', ['Restangular', '$scope', '$rootScope', '$stateParams', UserEditC
4  angular.module('dashboard').controller('UserProfileCtrl', ['Restangular', '$scope', '$rootScope', UserProfileCtrl]);
5
6
7  function UsersListCtrl(Restangular, $scope, $rootScope) {
8
9      $scope.loading_users = false;
10     $scope.users = {};
11
12     $scope.loadUsers = function () {
13         $scope.loading_users = true;
14         Restangular.all('user-management/users').getList().then(
15             function (response) {
16                 $scope.users = response;
17                 console.log("GetUserList " + response.length + " users found");
18                 $scope.loading_users = false;
19             }, function (response) {
20                 console.log("GetUserList error with status code " + response.status);
21                 console.log("GetUserList error message: " + response.data.detail);
22                 $scope.loading_users = false;
23             });
24     };
25
26     $scope.deleteUser = function (id, username) {
27         $scope.loading_users = true;
28         Restangular.one("user-management/users", id).remove().then(
29             function () {
30                 $scope.loading_users = false;
31                 console.log("User " + username + " has been successfully deleted");
32                 $scope.loadUsers();
33             }, function (response) {
34                 console.log("DeleteUser error with status code " + response.status);
35                 console.log("DeleteUser error message: " + response.data.detail);
36                 $scope.loading_users = false;
37
38             });
39     };
40
41     $scope.userDeleteConfirm = function (id, username) {
```

Figure 3-6 User management controller

3.7. VNF management controller

This controller interacts with the NFStore towards allowing Function Providers to create their functions.

```
angular.module('dashboard').controller('VNFCreateCtrl', ['Restangular', '$scope', '$rootScope', '$state', 'ModalService'
angular.module('dashboard').controller('VNFEditCtrl', ['Restangular', '$scope', '$rootScope', '$state', '$stateParams',
angular.module('dashboard').controller('VNFListCtrl', ['Restangular', '$scope', '$rootScope', 'ModalService', VNFListCtr


function VNFCreateCtrl(Restangular, $scope, $rootScope, $state, ModalService){

    $scope.vnf_types = [
        {code:"TC", desc: "Traffic Classification"},
        {code:"FW", desc: "Firewall"},
        {code:"HG", desc: "Home Gateway"},
        {code:"SA", desc: "Security Appliance"}
    ];

    $scope.memory_units = ['MB', 'GB'];
    $scope.storage_units = ['MB', 'GB', 'TB'];
    $scope.network_units = ['Kbps', 'Mbps', 'Gbps'];


    $scope.available_bandwidths = [
        "10Mbps",
        "100Mbps",
        "1Gbps",
        "10Gbps",
        "Unlimeted"
    ];


    $scope.billing_model_types = [
        {code: "PAYG", desc: "Pay-As-You-Go"},
        {code: "RS", desc: "Revenue Sharing"}
    ];

    $scope.billing_currencies = [
        {code: "EUR", desc: "Euro"},
        {code: "USD", desc: "US Dollars"}
    ];
```

Figure 3-7 VNF management controller

3.8. NFStore Images Management controller

This controller interacts with the NFStore towards allowing Function Providers to manage the images of their implemented functions.

```
angular.module('dashboard').controller('ImageListCtrl', ['Restangular', '$scope', '$rootScope', 'ModalService', ImageList
angular.module('dashboard').controller('ImageUploadCtrl', ['$scope', '$cookies', 'Upload', ImageUploadCtrl]);

function ImageListCtrl(Restangular, $scope, $rootScope, ModalService) {

    $scope.loading_images = false;
    $scope.images = {};

    $scope.loadImages = function () {
        $scope.loading_images = true;
        Restangular.all('vnfs/images').getList().then(
            function (response) {
                $scope.images = response;
                console.log("GetImageList " + response.length + " Images found");
                $scope.loading_images = false;
            }, function (response) {
                console.log("GetImageList error with status code " + response.status);
                console.log("GetImageList error message: " + response.data.detail);
                $scope.loading_images = false;
            });
    };

    $scope.deleteImage = function (image_name) {
        $scope.loading_images = true;
        Restangular.one("vnfs/images", image_name).remove().then(
            function () {
                $scope.loading_images = false;
                console.log("Image  " + image_name + " has been successfully deleted");
                $scope.loadImages();

            }, function (response) {
                console.log("DeleteImage error with status code " + response.status);
                console.log("DeleteImage error message: " + response.data.detail);
                $scope.loading_images = false;

            });
    };
```

Figure 3-8 NFStore Images Management controller

3.9. Services Management controller

This controller interacts with the Service Catalogue towards allowing Service Providers to manage their services (e.g. composite a service from one or more network functions).

```
function ServiceListCtrl(Restangular, $scope, $rootScope, $state, ModalService) {

    $scope.services = {};

    $scope.loadNSDs = function () {
        Restangular.all('nsds').getList().then(
            function (response) {
                $scope.services = response;
                console.log("GetNSDList " + response.length + " NSDs found");
            }, function (response) {
                console.log("GetNSDList error with status code " + response.status);
                console.log("GetNSDList error message: " + response.data.detail);
            });
    };

    $scope.loadNSDs();


    $scope.deleteNSD = function (id, nsd_name) {
        Restangular.one("nsds", id).remove().then(
            function () {
                console.log("NSD " + nsd_name + " has been successfully deleted");
                $scope.loadNSDs();
                //$alert({title: '', content: "VNF " + vnf_name + " has been successfully deleted", placement: 'bot-righ
                //$mdToast.show($mdToast.simple().content("VNF " + vnf_name + " has been successfully deleted").position
            }, function (response) {
                console.log("DeleteNSD error with status code " + response.status);
                console.log("DeleteNSD error message: " + response.data.detail);
                //$alert({title: '', content: "Failed to delete vnf " + vnf_name + ".", placement: 'bot-right', type: 'd
                //$mdToast.show($mdToast.simple().content("Failed to delete vnf " + vnf_name).position('top right').hide
            });
    };
```

Figure 3-9 Service Management controller

3.10. Broker Controller

This controller interacts with the Brokerage module towards allowing Service Providers and Customers to participate in the trading operation.

```
    $scope.getPendingRequests = function(){

        var pending_requests = 0;

        angular.forEach($scope.trades, function (trade, trade_key) {

            if (trade.status=='pending') {
                pending_requests++;
            }

        });

        return pending_requests;
    };

    $scope.acceptTradeRequest = function (id) {

        Restangular.one("/broker/vnfs/trade/"+id+"/accept").get().then(
            function (response) {
                console.log("Trade Request " + response.id + " accepted");
            }, function (response) {
                console.log("Accept Trade Request error with status code " + response.status);
                console.log("Accept Trade Request error message: " + response.data.detail);
            });

    };

    $scope.rejectTradeRequest = function (id) {

        Restangular.one("/broker/vnfs/trade/"+id+"/reject").get().then(
            function (response) {
                console.log("Trade Request " + response.id + " rejected");
            }, function (response) {
                console.log("Reject Trade Request error with status code " + response.status);
                console.log("Reject Trade Request error message: " + response.data.detail);
            });

    };
```

Figure 3-10 Broker controller

4. USERS DASHBOARD INTERFACING WITH REST T-NOVA MARKETPLACE MODULES

This section presents the interface between the users' Dashboard and the rest t-NOVA marketplace modules, namely the Billing, UMAA (User Management, Authentication and Access Control), SLA management, Accounting, Brokerage, Business Service Catalogue, Function Store and Orchestrator.

4.1. APIs Definition

4.1.1. Billing

The billing API for the dashboard manages the following information between the dashboard and the billing module:

- Bills charged per user and per service (SP and customer).
- Charges done to SP's customers (BSS functionality to the SP).
- Charges done to FP's customers, which is the SP.

Queries

Usage query API for getting user's data

URL	http://localhost:8080/udr/usage/users/{user_id}
Type	GET
Headers	x-auth-token : String
Parameters	from : Date to : Date
Response Code	200 : Success
Request	None

Response	{
	"userid": "49588f5cea984040bc05d871eff67d2f",
	"time": {
	"to": "2015-01-12 01:10:00",
	"from": "2015-01-12 01:01:00"
	},
	"usage": {
	"openstack": [
	{
	"name": "cpu_util",
	"columns": [
	"time",
	"sequence_number",
	"avg"
	],
	"points": [
	[
	1421024460734,
	124666640001,
	74.31932
	],
	[
	1421024460734,
	124666550001,
	0.7899716
	]
	]
	}
	]

	} }

Usage query API for particular resource / service id

URL	http://localhost:8080/udr/usage/resources/{resource_id}
Type	GET
Headers	x-auth-token : String
Parameters	from : Date to : Date
Response Code	200 : Success
Request	None

Response	{
	"resourceid": "49588f5cea984040bc05d871eff67d2f",
	"time": {
	"to": "2015-01-12 01:10:00",
	"from": "2015-01-12 01:01:00"
	},
	"column": [

	"time", "mean", "userid"], "usage": [[0, 0, "46fe4a610a8b44948a5b61427b0b5ecd"], [0, 0, "49588f5cea984040bc05d871eff67d2f"], [0, 2.950836399999999, "99909daae8924e7a9b96cd964e9d64e3"],] }

Bill generation API for a particular customer

URL	http://localhost:8080/billing/invoice
Type	GET
Headers	x-auth-token : String
Parameters	Customerid: String from : Date

	to : Date
Response Code	200 : Success
Request	None
Response	{ "time": { "to": "2015-06-15 23:59", "from": "2015-06-15 00:00" }, "charge": { "columns": ["time", "sequence_number", "userid", "usage", "price", "resource"], "points": [[1434361731726, 413986240001, "f83aa92bc3c64a3497b334cc712b0491", 5, 15.84, "service-id-aaab-hg1562711-ahsbba"], [1434361731726, 413986230001,

	"f83aa92bc3c64a3497b334cc712b0491", 37, 124.4, "service-id-aaac-hg1562711-ahsbbs"]] } }

4.1.2. User Management, Authentication and Access control Module

UMMA's API collects all the information necessary to manage and authenticate the T-NOVA users or stakeholders. The root prefixes for this micro-service are **/user-management/** for the user/profile management and **/auth/** for the authentication. The micro-service uses these YAML files to initialize the permissions, groups and users:

- init_users.yml
- init_groups.yml
- init_permissions.yml

4.1.2.1. Authentication

Methods

- POST /auth/

User authentication request

POST /auth/
Request Body: {"username":"admin", "password":"123456"} ***Response Body:*** {"token":" eyJhbGciOiJIUzI1N....iIsInR5cCI6IkpXV "}

Request Example with JWT authentication

GET /user-management/users/
Content-Type: application/json

Authentication: JWT eyJhbGciOiJIUzI1N….iIsInR5cCI6IkpXV

Token Metadata


```
{
   'username': 'admin',
   'user_id': 1,
   'email': 'g.alexiou@pasiphae.eu',
   'company_name': 'TEIC',
   'exp': 1436620914
}
```


4.1.2.2. User Management

Methods

- GET /user-management/users/
- POST /user-management/users/
- GET /user-management/users/{pk}/
- PUT /user-management/users/{pk}/
- PATCH /user-management/users/{pk}/
- DELETE /user-management/users/{pk}/
- GET /user-management/users/{pk}/groups/
- GET /user-management/users/{pk}/permissions/
- GET /user-management/profile/
- PUT /user-management/profile/
- PATCH /user-management/profile/
- GET /user-management/profile/groups/
- GET /user-management/profile/permissions/
- GET /user-management/groups/
- GET /user-management/countries/

Listing all Users

This method returns the user's list.

GET /user-management/users/

Response Body:


```
[
 {
  "id": 1,
  "username": "admin",
  "groups": [
   3
```

```
  ],
  "company_name": "TEIC",
  "first_name": "",
  "last_name": "",
  "email": "g.alexiou@pasiphae.eu",
  "country": "GR",
  "city": "",
  "address": ""
 },
 {
  "id": 2,
  "username": "customer1",
  "groups": [
   1
  ],
  "company_name": "TEIC",
  "first_name": "",
  "last_name": "",
  "email": "customer1@t-nova.eu",
  "country": "GR",
  "city": "",
  "address": ""
 }
]
```


Create New User

This method creates a new user.

POST /user-management/users/

Request Body:


```
{
  "username": "new_user",
  "password": "123456",
  "groups": [
   3
  ],
  "company_name": "TEIC",
  "first_name": "",
  "last_name": "",
  "email": "g.alexiou@pasiphae.eu",
  "country": "GR",
  "city": "",
  "address": ""
```

```
}
Response Body:
{
  "id": 10,
  "username": "new_user",
  "groups": [
    3
  ],
  "company_name": "TEIC",
  "first_name": "",
  "last_name": "",
  "email": "g.alexiou@pasiphae.eu",
  "country": "GR",
  "city": "",
  "address": ""
}
```


Get User's info

This method returns the user info.


```
GET /user-management/users/10/
Response Body:
{
  "id": 10,
  "username": "new_user",
  "groups": [
    3
  ],
  "company_name": "TEIC",
  "first_name": "",
  "last_name": "",
  "email": "g.alexiou@pasiphae.eu",
  "country": "GR",
  "city": "",
```

```
  "address": ""
}
```

Edit User's info

This method changes user's info (PUT Method)

PUT /user-management/users/10/

Request Body:

```
{
  "id": 10,
  "username": "new_user",
  "groups": [
    3
  ],
  "company_name": "TEIC",
  "first_name": "George",
  "last_name": "Alexiou",
  "email": "g.alexiou@pasiphae.eu",
  "country": "GR",
  "city": "Heraklion",
  "address": "Estaurmenos"
}
```

This method changes user's info partially (PATCH Method)

PATCH /user-management/users/10/

Request Body:

```
{
  "first_name": "George",
  "last_name": "Alexiou",
  "city": "Heraklion",
  "address": "Estaurmenos"
}
```

Delete User

This method deletes a user

DELETE /user-management/users/10/

Get User's groups

This method returns the user groups.

GET /user-management/users/10/groups/

Response Body:
[
{
"id": 10,
"name": "Customer",
"permissions": [
"umaa.edit_own_profile",
"umaa.view_own_profile"
]
}
]

Get User's permissions

This method returns the user permissions.

GET /user-management/users/10/permissions/
Response Body: ["umaa.edit_own_profile", "umaa.view_own_profile"]

Get User's profile

This method returns the user's profile.

GET /user-management/profile/
Response Body: { "id": 10, "username": "new_user",

```
  "groups": [
    3
  ],
  "company_name": "TEIC",
  "first_name": "George",
  "last_name": "Alexiou",
  "email": "g.alexiou@pasiphae.eu",
  "country": "GR",
  "city": "Heraklion",
  "address": "Estaurmenos"
}
```


Edit User's profile

This method changes user's profile info (PUT Method)

PUT /user-management/profile/
Request Body:


```
{
  "id": 10,
  "username": "new_user",
  "groups": [
    3
  ],
  "company_name": "TEIC",
  "first_name": "George",
  "last_name": "Alexiou",
  "email": "g.alexiou@pasiphae.eu",
  "country": "GR",
  "city": "Heraklion",
  "address": "Estaurmenos"
}
```


This method changes user's profile info partially (PATCH Method)

PATCH /user-management/profile/

Request Body:

```
{
  "first_name": "George",
  "last_name": "Alexiou",
  "city": "Heraklion",
  "address": "Estaurmenos"
}
```

Get User's profile groups

This method returns the user groups.

GET /user-management/profile/groups/

Response Body:

```
[
   {
      "id": 10,
      "name": "Customer",
      "permissions": [
         "umaa.edit_own_profile",
         "umaa.view_own_profile"
      ]
   }
]
```

Get User's profile permissions

This method returns the user permissions.

GET /user-management/profile/permissions/

Response Body:

```
[
   "umaa.edit_own_profile",
   "umaa.view_own_profile"
]
```

Get Available Groups

This method returns the available groups.

GET /user-management/groups/
Response Body: [{"id":3,"name":"Administrator"}, {"id":1,"name":"Customer"}, {"id":4,"name":"Function Provider"}, {"id":2,"name":"Service Provider"}]

Get Available Countries

This method returns the available countries.

GET /user-management/countries/
Response Body: [{code":"AF","name":"Afghanistan"}, ... {"code":"ZM","name":"Zambia"}, {"code":"ZW","name":"Zimbabwe"}]

4.1.3. SLA Management

This API's goal is to show the users the following information coming from the SLA management module:

- SLA template specification to be filled by the SP and FPs.
- SLA offering to the customer and associated to each service.
- SLA monitoring by all the stakeholders.
- SLA penalties for each network service and VNF.

- **Providers**

POST Creates a provider. The uuid is in the request

URL	/providers
Type	POST
Headers	Accept: application/json

	Content-type: application/json
Parameters	
Response code	409: The uuid or name already exists in the database. 201: Created.
Request example	POST /providers/ HTTP/1.1

POST item example	{
	"uuid":"fc923960-03fe-41eb-8a21-a56709f9370f",
	"name":"provider-prueba"
	}

- **Templates:**

POST Creates a new template. The file might include a Template Id or not. In case of not being included, a uuid is assigned.

URL	/templates
Type	POST
Headers	Accept: application/json Content-type: application/json
Parameters	
Response code	▪ 409: The uuid already exists in the database. ▪ 409: The provider uuid specified in the template doesn't exist in the database. ▪ 500: Incorrect data has been suplied. ▪ 201: Created.
Request example	POST /templates/ HTTP/1.1
POST item example	SLA template (see annex section 2).

UPDATE Updates the template identified by *TemplateId*. The body might include a *TemplateId* or not. In case of including a *TemplateId* in the file, it must match with the one from the url.

URL	/templates/{templateId}
Type	PUT
Headers	Accept: application/json

	Content-type: application/json
Parameters	▪ TemplateId: Id of the template we want to modify.
Response code	▪ 409: The *templateId* from the url doesn't match with the one from the file. ▪ 409: Template has agreements associated. ▪ 409: Provider doesn't exist ▪ 500: Incorrect data has been suplied ▪ 200: OK
Request example	PUT /templates/vnfvnf5gold HTTP/1.1
PUT item example	SLA template (see annex section 2).

GET Retrieves a template identified by *templateId*.

URL	/templates/{templatetId}
Type	GET
Headers	Accept: application/json Content-type: application/json
Parameters	▪ *templateId*: Id of the template we want to retrieve.
Response code	▪ 404: The *templateId* doesn't exist in the database. ▪ 200: OK.
Request example	GET /templates/vnfvnf5gold HTTP/1.1
Response example	SLA template in JSON form (see annex section 2)

- **Agreements**

GET Retrieves an agreement identified by *agreementId*.

URL	/agreements/{agreementId}
Type	GET
Headers	Accept: application/json Content-type: application/json
Parameters	▪ *agreementId*: Id of the agreement we want to retrieve.
Response code	▪ 404: The uuid doesn't exist in the database.

	▪ 200: OK.
Request example	GET /agreements/vnfidf51 HTTP/1.1
Response example	SLA agreement in JSON form (see annex section 3)

The *AgreementId* matches the *AgreementId* attribute of *wsag:Agreement* element when the agreement is created.

- **SLA Information**

Retrieve SLA related information to show in the dashboard given the userId and the wheter you want to retrieve VNFs or network services.

URL	/sla-info/?clientId={clientId}&kind={ns|vnf}
Type	GET
Headers	Accept: application/json Content-type: application/json
Parameters	*clientId*: Id of the user that is using the network service or the VNF. *kind*: It can take 2 values: ns | vnf.
Response code	400 - Bad request: when the body in the request is not well formed. 500: There is a connection problem between the Accounting and the SLA modules. 200: OK.
Request example	GET /sla-info/?clientId=c1&kind=ns HTTP/1.1

Response example	[
	{
	"productId": "service6",
	"productType": "ns",
	"clientId": "c1",
	"providerId": "p6",
	"SLAPenalties": 35,
	"agreementId": "serviceids101",
	"dateCreated": "2015-10-08T07:31:37Z",
	"dateTerminated": "2015-12-15T17:26:45.071444"
	}

	]

Retrieves the list of all running services the user (customer) is using.

URL	/servicelist/{userId}/
Type	GET
Headers	Accept: application/json Content-type: application/json
Parameters	*userId*: Id of the user for whom we want to retrieve the service list.
Response code	400 - Bad request: when the body in the request is not well formed. 200: OK.
Request example	GET /servicelist/c1 HTTP/1.1
Response example	[{ "id": 2, "instanceId": "id02", "productId": "s1", "agreementId": "s1vnf2_4", "relatives": "id01, id03", "productType": "ns", "flavour": null, "startDate": "2015-06-11T00:00:00Z", "lastBillDate": "2015-06-11T00:00:00Z", "providerId": "p1", "clientId": "c1", "status": "running", "billingModel": "PAYG", "period": "P1D", "priceUnit": "EUR", "periodCost": 1.0, "setupCost": 1.0, "renew": true, "dateCreated": "2015-06-11T13:29:16Z",

	"dateModified": "2015-12-10T09:29:41Z" },]

Retrieves the list of all running VNFs the user (service provider) is using.

URL	/vnflist/{userId}/
Type	GET
Headers	Accept: application/json Content-type: application/json
Parameters	*userId*: Id of the user for whom we want to retrieve the service list.
Response code	400 - Bad request: when the body in the request is not well formed. 200: OK.
Request example	GET /vnflist/p5 HTTP/1.1
Response example	[{ "id": 24, "instanceId": "idf50", "productId": "vnf5", "agreementId": "s1vnf2_4", "relatives": "ids100", "productType": "vnf", "flavour": null, "startDate": "2015-10-08T07:07:43Z", "lastBillDate": "2015-10-08T07:07:43Z", "providerId": "f5", "clientId": "p5", "status": "running", "billingModel": "PAYG", "period": "P1D", "priceUnit": "EUR", "periodCost": 1.0, "setupCost": 2.0,

	"renew": false, "dateCreated": "2015-10-08T07:07:43Z", "dateModified": "2015-10-08T07:07:43Z" }]

4.1.4. Billing

- **Bill Report**

Returns all the CDRs for a user within a specified date.

GET /billing/invoice?userId={userId}&from={from_date}&to={to_date}
Response Body: { "name": "charge", "from": "2015-12-03 00:00:00", "to": "2015-12-09 23:59:59", "charges": [{ "priceUnit": "EUR", "setupCost": 0, "resource": "id02", "providerId": "p1", "price": 0.003472222222222222, "usage": "300", "time": "2015-12-09T08:10:52Z", "userId": "c1", "periodCost": 1 }, { "priceUnit": "EUR", "setupCost": 0, "resource": "id02",

```
    "providerId": "p1",
    "price": 0.003472222222222222,
    "usage": "300",
    "time": "2015-12-09T08:10:52Z",
    "userId": "c1",
    "periodCost": 1
   }
 ]
}
```


- **Revenue Sharing Report**

Returns the revenues that a specified Service Provider owes to a VNF Provider

GET
/billing/revenue/report?vfpId={fpId}&spId={spId}&from={from_date}&to={to_date}

Response Body:


```
{
 "VNFProvider": "f1",
 "from": "2015-12-07 00:00:00",
 "to": "2015-12-09 23:59:59",
 "revenues": [
  {
   "priceUnit": "EUR",
   "instanceId": "id01",
   "price": 0.003472222222222222,
   "VNFProvider": "f1",
   "name": "tnova_revenue_sharing",
   "SProvider": "p1",
   "time": "2015-12-09T08:10:52Z"
  },
  {
   "priceUnit": "EUR",
   "instanceId": "id03",
   "price": 0.003472222222222222,
```

```
      "VNFProvider": "f1",
      "name": "tnova_revenue_sharing",
      "SProvider": "p1",
      "time": "2015-12-09T08:10:52Z"
    }
  ]
}
```


Returns the revenues that a NON specified Service Provider owes to a VNF Provider

GET /billing/revenue/report?vfpId={fpId}&from={from_date}&to={to_date}

Response Body:


```
{
  "VNFProvider": "f1",
  "from": "2015-12-07 00:00:00",
  "to": "2015-12-09 23:59:59",
  "revenues": [
    {
      "priceUnit": "EUR",
      "instanceId": "id01",
      "price": 0.003472222222222222,
      "VNFProvider": "f1",
      "name": "tnova_revenue_sharing",
      "SProvider": "p2",
      "time": "2015-12-09T08:10:52Z"
    },
    {
      "priceUnit": "EUR",
      "instanceId": "id03",
      "price": 0.003472222222222222,
      "VNFProvider": "f1",
      "name": "tnova_revenue_sharing",
      "SProvider": "p1",
```

```
      "time": "2015-12-09T08:10:52Z"
    }
  ]
}
```

- **Usage Data Records**

Return usage data records

GET udr/usage/users/{customerId}?from={from_date}&to={to_date}

Response Body:

```
{
    "name": "UDR",
    "from": "2015-12-10 00:00",
    "to": "2015-12-10 14:56",
    "instanceId": "id02",
    "usages": [
        {
            "time": "2015-12-10T07:36:27Z",
            "usage": 300
        },
        {
            "time": "2015-12-10T07:36:27Z",
            "usage": 300
        },
        {
            "time": "2015-12-10T07:36:27Z",
            "usage": 300
        }
    ]
}
```

4.1.5. Brokerage

This interface is exploited for trading issues, among the T-NOVA users (i.e. Service Providers and Function Providers) and the brokerage module. The information that will go through this API will be related to:

- Service composition/VNF request: this functionality enables the SP to request network functions or compose a new service.
- Advertise VNF: this functionality is exploited for the communication between FP and the brokerage module, as the latter perform the intermediate communication, this is trading.

Get Available VNFs

Returns the available VNFs

GET /broker/vnfs/?<filters>

Response Body:
[
{
provider: "TEIC",
release: "T-NOVA",
type: "FW",
id: 562,
description: "PFSense is a firewall…"
…
},{
provider: "TEIC",
release: "T-NOVA",
type: "FW",
id: 563,
description: "UNTagle is a firewall…"
…
}
]

New Trade Request

This method creates a new trade request

POST /broker/vnfs/trade/

Request Body:
{
provider_id: 6,
vnf_id: 563,
price_override: 12.0
}

Get Trade Request

This method returns trade request

GET /broker/vnfs/trade/4

Response Body:
{
created_at: %Y-%m-%dT%H:%M:%SZ,
modified_at: %Y-%m-%dT%H:%M:%SZ,
provider_id: 6,
vnf_id: 563,
price_override: 12.0,
status:”pending”,
id:4
}

Get Trades Requests

This method returns a list of trade requests

GET /broker/vnfs/trade/
Response Body: [{

created_at: %Y-%m-%dT%H:%M:%SZ,

modified_at: %Y-%m-%dT%H:%M:%SZ,

provider_id: 6,

vnf_id: 563,

price_override: 12.0,

status:"pending",

id:4

}

{

created_at: %Y-%m-%dT%H:%M:%SZ,

modified_at: %Y-%m-%dT%H:%M:%SZ,

provider_id: 6,

vnf_id: 564,

price_override: 6.0,

status:"accepted",

id:5

}

]

Accept Trade Requests

This method accepts a trade request offer

GET /broker/vnfs/trade/5/accept/

Reject Trade Requests

This method rejects a trade request offer

GET /broker/vnfs/trade/5/reject/

4.1.6. Function Store

This interface allows the FPs to publish and manage their VNFs into the NF Store. The publication consists in uploading the VNF image, registering the VNF and its metadata

into the function store. The VNFs are versioned allowing the FPs to provide further upgrades. Finally, the FPs can remove their VNFs. In summary, the information managed with this interface is:

- VNF image and VNF metadata descriptor.
- VNF version.
- Upload, upgrade and delete the VNF package.

Methods

- GET /vnfs/
- POST /vnfs/
- GET /internal/vnfs/
- GET /vnfs/{pk}/
- PUT /vnfs/{pk}/
- DELETE /vnfs/{pk}/
- GET /vnfs/{pk}/yaml/
- PUT /vnfs/{pk}/yaml/

Listing all VNFs

This method return VNF list.

GET /vnfs/ GET /internal/vnfs/ (for module to module communication only)

Response Body:
[
{
provider: "TEIC",
release: "T-NOVA",
type: "FW",
id: 562,
description: "PFSense is a firewall..."
...
},{
provider: "TEIC",
release: "T-NOVA",
type: "FW",
id: 563,
description: "UNTagle is a firewall..."
...
}
]

Create New VNF

This method creates a new VNF.

POST /vnfs/

Request Body:
{
provider: "TEIC",
release: "T-NOVA",
type: "FW",
id: 562,
description: "PFSense is a firewall…"
…
}

Get VNFD

This method returns a vnfd

GET /vnfs/562/

Response Body:
{
provider: "TEIC",
release: "T-NOVA",
type: "FW",
id: 562,
description: "PFSense is a firewall…"
…
}

Update VNFD

This method updates a vnfd

PUT /vnfs/562/

Request Body:
{
provider: "TEIC",
release: "T-NOVA",
type: "FW",
id: 562,
description: "PFSense is a firewall…"
…
}

Delete VNFD

This method deletes a vnfd

DELETE /vnfs/562/

Get YAML VNFD

This method returns a vnfd file in YAML format

GET /vnfs/562/yaml/
Response Body:
id: 562
modified_at: '2015-10-18T17:50:09Z'
name: pfSense Firewall
provider: TEIC
provider_id: 4
release: T-NOVA
....

Update YAML VNFD

This method updates a vnfd file in YAML format

PUT /vnfs/562/yaml/
Request Body:
id: 562
modified_at: '2015-10-18T17:50:09Z'
name: pfSense Firewall
provider: TEIC
provider_id: 4
release: T-NOVA
....

4.1.7. Business Service Catalogue

Get available services

GET /service/catalog
Response Body:
[
{

```
    "nsd": {
      "id": "56931f5de4b0c74fd56e890d",
      "name": "network-service-0",
      "vendor": "T-NOVA",
      "version": "1.0",
      "manifest_file_md5": "fa8773350c4c236268f0bd7807c8a3b2",
      "vnfds": [
        "52439e7c-c85c-4bae-88c4-8ee8da4c5485"
      ],
...
      ],
      "auto_scale_policy": {
        "criteria": [
          {
            "end-to-end bandwidth": "10Mbps"
          },
          {
            "test": "test"
          }
        ],
        "action": "upgrade"
      },
      "connection_points": [
        {
          "connection_point_id": "mgnt0",
          "type": "ip"
        },
        {
          "connection_point_id": "data0",
          "type": "bridge"
        },
        {
          "connection_point_id": "stor0",
```

```
        "type": "bridge"
      }
    ]
  }
 }
]
```

Create new Service

POST /service/catalog

Request Body:

```
{
 "nsd": {
  "auto_scale_policy": {
   "action": "string",
   "criteria": [
    {
     "end-to-end bandwidth": "string",
     "test": "string"
    }
   ]
  },
  "connection_points": [
   {
    "connection_point_id": "string",
    "type": "string"
   }
  ],
  "id": "string",
  ...
          "constituent_vnfs": [
           "string"
          ],
          "graph": [
```

```
            "string"
          ],
          "nfp_id": "string"
        }
      ],
      "number_of_endpoints": 0,
      "number_of_virtual_links": 0,
      "vnffg_id": "string"
    }
  ]
 }
}
```


Get service information

GET /service/catalog/{serviceId}

Response Body:


```
{
 "nsd": {
  "auto_scale_policy": {
   "action": "string",
   "criteria": [
    {
     "end-to-end bandwidth": "string",
     "test": "string"
    }
   ]
  },
  ...
          "graph": [
           "string"
          ],
          "nfp_id": "string"
```

```
          }
         ],
         "number_of_endpoints": 0,
         "number_of_virtual_links": 0,
         "vnffg_id": "string"
       }
     ]
   }
 }
}
```


Delete service

DELETE /service/catalog/{serviceId}

4.1.8. Service Selection

Service instantiation

POST /service/selection

Request Body:
{
"created_at": "string",
"id": 0,
"nsd_id": "string",
"status": "string",
"updated_at": "string",
"vnfs": [
{
"id": 0,
"vnfd_id": "string",
"vnfi_id": "string"
}
]
}

Response Body:
{
"callbackUrl": "string",
"customer_id": "string",
"nap_id": "string",
"ns_id": "string"
}

Service termination

DELETE /service/selection/{service_instance_id}

4.1.9. Orchestrator

Get Monitoring Information

Dashboard use this interface to the orchestrator in order to retrieve monitoring information.

GET orchestrator/instances/{instanceId}/monitoring-data/?instance_type={instance_type}&metric={metric}

Response Body:
[
{
"metricname": "packets_per_second",
"value": 100,
"date: "2015-01-21T18:49:00CET",
}
]

Parameters:

- instanceType: it can be "ns" or "vnf"
- instanceId: instance ID of the service or function
- metric: name of the metric we want to obtain the monitoring data

5. Graphical User Interface and Functionalities

5.1. User Registration

The Customers or the Function Providers can register to the dashboard from this page.

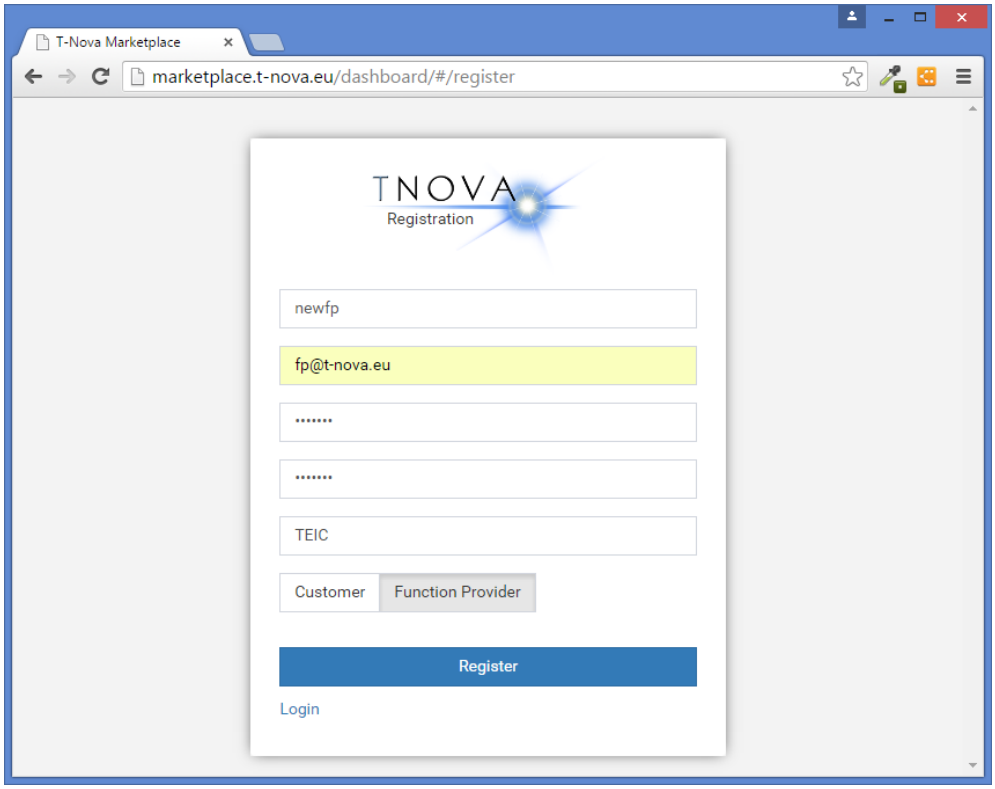

Figure 5-1 User registration

5.2. User Login

This the Dashboard's login page.

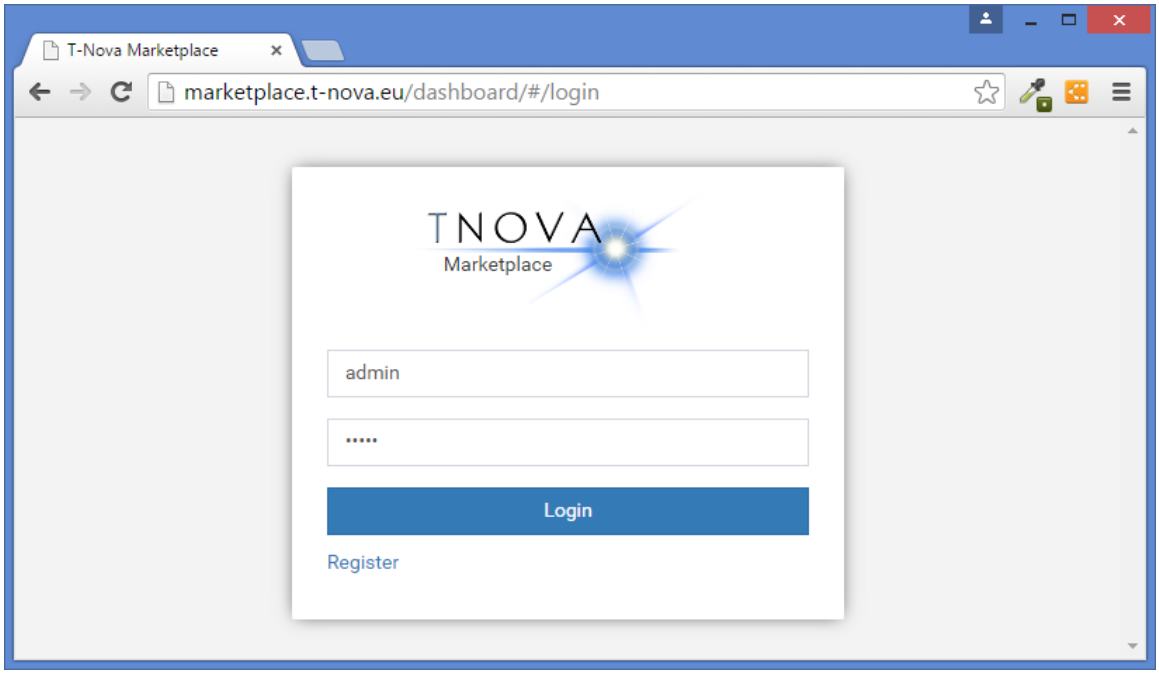

Figure 5-2 Dashboard's login page

5.3. VNF Provider

This is the VNF Providers' Home Page.

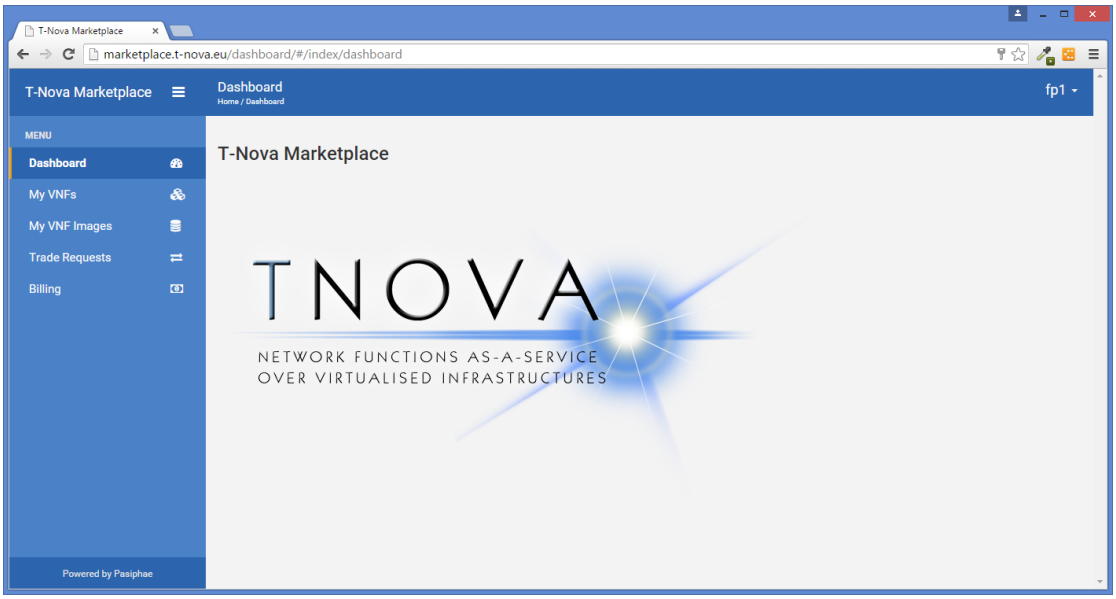

Figure 5-3 VNF Providers home page

5.3.1. VNF Creation

Here the function provider can create new VNFs.

5.3.1.1. Step 1 – VNF Basic Information

The First step of the VNF creation is to give basic information about the Network Function

- The name of the VNF
- A Description
- VNF Version
- Descriptor Version
- VNF Type (Traffic Classifier, Firewall, etc.)

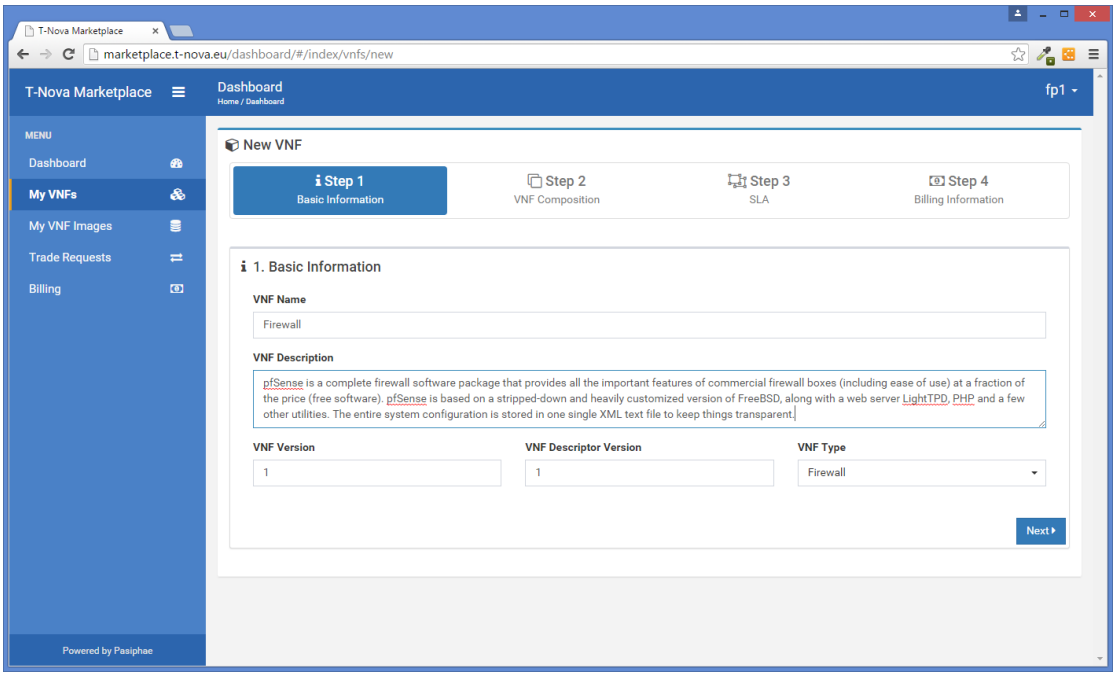

Figure 5-4 VNF Creation - Step 1: VNF Basic Information

5.3.1.2. Step 2 - VNF Composition

The second step of the VNF Creation is the VNF composition, here the provider can select the flavor that prefers (gold/silver/bronze) and create a composition of every flavor. Then the function provider can add a Virtual Machine (i.e. VDU) to the flavor, in this step the provider can define the required resources for the virtual machine as long as the image, the monitoring parameters and the lifecycle events. The last step is to define the virtual links that will be available in the composition in order to link individual virtual machines or give them external access.

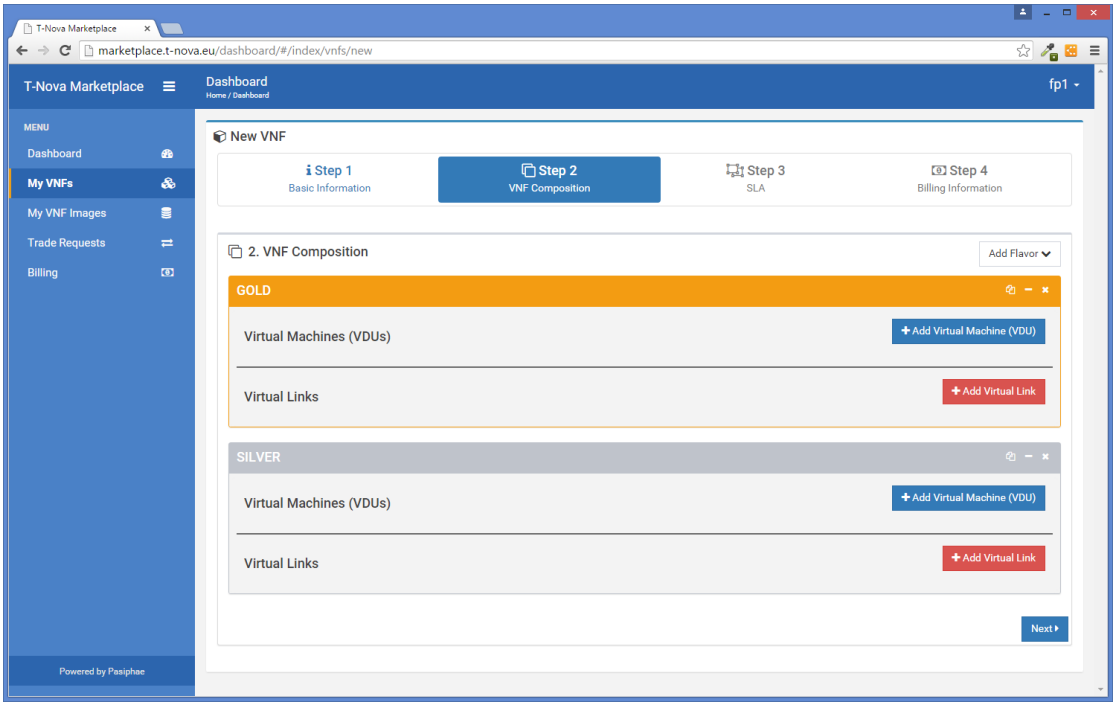

Figure 5-5 VNF Creation - Step 2: VNF Composition, SLA Flavors

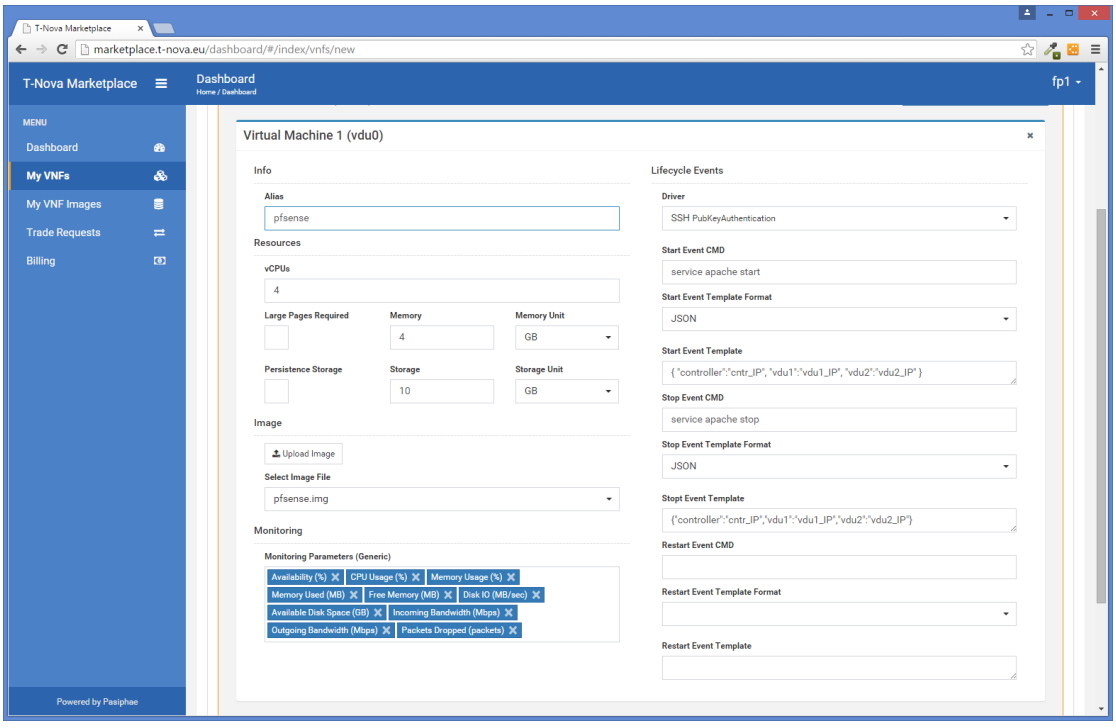

Figure 5-6 VNF Creation - Step 2: VNF Composition, Virtual Machines

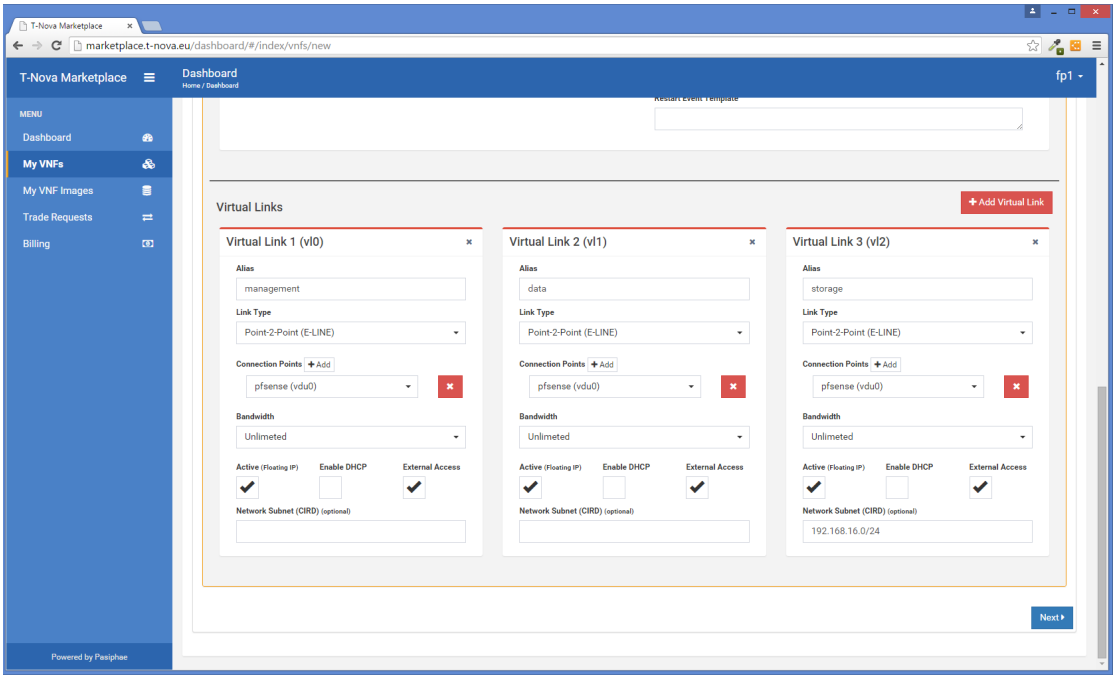

Figure 5-7 VNF Creation - Step 2: VNF Composition, Virtual Links

5.3.1.3. Step 3 – SLA

Here the provider defines the assurance parameters for every selected flavor. For example, the provider can apply a discount, if there is an SLA violation.

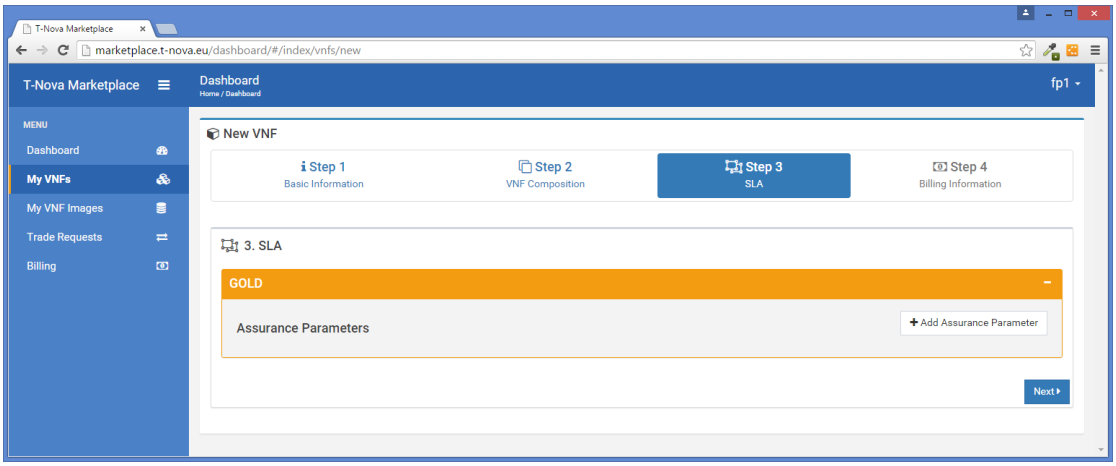

Figure 5-8 VNF Creation - Step 3: SLA

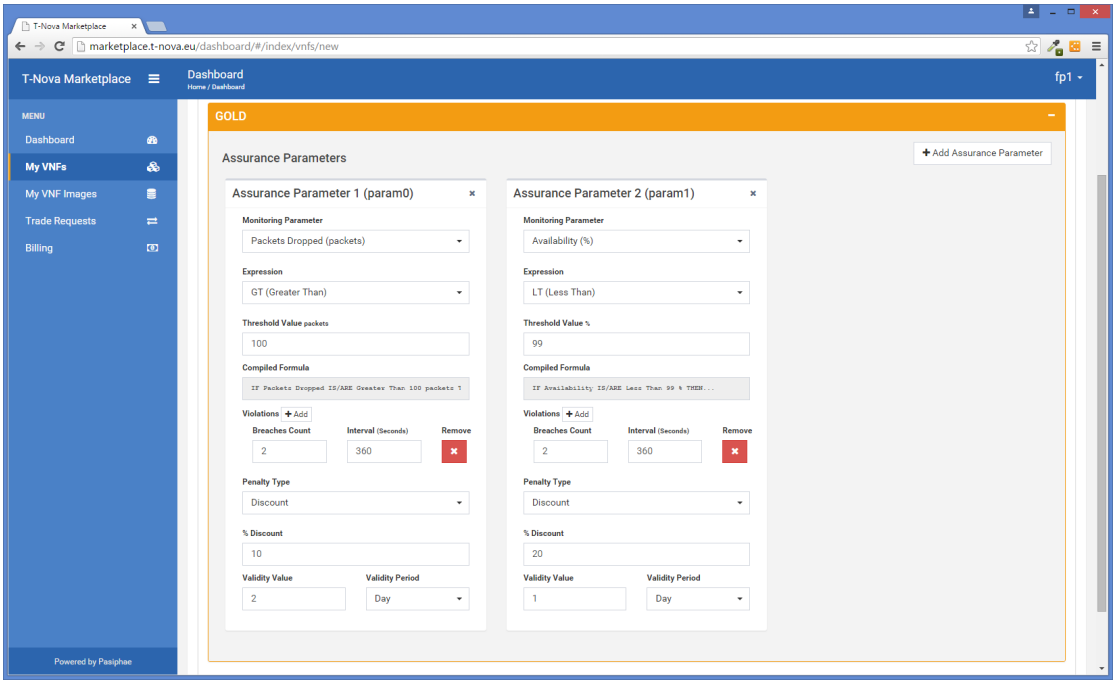

Figure 5-9 VNF Creation - Step 3: SLA, Assurance Parameters

5.3.1.4. Step 4 – Billing

In this final step, the provider can define the billing model, price, currency and to enable the trading (for the broker module).

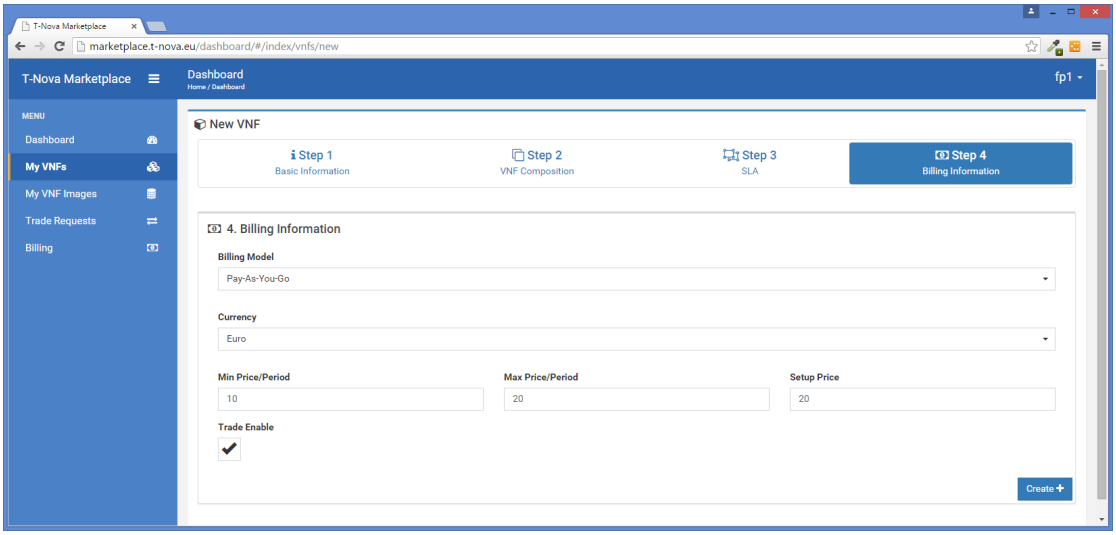

Figure 5-10 VNF Creation - Step 4: Billing

5.3.2. VNF Listing

Here the providers are able to manage their VNFs, view the generated VNFD file, edit the NFD with the YAML editor, view the composed VNF (vnf diagram) and delete a VNF.

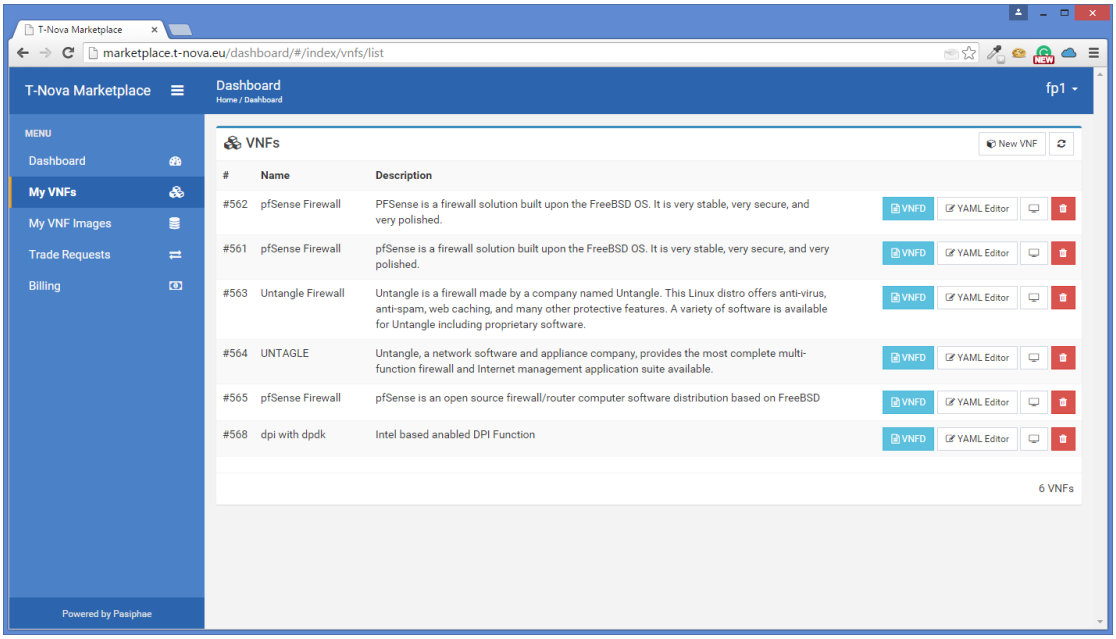

Figure 5-11 VNF Listing

5.3.3. VNF Tools

5.3.3.1. Generated VNFD Viewer

ETSI compliant Generated VNFD.

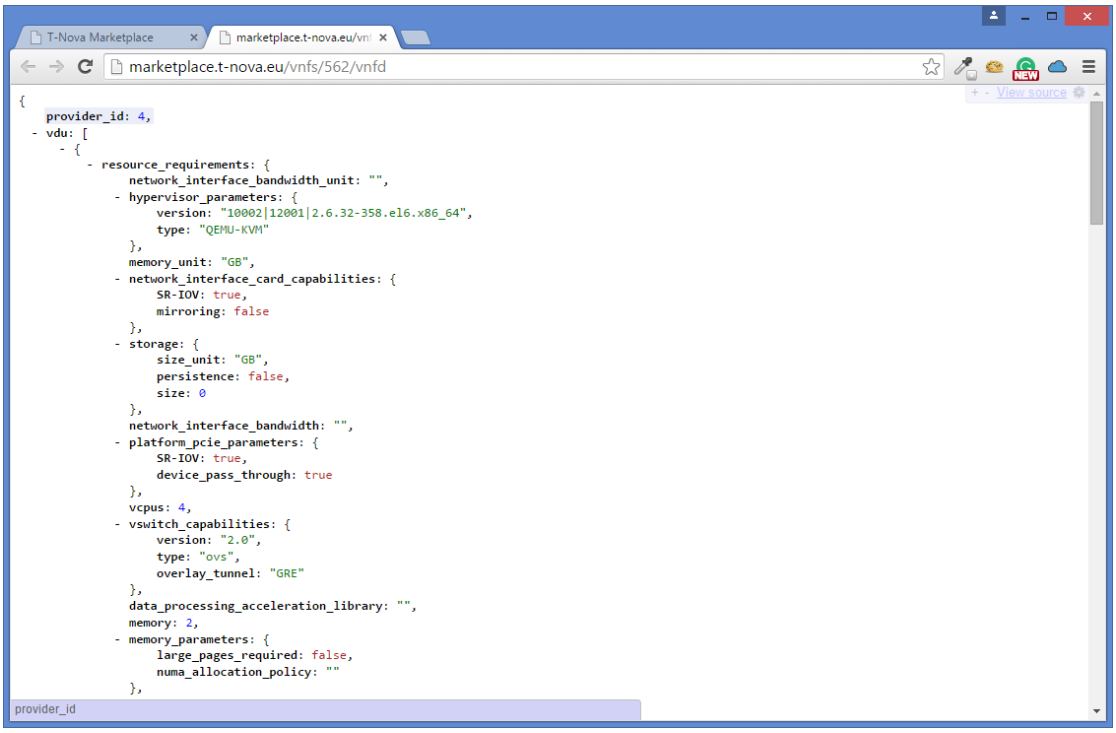

Figure 5-12 Genereted VNFD Viewer

5.3.3.2. VNFD YAML Editor

The provider can edit the generated VNFD with the YAML editor.

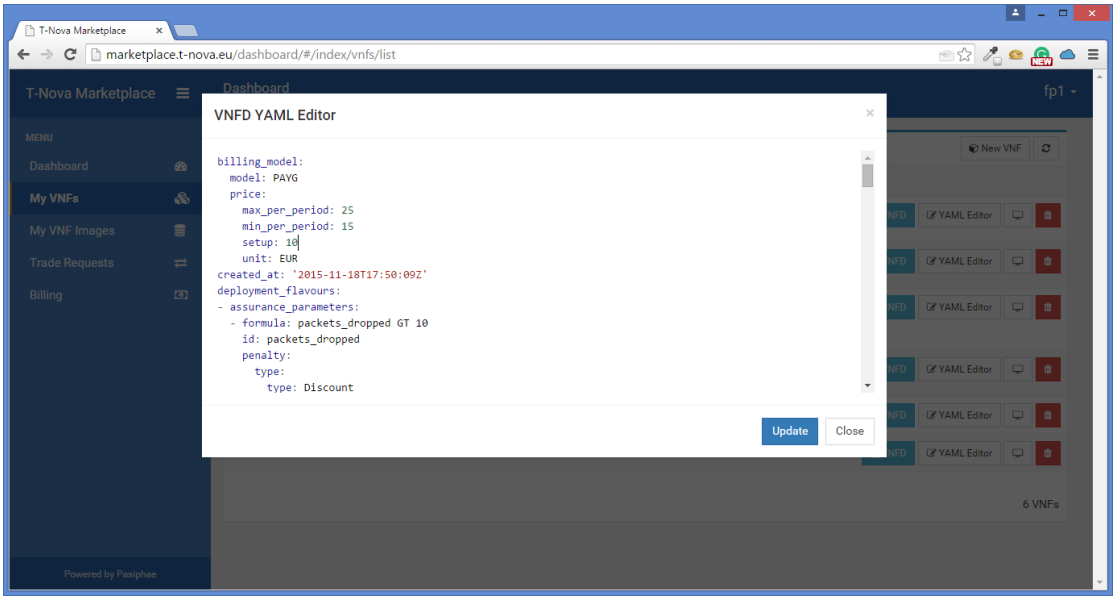

Figure 5-13 VNFD YAML Editor

5.3.3.3. VNFD Diagram

The provider can (visualize the VNFD compositions) view the diagram of the generated VNFD.

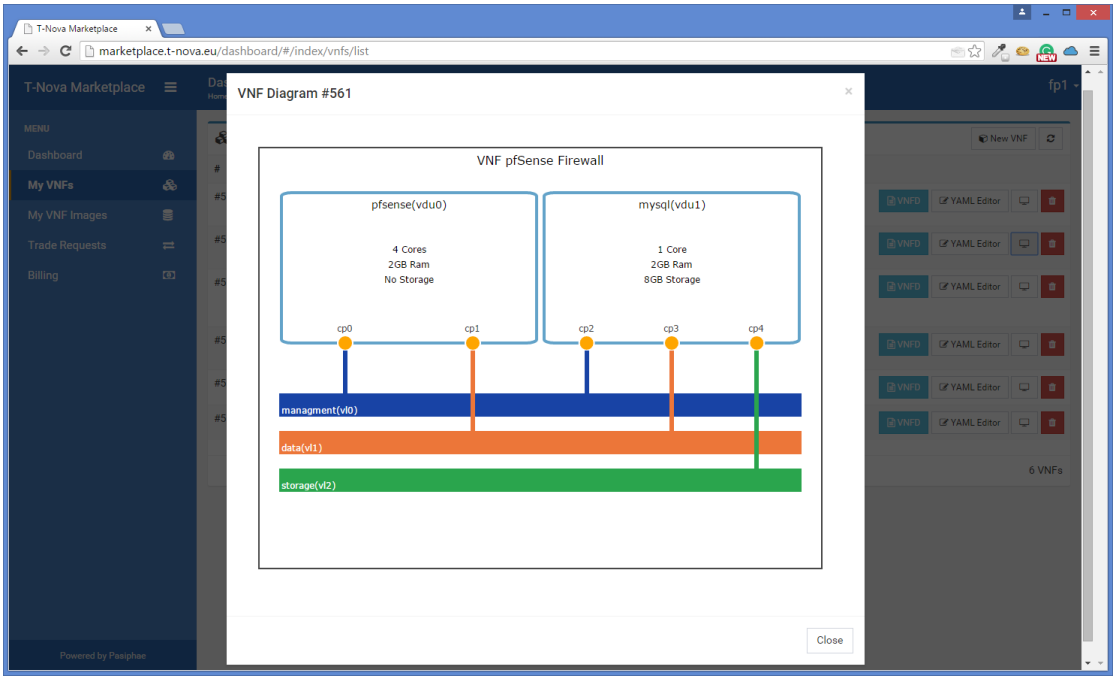

Figure 5-14 VNF Diagram

5.3.4. Images

From this page the function provider can manage the VNF images.

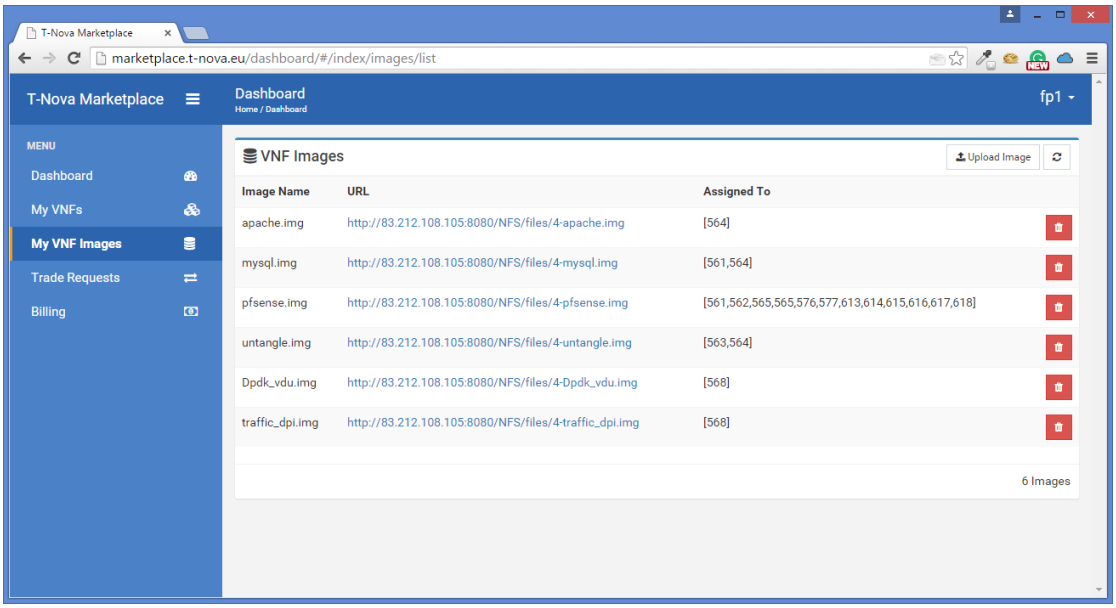

Figure 5-15 Function Provider VNF Images

5.4. Service Provider

This is the Service Provider's home page.

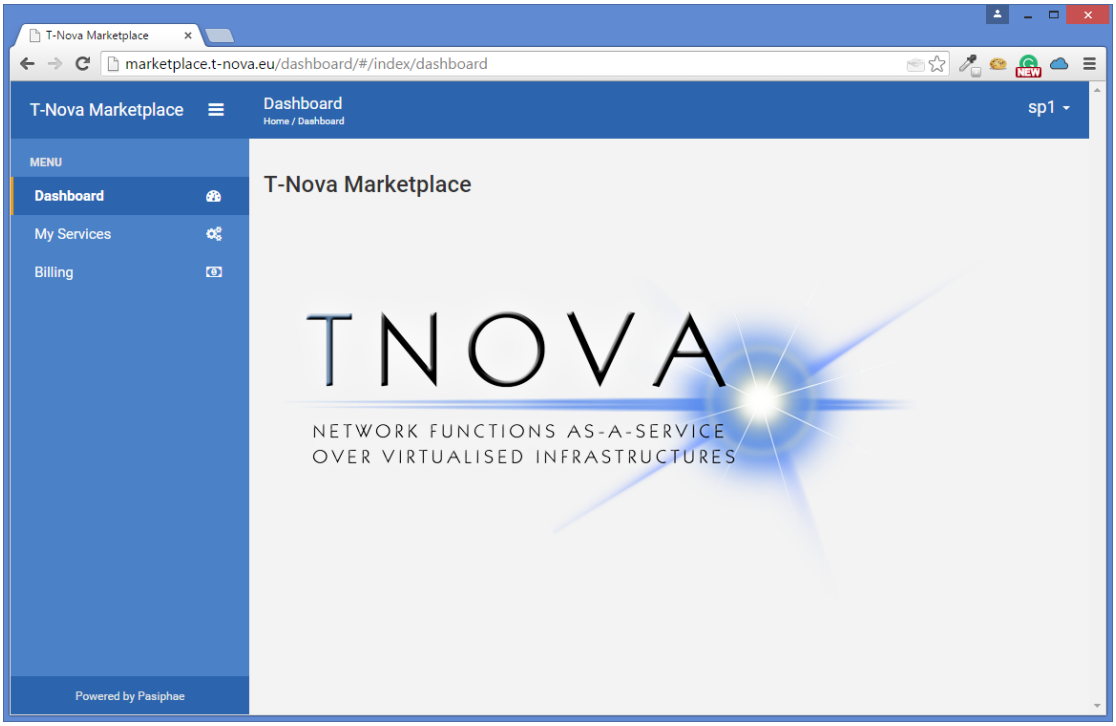

Figure 5-16 Service Provider

5.4.1. Service/NSD Creation

In the below pages, the service provide can compose/create a new service

5.4.1.1. Step 1 - VNF Selection

The first step of the NSD/service composition is the VNF Selection. The service provider selects the VNFs that would like to apply/add on this service.

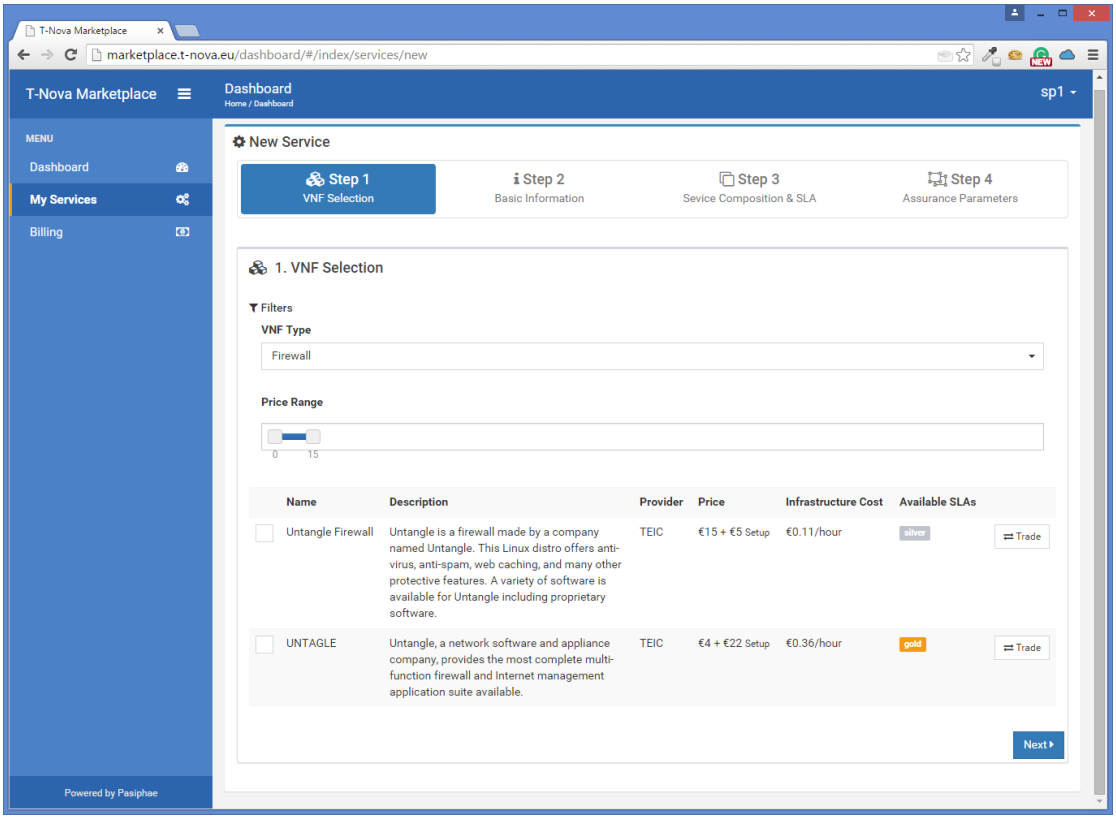

Figure 5-17 NSD Creation - Step 1: VNF Selection

5.4.1.2. Step 1 - VNF Trading

The service provider is able to negotiate with the function provider requesting a better price (Figure 5-18 Figure 5-19). The function provider can accept this request/offer or to reject it Figure 5-20). In **Error! Reference source not found.**, the Function provider accepts the request, so the new price will override the old price.

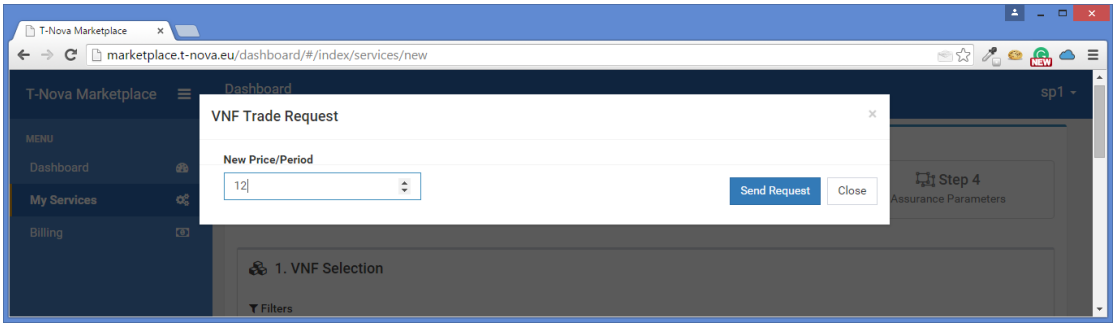

Figure 5-18 NSD Creation - Step 1: VNF Selection, Trade Request

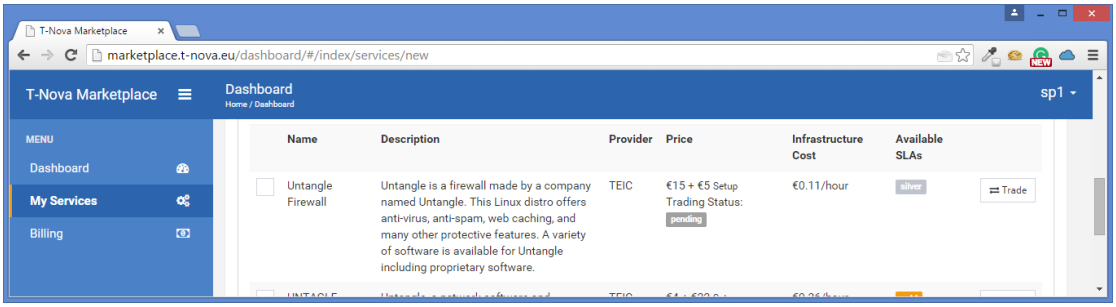

Figure 5-19 NSD Creation - Step 1: VNF Selection, Pending Trade Request

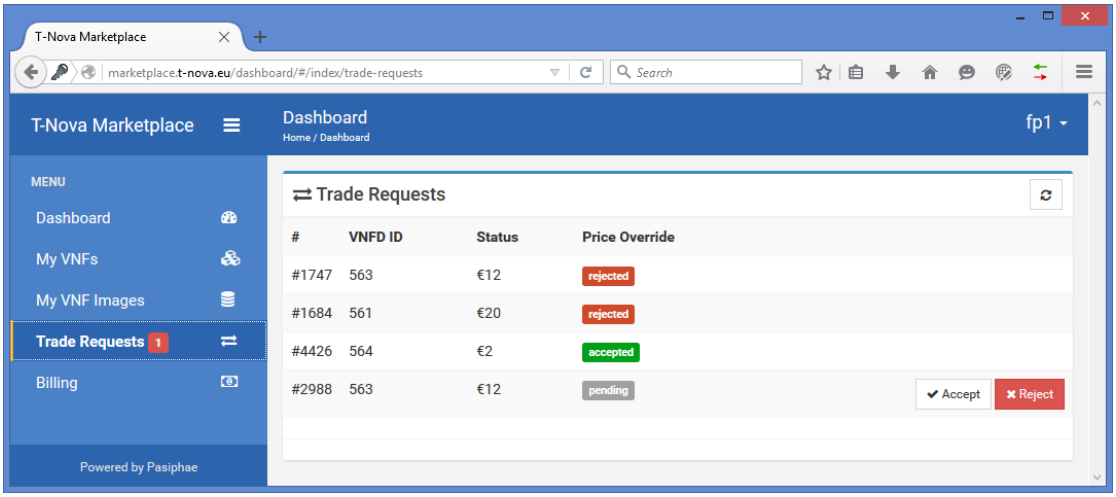

Figure 5-20 Trade Request FP View

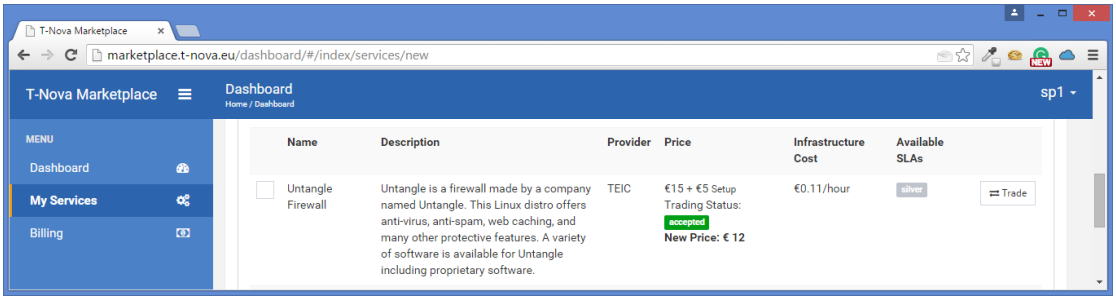

Figure 5-21 Accepted Trade Offer

5.4.1.3. Step 2 –Basic Information

The next step is the basic information of the Network service

- Service name
- A description
- NSD version
- Descriptor Version

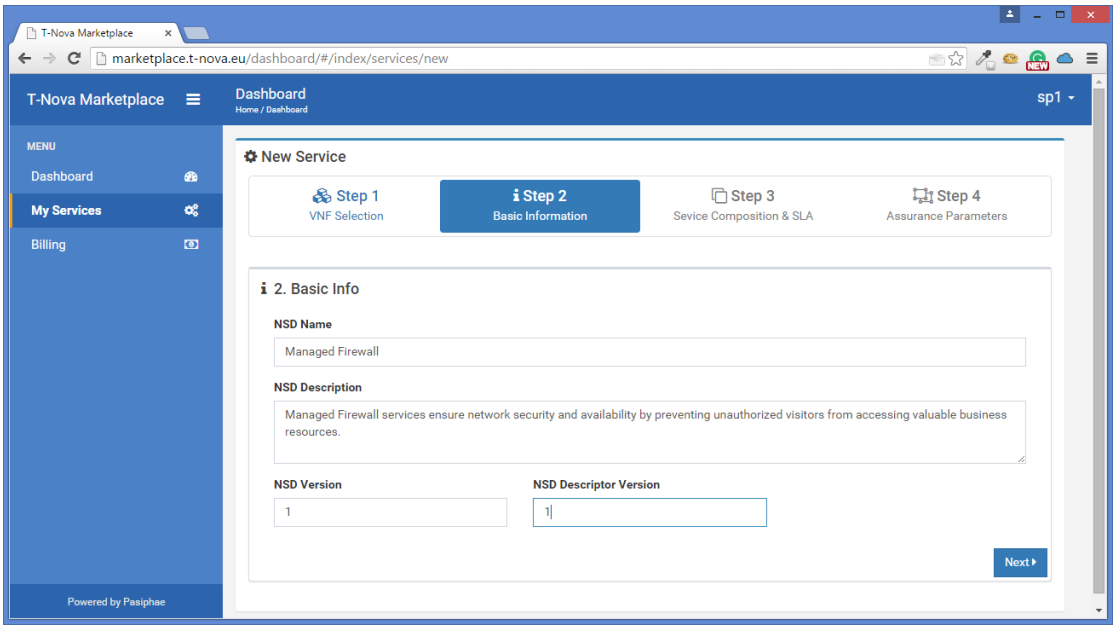

Figure 5-22 NSD Creation - Step 2: Basic Information

5.4.1.4. Step 3 – Service Composition and SLA

The next step is the service composition and the SLA specification. The Service Provider should specify SLA and add the constituent VNFs with the price and the billing model.

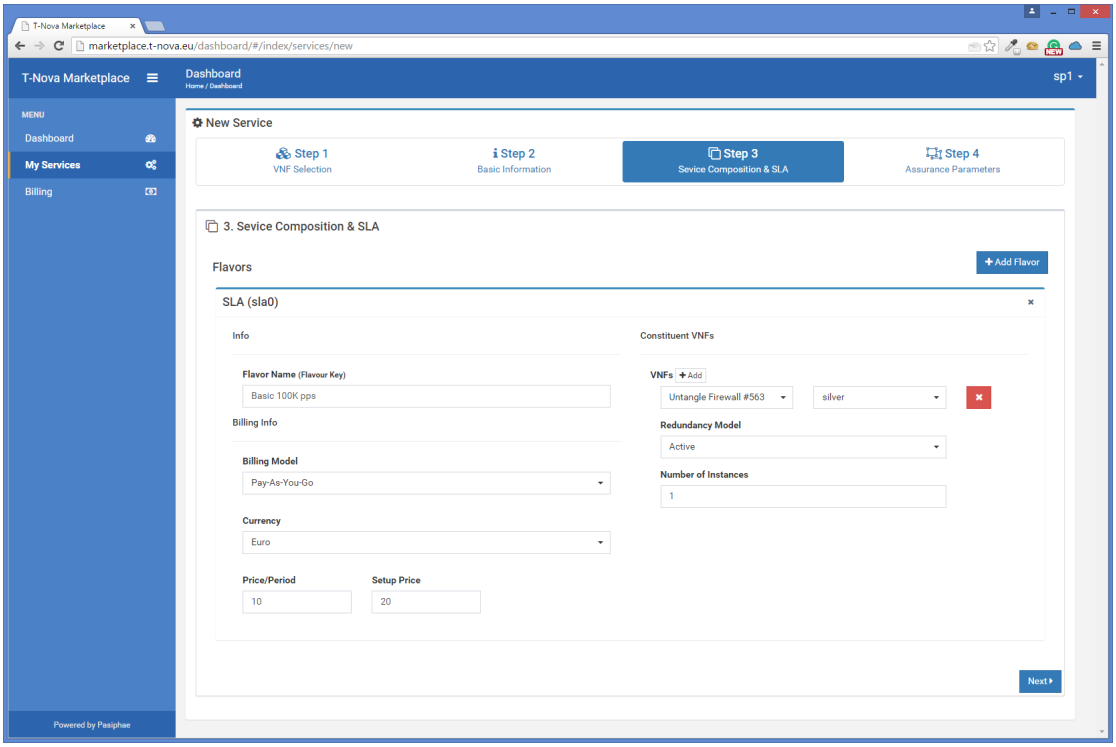

Figure 5-23 NSD Creation - Step 3: Service Composition and SLA

5.4.1.5. Step 4 – Assurance Parameters

The final step is to define the assurance parameters for the service for every available flavour; the provider can apply discount policies, if there is any SLA violation.

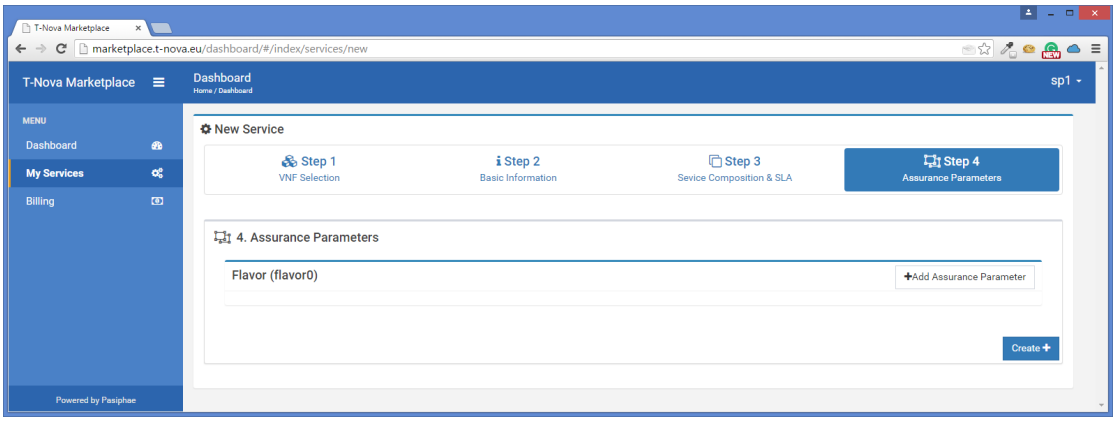

Figure 5-24 NSD Creation - Step 4: Assurance Parameters

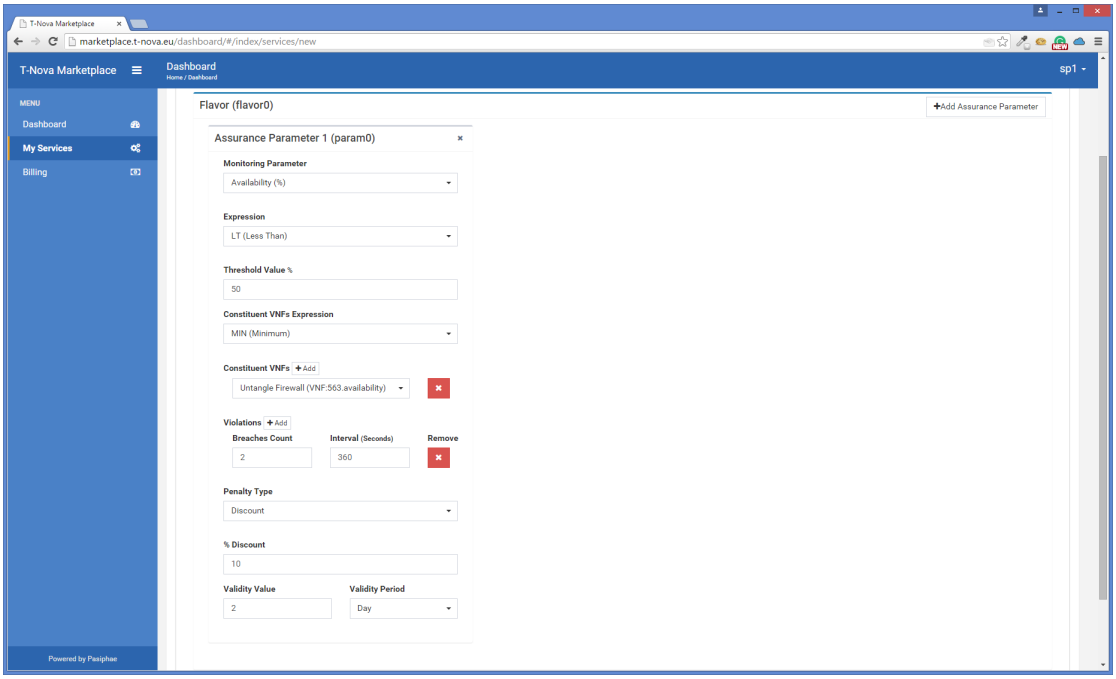

Figure 5-25 NSD Creation - Step 4: Assurance Parameters

5.4.2. NSD Listing

The Service provider can view and manage the created services, view the generated NSD or edit it with the NSD YAML editor or delete it.

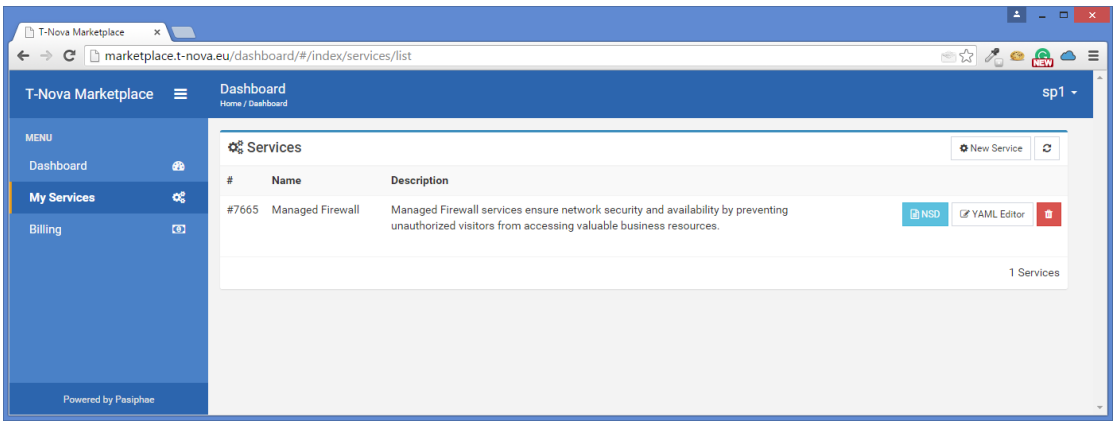

Figure 5-26 NSD Listing

5.4.3. NSD Tools

5.4.3.1. Generated NSD Viewer

The SP can view the generated NSD.

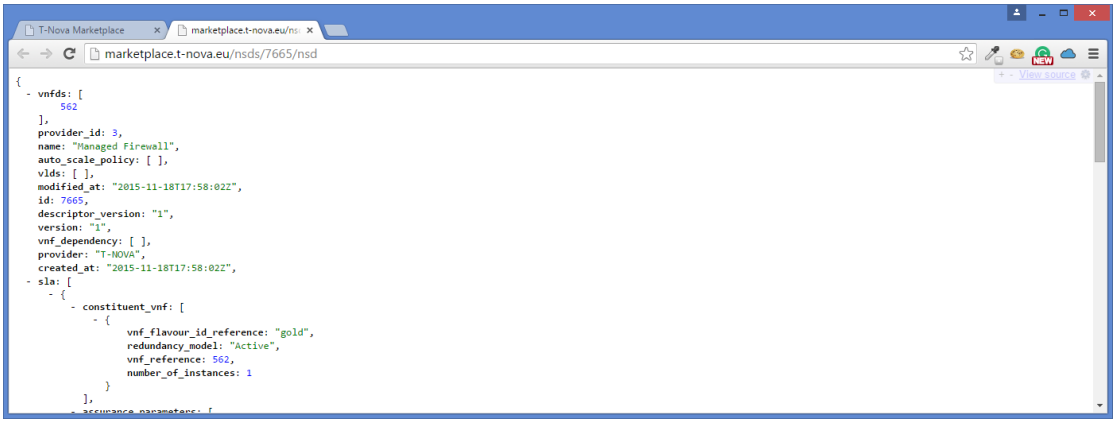

Figure 5-27 Generated NSD Viewer

5.4.3.2. NSD YAML Editor

The SP can edit the generated NSD with the NSD YAML Editor

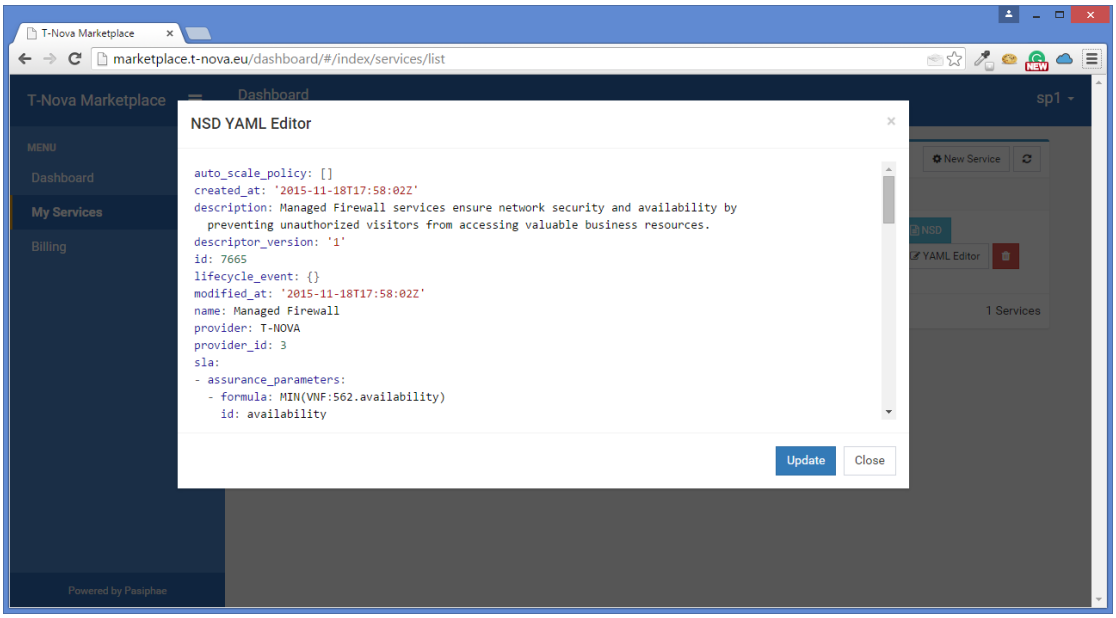

Figure 5-28 NSD YAML Editor

5.5. Customer

5.5.1. Customer – Service Selection

The Customer can browse the available services.

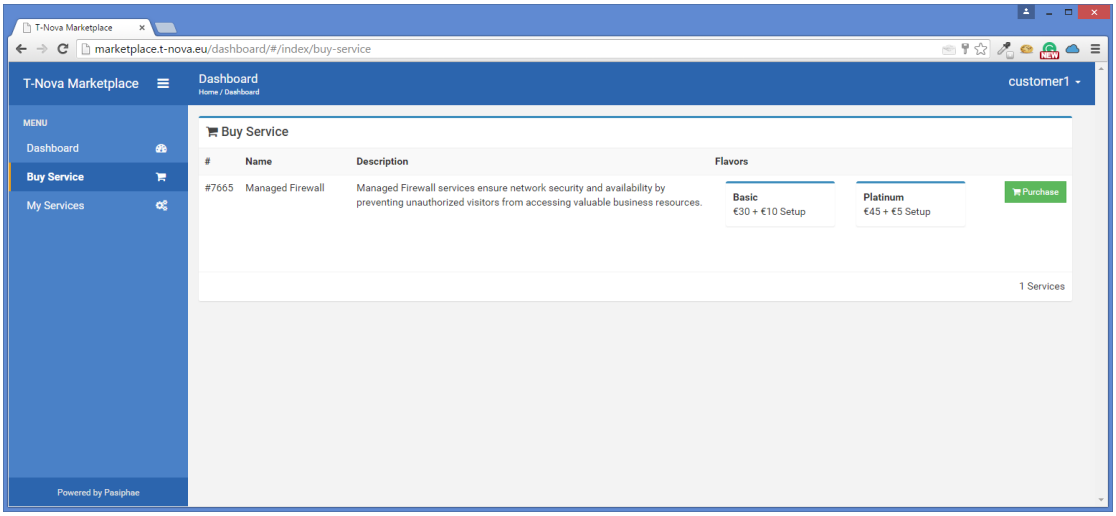

Figure 5-29 Customer – Service Selection

5.5.2. Customer - Service Purchase

The Customer can select the flavor of the service and purchase it.

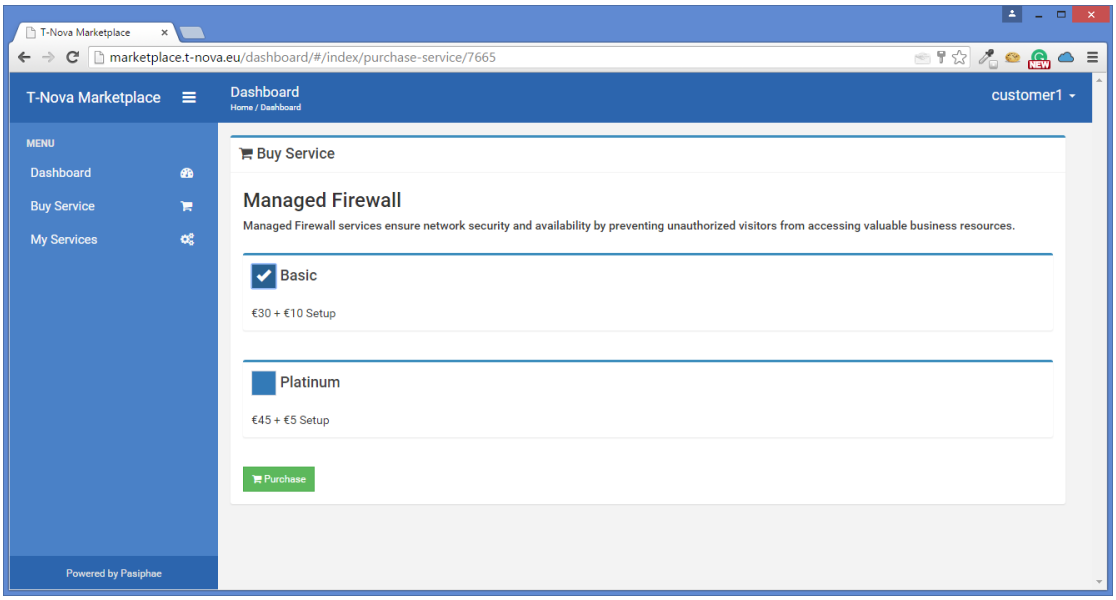

Figure 5-30 Customer - Service Purchase

5.5.3. Customer Services

The customer can view the running services.

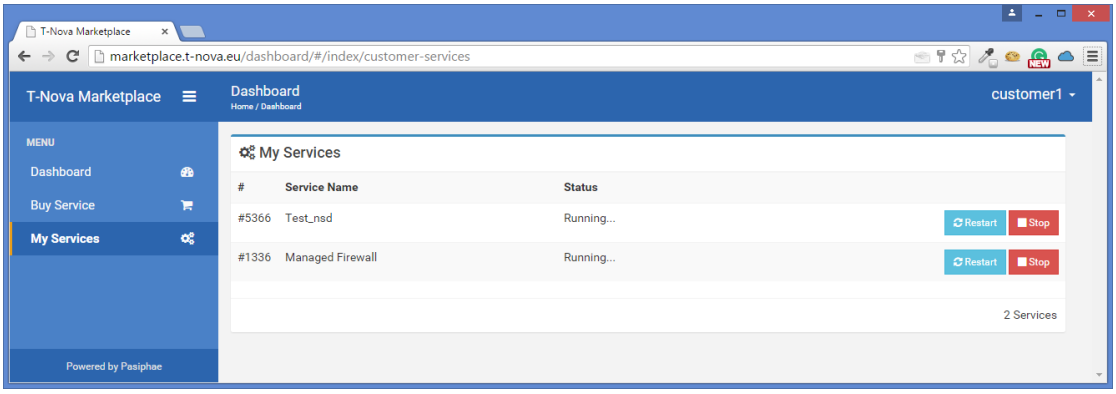

Figure 5-31 Customer Services

5.6. Administrator

This is the Dashboard's administrator home page.

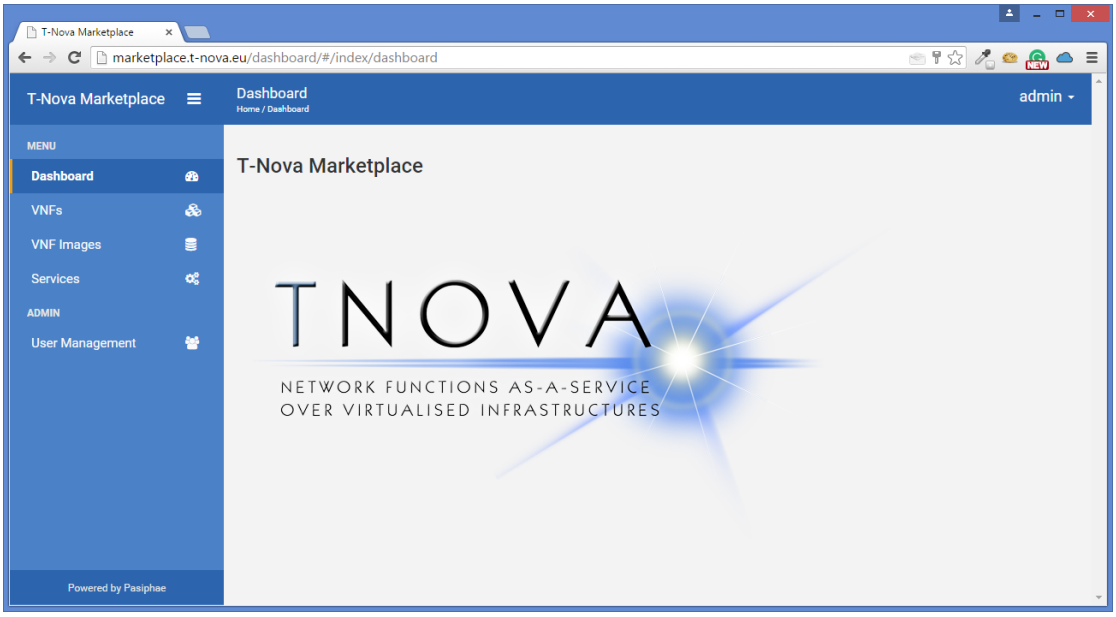

Figure 5-32 Administrator

5.6.1. User Management

The administrator can manage the user accounts.

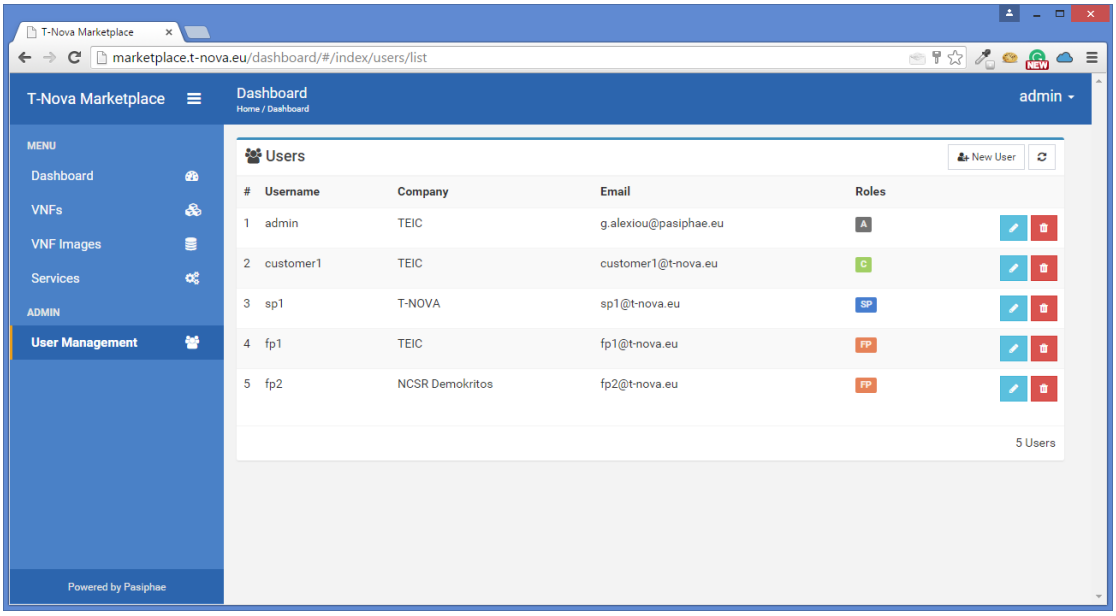

Figure 5-33 User Management

5.6.1.1. User Creation

Administrator can create a new user.

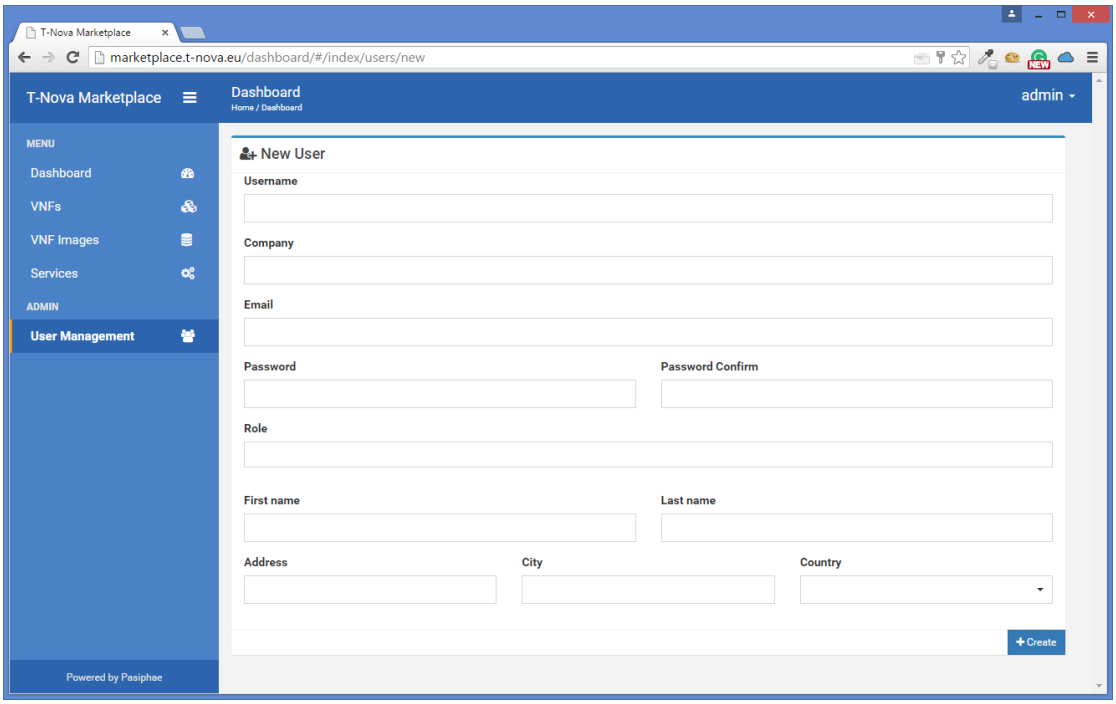

Figure 5-34 User Creation

5.6.1.2. Edit User Profile

The administrator can edit a user account.

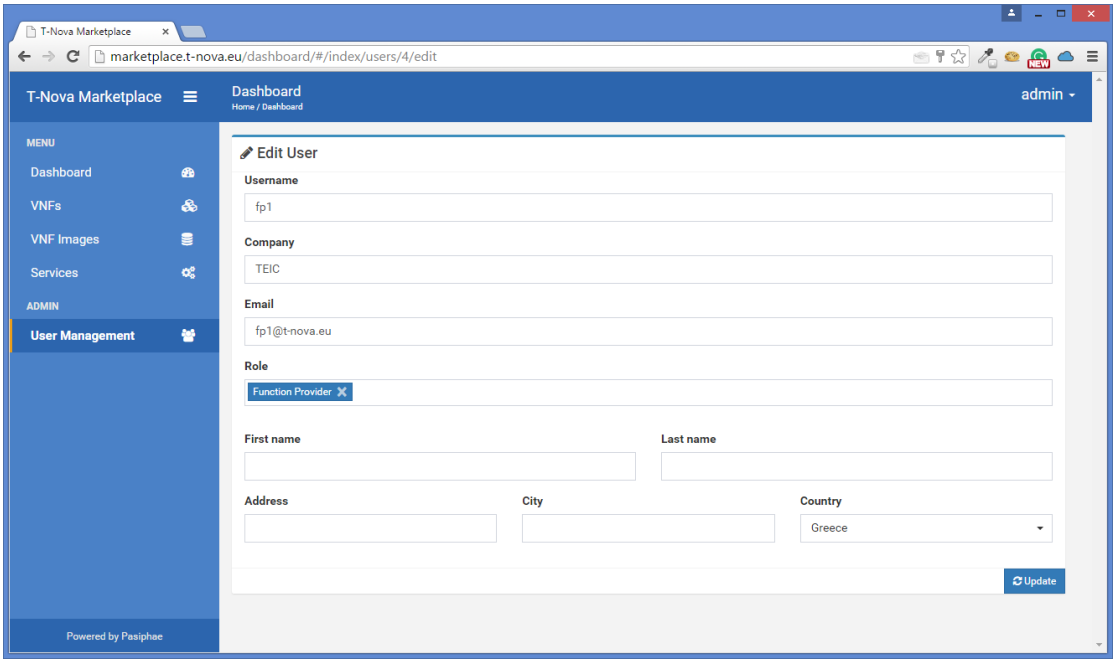

Figure 5-35 Edit User Profile

6. VALIDATION

6.1. Functional verification

To test the functionality of the Dashboard we have created some sample VNFs and Network Services with real data to make it as close to a real scenario as possible, trying to cover all the functionalities with one example.

#	Functionality	Action	Call from	Request to	Verification	Ok?
1	NSD storing	Store the recently created NSD in the BSC.	Dashboard	BSC	Verify the new NSD is present in the BSC database. Return code CREATED with the body including the autogenerated id after the call for NSD storage	Yes
2	Network services listing	List all available network services	Dashboard	BSC	Click on the NS catalogue and see the complete list on screen.	Yes
3	Filtered NS listing	List all available network services under a certain price	Dashboard	BSC	Set a maximum price, retrieve the filtered list of services and see all the network services in the list are below that price.	Yes
4	NSD modification	Select a NS from the BSC, modify any field	Dashboard	BSC	Verify the NSD has been updated in the BSC database.	Yes
5	NS deletion	Mark a NS for deletion from the BSC	Dashboard	BSC	Verify it has been deleted by retrieving again the complete list of services.	Yes
6	Providers registration	Register FP, SP and customer	Dashboard	SLA mgmt.	Verify the introduced providers are in the SLA management module database	Yes
7	VNF Templates introduction	Create VNF template based on the VNFD and send it to the SLA management module	Dashboard	SLA mgmt.	Check the SLA module database for the template and the logs to see there has not been errors in the process	Yes
8	Request bill	Request a bill generation for some user for a	Dashboard	Billing	The response contains the amount due for the customer.	Yes

		given time frame			The charge data records can be verified in the InfluxDB series.	
9	Request revenues report	Make a call to the billing module with the function provider ID and the time frame	Dashboard	Billing	The response contains the revenue share report along with any penalties for SLA violations. The periodic reports can be verified in the InfluxDB series.	Yes
10	Show SLA statistics	Request SLA statistics for a given NS or VNF, running or stopped	Dashboard	Accounting	See the charts drawn on the screen	Yes

Table 6-1 collects the results for this verification tests.

#	Functionality	Action	Call from	Request to	Verification	Ok?
1	NSD storing	Store the recently created NSD in the BSC.	Dashboard	BSC	Verify the new NSD is present in the BSC database. Return code CREATED with the body including the autogenerated id after the call for NSD storage	Yes
2	Network services listing	List all available network services	Dashboard	BSC	Click on the NS catalogue and see the complete list on screen.	Yes
3	Filtered NS listing	List all available network services under a certain price	Dashboard	BSC	Set a maximum price, retrieve the filtered list of services and see all the network services in the list are below that price.	Yes
4	NSD modification	Select a NS from the BSC, modify any field	Dashboard	BSC	Verify the NSD has been updated in the BSC database.	Yes
5	NS deletion	Mark a NS for deletion from the BSC	Dashboard	BSC	Verify it has been deleted by retrieving again the complete list of services.	Yes
6	Providers registration	Register FP, SP and customer	Dashboard	SLA mgmt.	Verify the introduced providers are in the SLA management module database	Yes

7	VNF Templates introduction	Create VNF template based on the VNFD and send it to the SLA management module	Dashboard	SLA mgmt.	Check the SLA module database for the template and the logs to see there has not been errors in the process	Yes
8	Request bill	Request a bill generation for some user for a given time frame	Dashboard	Billing	The response contains the amount due for the customer. The charge data records can be verified in the InfluxDB series.	Yes
9	Request revenues report	Make a call to the billing module with the function provider ID and the time frame	Dashboard	Billing	The response contains the revenue share report along with any penalties for SLA violations. The periodic reports can be verified in the InfluxDB series.	Yes
10	Show SLA statistics	Request SLA statistics for a given NS or VNF, running or stopped	Dashboard	Accounting	See the charts drawn on the screen	Yes

Table 6-1 Dashboard functional verification

6.2. Requirements fulfillment

The main design decision, derived from the requirements in D2.42, has been to provide a common dashboard with different customized views based on different roles. In this context, Table 6-2 holds the Dashboard's requirements that are common for the Service Providers, the Function Providers and the Customers, as well as its implementation status.

Req. id	Requirement Description	Justification of Requirement	Implementation Status	Implementation Justification
D.1	The dashboard SHALL provide a "login in" page for the different stakeholders to be authenticated	Stakeholders interacting with the T-NOVA system should be authenticated and authorised in order to be able to browse the Business Service Catalog, issue SLA requests, or upload NFVs	YES	All three Roles identified are implemented
D.2	The "login" page in the SHALL offer to the different stakeholders means to use (username, password, OpenID, Google API) for authentication	Stakeholders interacting with the T-NOVA system should be able to use different authentication techniques to access T-NOVA	Partially	Google API and OpenID are simple to implement and not needed for the Proof of Concept Implementation

Req. id	Requirement Description	Justification of Requirement	Implementation Status	Implementation Justification
D.3	The "login" page in the SHALL offer to the different stakeholders means to remember credentials when logging on	Stakeholders interacting with the T-NOVA system should not be obliged to insert credentials when accessing again the system	Yes	This is done through the Browser
D.4	The Dashboard SHALL be accessible to authorized users via the Internet	The Dashboard will provide the necessary interface in order to be viewed over the Internet	YES	The Dashboard is available on a public IP
D.5	The Dashboard SHOULD provide multiple users login and no less than 10	The Parallel access will provide the necessary tools for every user to be able to provide his content	YES	The dashboard supports multiple login from multiple roles
D.6	The Dashboard SHOULD be available 24/7 365 days per year	The Dashboard must be always on in order to control every part of the T-NOVA infrastructure.	Partially	This is an implementation guideline for the real environment
D.7	The Dashboard MUST be as light way as possible in the Server side	The Dashboard must be able to take into account the processing power of the stakeholder accessing it.	YES	The use of the technologies identified in this deliverable constitutes the Dashboard a Ultra-Light Web page.
D.8	The Dashboard MUST be Open source	The Dashboard must be Open source	YES	Dashboard will be provided open source with a Open license that needs to be defined

Table 6-2 Dashboard basic requirements

7. CONCLUSIONS

This deliverable presented the design, implementation and operation of T-NOVA's users Dashboard. More specifically, it provided an overview of the available web development frameworks used in frontend, middleware and backend deployments of web applications. Based on factors such as community support, maturity and extensibility, Bootstrap, AngularJS and Django were chosen as frontend, middleware and backend web development frameworks respectively. In addition it, discussed the existing software development approaches for developing a web application, namely the Server-Centric Web Application (SCWA) and Browser-Centric Web Application (SCWA), focusing on the latter because of its advantages in scalability. In the same context, it examined the monolithic and micro-services deployment approaches and proposed the adoption of micro-services architecture enhanced with the Docker paradigm; this innovative approach allowed for faster development cycles, interoperability between different software development frameworks and, most important, seamless-transparent deployment and execution of the Dashboard in any physical or virtual OS, without having to worry about libraries and third party tools incompatibilities. Following this, the implementation of the Dashboard was presented in the form of code-snippets screenshots. Finally, a detailed description of the APIs between the users' Dashboard and the rest T-NOVA management modules was given followed by the demonstration of the Dashboard's operation.

7.1.

8. REFERENCES

[1] Semantic-UI, http://semantic-ui.com/introduction/new.html, Last Access 10/2015

[2] BootStrap, http://getbootstrap.com/, Last Access 10/2015

[3] AngularJS, https://angularjs.org/, Last Access 10/2015

[4] JQuery, https://jquery.com/, Last Access 10/2015

[5] Ember, http://emberjs.com/, Last Access 10/2015

[6] React, http://facebook.github.io/react/, Last Access 10/2015

[7] Django: The Web framework for perfectionists with deadlines, https://www.djangoproject.com/, Last Access 10/2015

[8] Flask (A Python Microframework), http://flask.pocoo.org/, Last Access 10/2015

[9] Symfony, High Performance PHP Framework for Web Development, https://symfony.com/, Last Access 10/2015

[10] Yii PHP Framework: Best for Web 2.0 Development, http://www.yiiframework.com/, Last Access 10/2015

[11] Docker – Build, Ship, and Run Any App, Anywhere, https://www.docker.com/, Last Access 11/2015

9. ANNEX A: