Secure Provisioning of Cloud Services based on SLA Management

SPECS Project - Deliverable 1.4.2

Module Shared API and Core Services

Version 1.1
15 February 2016

The activities reported in this deliverable are partially supported by the European Community's Seventh Framework Programme under grant agreement no. 610795.

Deliverable information

Deliverable no.:	D1.4.2
Deliverable title:	Module Shared API and Core Services
Deliverable nature:	Prototype
Dissemination level:	Public
Contractual delivery:	31 October 2015
Actual delivery date:	31 October 2015
Author(s):	Massimiliano Rak (CeRICT)
Contributors:	Giancarlo Capone (CeRICT), Silviu Panica (IeAT), Damjan Murn (XLAB), Valentina Casola (CeRICT), Nicola De Filippo (CeRICT)
Reviewers:	Umberto Villano (CeRICT), Jolanda Modic (XLAB)
Task contributing to the deliverable:	T1.4
Total number of pages:	86

Executive summary

This deliverable provides the supporting documentation for the prototypes demonstrating the SLA Platform module and two of the components of the Vertical layer, namely the Auditing component and the User Manager component, whose final design is presented in D1.4.1.
The SLA Platform is in charge of managing every aspect of the SLA life cycle and has to guarantee interoperability among all Core modules, namely Enforcement, Negotiation and Monitoring. In particular, it exposes a high-level and service-oriented API to manage the SLA life cycle. Each operation on an SLA is performed by invoking the appropriate service, abstracting on common issues, like concurrency control or data storage and representation. Furthermore, it exposes a set of high-level functionalities to the SPECS Core modules, as it is responsible for the communication among Enforcement, Negotiation and Monitoring modules in a centralized manner, allowing for the decoupling of the design and of the implementation of such modules.
As described in deliverables D1.1.3 and D1.4.1, in order to cover the large number of requirements and to cope with the feedbacks from implementation and validation activities, the SLA Platform design has been updated. The final module architecture includes four components, i.e., SLA Manager, Service Manager, Security Metrics Catalogue and Interoperability layer, and a number of different artifacts, mainly models, to cope with the definition of common data models and APIs.

The Vertical layer provides some cross-cutting functionalities used by the SLA Platform and by other modules. In particular, the Auditing component and the User Manager component described in this deliverable are responsible for supporting auditing in all the steps of the SPECS flow and for managing platform's user, respectively. As said in deliverable D1.4.1, the other two components of the Vertical layer (i.e., Credential Service and Security Tokens) are described in WP4 deliverables (i.e., D4.4.1 and D4.4.2).

In this document, we present the status of the development activities related to T1.4, and show how to install and use the prototype components developed in the framework of this task. Such components are all currently available, and links to repositories are provided in this document. Moreover, in accordance with the methodology proposed in D4.5.2, unit test results are presented for all components.
The components reported in the following cover a very large number of requirements and they are at the core of the SPECS solution, although most of them are transparent to the SPECS End-users. The analysis of elicited requirements and motivations behind the choice of the presented solution are discussed in deliverables D1.1.2, D1.1.3 and D1.4.1.

Table of contents

Deliverable information ... 2
Executive summary ... 3
Table of contents ... 4
Index of figures ... 6
Index of tables ... 7
1. Introduction ... 8
2. Relationship with other deliverables ... 9
3. The SPECS SLA Platform ... 10
 3.1. Status of Development Activities ... 10
 3.2. Repository ... 12
 3.3. Installation ... 12
  3.3.1. Requirements ... 13
  3.3.2. Starting a SPECS SLA Platform instance ... 13
   3.3.2.1. Selecting the Provider ... 13
   3.3.2.2. Describing the cluster configuration ... 14
   3.3.2.3. Deployment configuration ... 14
   3.3.2.4. Starting the cluster ... 15
  3.3.3. Controlling the SPECS SLA Platform instances ... 15
  3.3.4. Accessing the SPECS SLA Platform ... 16
4. SPECS SLA Platform Components ... 17
 4.1. SLA Manager ... 17
  4.1.1. Repository ... 17
  4.1.2. Description and design ... 17
  4.1.3. Installation ... 17
   4.1.3.1. Installing by using precompiled binaries ... 17
   4.1.3.2. Compiling and installing from source ... 18
  4.1.4. Usage ... 18
  4.1.5. Tests ... 20
 4.2. Service Manager ... 21
  4.2.1. Repository ... 21
  4.2.2. Description and design ... 22
  4.2.3. Installation ... 22
   4.2.3.1. Installing by using precompiled binaries ... 22
   4.2.3.2. Compiling and installing from source ... 23
  4.2.4. Usage ... 23
  4.2.5. Tests ... 25
 4.3. Security Metrics Catalogue ... 27
  4.3.1. Repository ... 28
  4.3.2. Description and design ... 28
  4.3.3. Installation ... 28
   4.3.3.1. Installing by using precompiled binaries ... 28
   4.3.3.2. Compiling and installing from source ... 29
  4.3.4. Usage ... 29
  4.3.5. Tests ... 31
 4.4. Interoperability Layer ... 33
  4.4.1. Repository ... 33

4.4.2. Description and Design 34
4.4.3. Installation 34
4.4.3.1. Installing by using precompiled binaries 34
4.4.3.2. Compiling and installing from source 34
4.4.4. Usage 35
4.4.5. Tests 35
Vertical Layer Components 37
4.5. Auditing 37
4.5.1. Repository 37
4.5.2. Description and Design 37
4.5.3. Installation 37
4.5.3.1. Audit Server 37
4.5.3.2. Audit Client 38
4.5.4. Usage 38
4.5.5. Tests 39
4.6. User Manager 40
4.6.1. Repository 40
4.6.2. Description and Design 41
4.6.3. Installation 41
4.6.3.1. Installing by using precompiled binaries 41
4.6.3.2. Compiling and installing from source 41
4.6.4. Usage 43
4.6.5. Tests 44
5. Conclusions 45
6. References 46
Annex A 47
A.1. SLA Manager Tests 48
A.2. Service Manager Tests 57
A.3. Security Metrics CatalogueTests 63
A.4. Interoperability Layer Tests 79
A.5. Auditing Tests 85

Index of figures

Figure 1: Relationship with other deliverables 9
Figure 2: SPECS platform setup process 13
Figure 3: SPECS Launcher. Web User Interface 14
Figure 4: SPECS Launcher. Sessions interface 15
Figure 5: SLA Manager – *Get a collection of SLAs* API call example 19
Figure 6: SLA Manager – *Get a single SLA* API call example 19
Figure 7: Code Quality Analysis Report for SLA Manager Backend 20
Figure 8: Code Quality Analysis Report for SLA Manager Frontend 21
Figure 9: Service Manager - *Get all SMs* API call example 24
Figure 10: Service Manager - *Get SM Metadata* API call example 24
Figure 11: Service Manager - *Get all SCs* API call example 25
Figure 12: Code Quality Analysis Report for Service Manager Backend 26
Figure 13: Code Quality Analysis Report for Service Manager Frontend 27
Figure 14: Security Metrics Catalogue - *Get all metrics* API call example 30
Figure 15: Security Metrics Catalogue - *Get a specific metric* API call example 31
Figure 16: Code Quality Analysis Report for Metric Catalogue Backend 32
Figure 17: Code Quality Analysis Report for Metric Catalogue Frontend 33
Figure 18: Code Quality Analysis Report for Interoperability 36
Figure 19: Code Quality Analysis Report for Audit 40
Figure 20: User Manager – The landing page 43
Figure 21: User Manager - User allowed resources WUI 44
Figure 22: User Manager - New account WUI 44

Index of tables

Table 1: SLA Platform components requirements coverage 11
Table 2: Vertical Layers components requirements coverage 11
Table 3: SLA Platform module and Vertical layer module available artifacts 12

1. Introduction

This deliverable provides the supporting documentation for the prototypes demonstrating the SLA Platform and part of the Vertical layer.
Based on the final design, reported in D1.1.3 and D1.4.1, the components belonging to the SLA Platform are the SLA Manager, the Service Manager, the Security Metrics Catalogue and the Interoperability Layer. For what regards the Vertical layer, it comprises the Auditing component, the User Manager component, the Credential Service component and the Security Tokens component. As specified in D1.4.1, only the Auditing and User Manager components have been analysed in Task 1.4, while the other two components are dealt with in WP4 and discussed in D4.2.2 and D4.4.2.
For each of the above components, we report the source repository where the component code is available, the installation steps, the usage guides and the testing results. Moreover, we also provide a guide to set-up the whole SLA Platform on the SPECS Testbed by exploiting the Launcher interface provided by the Enabling Platform and described in D1.6.1. This guide is useful for those non-expert End-users (e.g., SPECS Owners) that do not want to set-up all the components separately, and enables to have the SLA Platform up and running in few steps.

The deliverable is organized as follows. In Section 2, the relationship with other deliverables is reported. In Section 3, the status of development activities related to the SLA Platform is illustrated, and the guide for the set-up of the SLA Platform by means of the Launcher is presented. In Section 4 and Section 5 the details on implementation, installation, usage and testing of the single components belonging to the SLA Platform and to the Vertical layer respectively are given. In particular:

- Section 4.1 discusses the SLA Manager, devoted to handling the SLA life cycle; its main aim is to store all the SLAs that have been signed by End-users and to enable their management;
- Section 4.2 discusses the Service Manager, devoted to storing and managing all the information about the Security Mechanisms and Security Capabilities;
- Section 4.3 discusses the Security Metrics Catalogue, devoted to storing and managing all the information about the Security Metrics;
- Section 4.4 discusses the Interoperability layer, devoted to offering functionalities for enabling a transparent communication among different modules;
- Section 4.5 discusses the Auditing component, devoted to offering logging and audit services to all other components and modules of the SPECS framework;
- Section 5.2 discusses the User Manager, which provides an access control mechanism that can be integrated in all SPECS applications.

Finally, Section 6 reports our conclusions, while all the details on testing activities and results are reported in Annex A.

2. Relationship with other deliverables

As illustrated in Figure 1, this document takes as primary input the Deliverable D1.4.1, where the refined architecture of the SLA Platform and of the Vertical layer is presented. The design and implementation choices are also strictly connected with deliverables D1.1.2, D1.3 and D4.4.2, where specific components and offered APIs are presented.
Furthermore, the testing methodology proposed in D4.5.2 has been chosen for designing the testing campaign of all SLA Platform components.

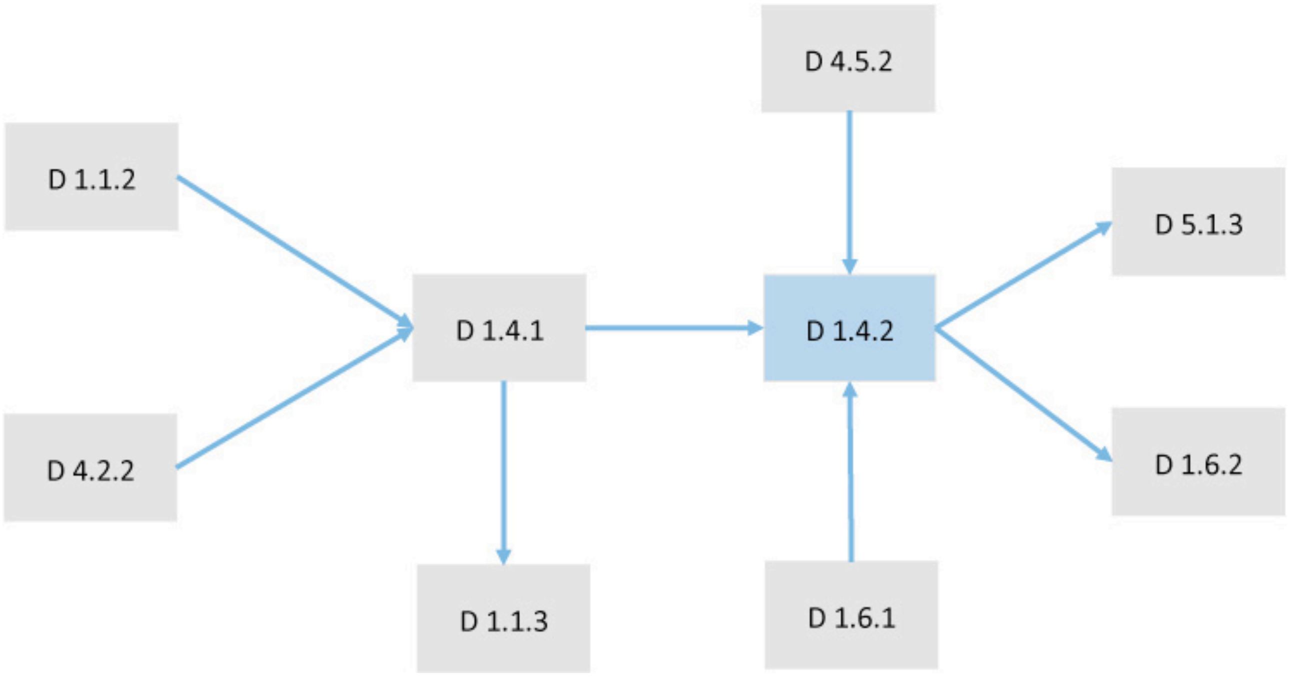


Figure 1: Relationship with other deliverables

The current implementation is input for the deliverable associated with the Case Studies, namely D5.1.3 (and to industrial applications, too), and will be used as input to the final design of the Enabling Platform for the development of the SPECS Testbed that will be reported in D1.6.2.

3. The SPECS SLA Platform

In this section, the status of development activities of the SLA Platform is presented, and the guide for the set-up of the SLA Platform by means of the Launcher interface (Section 3.3.2) of the Enabling Platform is presented.

3.1. Status of Development Activities

Table 1 and Table 2 schematically report the list of SPECS software components under development that are associated with the SLA Platform module and to the Vertical Layer module respectively, together with the requirements they cover.

SLA Platform Module	***SPECS Software components***			
Requirements	***SLA Manager***	***Service Manager***	***Interoperability Layer***	***Security Metrics Catalogue***
SLAPL_R1		X		
SLAPL_R2	DEPRECATED			
SLAPL_R3	Covered by other module (Monitoring- Archiver)			
SLAPL_R4				
SLAPL_R5		X		X
SLAPL_R6		X		X
SLAPL_R7		X		X
SLAPL_R8		X		X
SLAPL_R9		X		X
SLAPL_R10	X			
SLAPL_R11	X			
SLAPL_R12	X			
SLAPL_R13	X			
SLAPL_R14	X			
SLAPL_R15	Covered by other modules (Monitoring- Archiver and CTP Exporter)			
SLAPL_R16	Covered by other modules (Monitoring- Archiver)			
SLAPL_R17	DEPRECATED			
SLAPL_R18	Covered by Vertical Layer (User Manager)			
SLAPL_R19	X			
SLAPL_R20	X			
SLAPL_R21	X			
SLAPL_R22	X			
SLAPL_R23	X (Not yet)			
SLAPL_R24	X			
SLAPL_R25	X			
SLAPL_R26	X			
SLAPL_R27	X			
SLAPL_R28	X			
SLAPL_R29	X			
SLAPL_R30	X			
SLAPL_R31	X			
SLAPL_R32	X			
SLAPL_R33	X			
SLAPL_R34	X			

SLAPL_R45			X	
CERT_R1	X			
CERT_R2	X			
CERT_R3	X			

Table 1: SLA Platform components requirements coverage

As for the SLA Manager, this is one of the most important components of the whole SPECS architecture. There are 24 requirements associated with it, of which 1 (SLAPL_R23) is not covered yet and 3 (CERT_R1-R3) are not functional and mainly related to legal and certification issues. We can consider them covered, but with the limitations discussed in D1.1.2. The current implementation covers 99% of the SLA Manager requirements.
For what regards the Service Manager component, the Interoperability layer and the Security Metrics Catalogue, as shown in Table 1, there are few requirements associated with them and they are all covered (100%).

For what regards the Auditing component, as shown in Table 2, there are 14 requirements associated with it, 3 of which are not yet covered in the current implementation (current coverage is 79%). Similarly, as for the User Manager component, there are 9 requirements associated with it, 4 of which are not yet covered in the current implementation (current coverage is 55%). These are related to management of user and policy repositories and do not affect the efficiency of the prototype components.

SLA Platform Vertical Layers	***SPECS Software components***	
Requirements	***Auditing***	***User Manager***
ENF_PLAN_R6	X	
ENF_IMPL_R6	X	
ENF_IMPL_R8	X	
ENF_DIAG_R9	X	
ENF_DIAG_R11	X	
ENF_DIAG_R12	X	
ENF_DIAG_R14	X	
ENF_DIAG_R17	X	
ENF_REM_R2	X	
ENF_AUD_R1	X	
ENF_AUD_R2	X	
ENF_AUD_R3	X (not yet)	
ENF_AUD_R4	X (not yet)	
ENF_AUD_R5	X (not yet)	
SLAPL_R18		X
SLAPL_R37		X
SLAPL_R38		X (not yet)
SLAPL_R39		X (not yet)
SLAPL_R40		X
SLAPL_R41		X
SLAPL_R42		X
SLAPL_R43		X (not yet)
SLAPL_R44		X (not yet)

Table 2: Vertical Layers components requirements coverage

In Table 3, we report the status of development of all SPECS artifacts associated to the SLA Platform module and to the Vertical layer. In particular, these artifacts include both the components and the models that are under development in tasks of WP1 and will be completed by the end of the project.

Module	Artifacts under development	Status
SLA Platform module	component:SLA Manager	Available
	component:Service Manager	Available
	component:Security Metrics Catalogue	Available
	component:Interoperability Layer	Available
	component:Auditing	Available
	component:User Manager	Available
	model: SLA machine readable format	Available
	model: SLA XML framework	Available
	model: SPECS data model	Available

Table 3: SLA Platform module and Vertical layer module available artifacts

The SLA related models are described in detail in Deliverable D1.4.1, while the SPECS Data model has been introduced in D1.3 and reported in detail in Section 5 of D1.4.1.

3.2. Repository

The SPECS SLA Platform is offered in form of a Chef[1] cookbook[2] that can be used to automate its deployment and execution over the Enabling Platform. The SLA Platform Chef cookbook is available in the Enabling Platform Chef repository at the following URL:

- https://bitbucket.org/specs-team/specs-core-enabling_platform-repository

In particular, the cookbook devoted to SLA Platform is contained in the `sla-platform` source folder inside the repository.

3.3. Installation

In this section, we illustrate how the SPECS Owner can bootstrap a new instance of the SPECS SLA Platform on the SPECS Testbed, hosted on the infrastructure available in the project, by means of the Enabling Platform Launcher (described in D1.1.3). The SLA Platform instance configured through this guide includes a subset of the components of the SLA Platform, namely the SLA Manager, the Service Manager and the Security Metrics Catalogue.
The process to follow in order to set-up a SPECS SLA Platform and to launch a SPECS Application on top of it is summarized in Figure 2 and is described in detail in the following sub-sections. The steps to be followed are: (i) Select Provider, (ii) Describe Cluster, (iii) Configure Deployment, (iv) Start the Platform, (v) Monitor Resources Start-up, and (vi) Use Platform components.

[1] Chef technology – automatic deployment software system (http://chef.io)
[2] Chef Cookbook – a collection of recipes used by Chef to automatize the deployment process

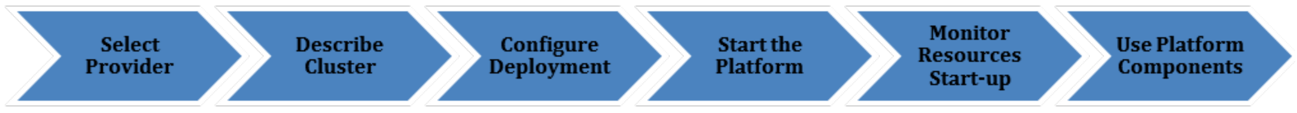


Figure 2: SPECS platform setup process

3.3.1. Requirements

Starting the SPECS SLA Platform by using the SPECS Enabling Platform's Launcher (hereafter, Launcher) requires the End-User (EU) to have access to the resource providers officially supported by SPECS, namely to the SPECS Testbed that is powered by HP Helion Private Cloud Solution [18] (previously known as Eucalyptus Cloud) and Amazon EC2 [7]. The SPECS Testbed usage details are described in D1.6.1.

3.3.2. Starting a SPECS SLA Platform instance

The Launcher offers a web user interface (WUI) hosted at http://dashboard.cloud.specs-project.eu/. The interface provides several controls to create a launching configuration, used to bootstrap the SPECS SLA Platform. It has two sections: *Cluster descriptor*, used to start the SPECS SLA Platform, and *Sessions,* used to monitor the deployment of the launched configurations.

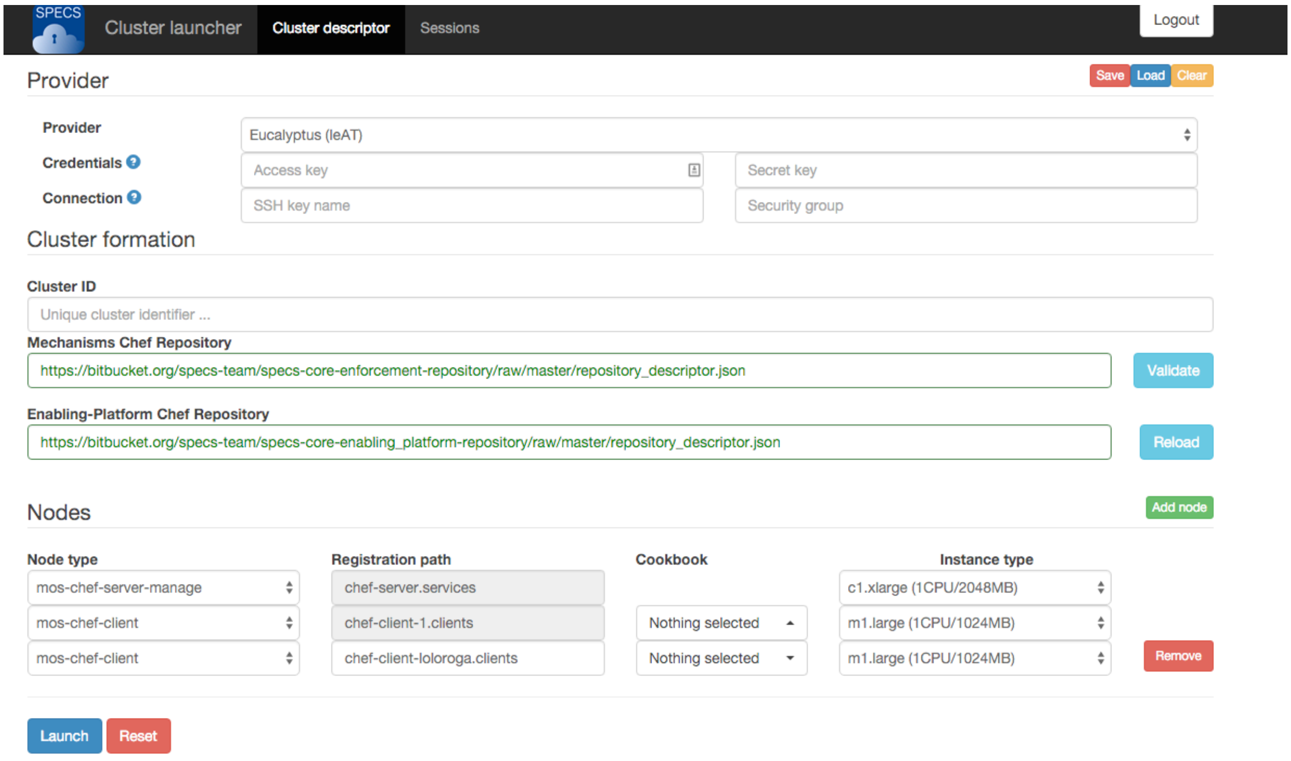
The first time a SPECS Owner accesses the Launcher, a pop-up window requests for the Cloud Resource Allocator token. The token is provided by the SPECS support team[3].

3.3.2.1. Selecting the Provider

The SPECS Owner can start the SPECS SLA Platform either on the IeAT Cluster or on Amazon EC2. Figure 3 shows the *Cluster descriptor* section of the WUI. In the panel labelled as *Provider,* on the top section of the page, the SPECS Owner can specify:

- *Provider name:* the target cloud provider, i.e., either the SPECS Testbed (Eucalyptus) or Amazon EC2;
- *Credentials:* the access key and secret key necessary for authentication; for the SPECS Testbed, the SPECS Owner can click on the question mark sign near *Credentials* label in order to get information about how to generate a credentials pair; for Amazon EC2, the Owner must follow Amazon tutorial [15] on how to obtain an access credentials pair;
- *Connection:* the SSH key name (i.e., the secure access public key name for remote connection access authorization to the virtual machines) and the security group (i.e., the name of the security group with the access rules to be applied to the VMs). By default, the access to the virtual machines (VMs) is fully restricted on both cloud providers. The SPECS Owner must follow the SPECS Testbed tutorial [16] and the Amazon EC2 documentation[17] to obtain this information.

[3] support@specs-project.eu



Provider

Provider	Eucalyptus (IeAT)	
Credentials	Access key	Secret key
Connection	SSH key name	Security group

Cluster formation

Cluster ID: Unique cluster identifier ...

Mechanisms Chef Repository: https://bitbucket.org/specs-team/specs-core-enforcement-repository/raw/master/repository_descriptor.json

Enabling-Platform Chef Repository: https://bitbucket.org/specs-team/specs-core-enabling_platform-repository/raw/master/repository_descriptor.json

Nodes

Node type	Registration path	Cookbook	Instance type
mos-chef-server-manage	chef-server.services		c1.xlarge (1CPU/2048MB)
mos-chef-client	chef-client-1.clients	Nothing selected	m1.large (1CPU/1024MB)
mos-chef-client	chef-client-loloroga.clients	Nothing selected	m1.large (1CPU/1024MB)

Figure 3: SPECS Launcher. Web User Interface

3.3.2.2. Describing the cluster configuration

The second panel of the *Cluster descriptor* section, labelled *Cluster Formation*, sets the information associated to the composition of the VM cluster. The cluster must include a machine devoted to hosting the Chef Server[4], used to deploy both the components of the platform and the security mechanisms needed to enforce security capabilities on top of the services offered to End-Users.

The set of information reported in the panel is the following:

- *Cluster ID:* a string of letters and numbers that represents the identification *ID* of the current SPECS platform configuration;
- *Mechanisms Chef Repository:* the repository containing all the Chef cookbooks associated to mechanisms used by SPECS to configure Target Services according to SLAs;
- *Enabling-Platform Chef Repository:* the repository that contains all the Chef cookbooks used by the Launcher to start SPECS core components.

The SPECS Owner should supply the *Cluster ID*, which is a mnemonic name with the only constraint that it must not be used in other sessions. Links to Mechanism Chef Repository and Enabling-Platform Chef Repository are, by default, filled with SPECS official Bitbucket repositories. (https://bitbucket.org/specs-team/specs-core-enforcement-repository and https://bitbucket.org/specs-team/specs-core-enabling_platform-repository respectively).

3.3.2.3. Deployment configuration

The last panel, labelled *Nodes*, enables the SPECS Owner to choose how many nodes to use and to configure such nodes with the components needed to start the SPECS SLA Platform.

[4] Chef Server – the central service used to host the cookbooks and to dispatch the deployment process to the clients

Note that the first node always hosts the Chef Server, while the other nodes, hosting the clients, can be customized.
The panel reports a row for each node; the button *Add Node* adds new rows to the panel. It is possible to remove one node from the configuration by simply using the button *Remove Node*.

For each node, the SPECS Owner must report:

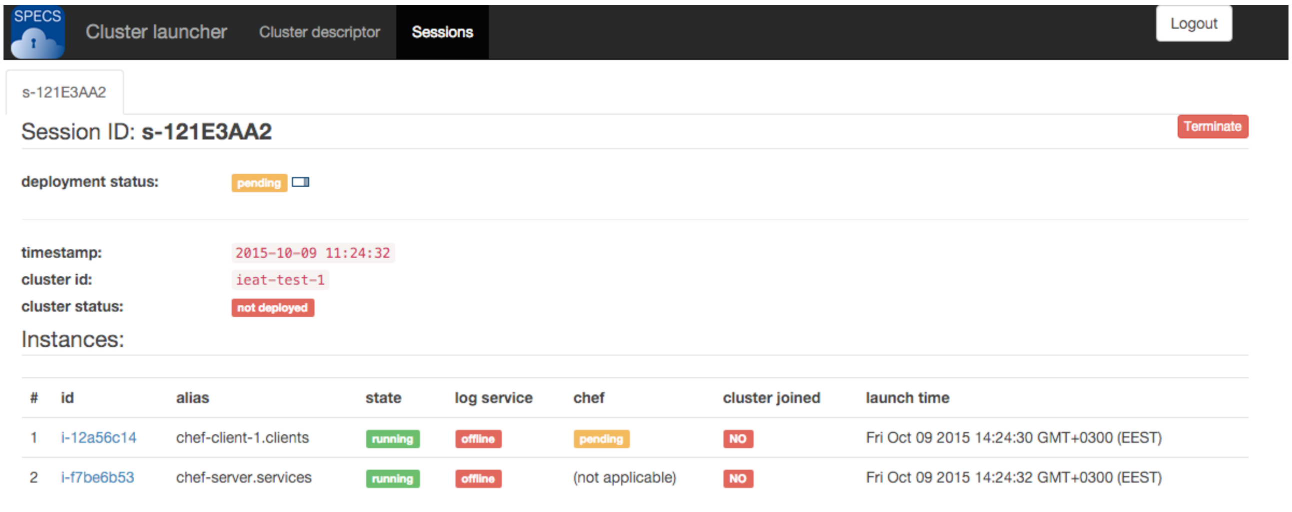
- *Node Type:* nodes can be of two types, namely Chef Server and Chef Client. The node configured by default as the Chef Server is used to store the configuration templates for both core components and mechanisms, while the Chef Client nodes are used to actually install the core components and the mechanisms;
- *Registration path:* this is the logical name associated to the node, when registered in the infrastructure (it is configured automatically);
- *Cookbook:* this is a list box that enables the selection of components to deploy on the node;
- *Instance Type:* reports the type of instance used to host the VM (hardware configuration). Available instance types are described in the testbed deliverables (D1.6.1). As an example, Amazon provides different types of instances, optimized for different needs, such as T2 (burstable performance instances), M (general purpose instances), etc.[5]

3.3.2.4. Starting the cluster

The last step needed to create the SPECS platform is to launch the cluster creation by pressing the *Launch* button in the *Cluster descriptor* shown in Figure 3. It opens a popup window, summarizing the configuration and asking for confirmation.
A new session is created (see Figure 4), which enables the SPECS Owner to monitor the cluster preparation and to access the resources, once they are available.

3.3.3. Controlling the SPECS SLA Platform instances

The *Sessions* tab (see Figure 4) shows information about the deployment process of the launched SPECS SLA Platform. Every time a SPECS SLA Platform is started, a new *Session* section is available in the panel enabling the monitoring of the infrastructure hosting the platform itself.



Session ID: s-121E3AA2

deployment status: pending

timestamp: 2015-10-09 11:24:32
cluster id: ieat-test-1
cluster status: not deployed

Instances:

#	id	alias	state	log service	chef	cluster joined	launch time
1	i-12a56c14	chef-client-1.clients	running	offline	pending	NO	Fri Oct 09 2015 14:24:30 GMT+0300 (EEST)
2	i-f7be6b53	chef-server.services	running	offline	(not applicable)	NO	Fri Oct 09 2015 14:24:32 GMT+0300 (EEST)

Figure 4: SPECS Launcher. Sessions interface

[5] https://aws.amazon.com/ec2/instance-types/?nc1=h_ls

The *Session*s page reports two main panels, the former devoted to summarizing the information given at configuration time, and the latter reporting the status of each node composing the cluster hosting SPECS.
The configuration panel contains the following information:

- *Deployment status*: represents the overall deployment process status; when the status becomes "deployed", the SPECS SLA Platform should be ready for use;
- *Timestamp*: the starting timestamp of the SPECS SLA Platform deployment process;
- *Cluster id*: the SPECS SLA Platform identification string chosen by the SPECS Owner at the *Cluster descriptor* step;
- *Cluster status:* provides information regarding the status of the Chef infrastructure deployment; when the Chef clients are all automatically registered, the status will change to "deployed" and the recipes will be applied to nodes according to what the SPECS Owner chose in the configuration step.

The second panel, labelled *Instances,* reports a table with a row for each node hosting the SPECS platform components. Each row contains:

- *Id:* the id number generated by the cloud provider;
- *Alias:* the registration path defined by the Owner (internally used);
- *State:* the status of the VM, which can assume the following values: pending, running or failed;
- *Log service:* a special service from which various debugging information can be extracted directly using a web browser; when state changes to "online", a click on the label will open a new browser tab listing the available logging information;
- *Chef:* reflects chef configuration status; when the value changes to "joined" then the node is part of the Chef infrastructure;
- *Cluster joined:* the status of the node inside the distributed system (internally used);
- *Launcher time* – the VM creation time.

3.3.4. Accessing the SPECS SLA Platform

When the SPECS SLA Platform is in "running" state, as reported in the *Session*s tab, all the nodes are started and configured and all SPECS components belonging to it are deployed on the chosen nodes.
Once started, the SLA Platform components are accessible through the exposed APIs, i.e., the *SLA API*, offered by the SLA Manager, and the *Services API*, offered by the Service Manager and by the Security Metrics Catalogue. Both APIs are accessible at the base URI `/cloud-sla/`.
A detailed description of available calls is reported in D1.3 and in D1.4.1, which presents some updates due to the architecture design refinement.

4. SPECS SLA Platform Components

In this section, the details on implementation, installation, usage and testing of the single components belonging to the SLA Platform is given.

4.1. SLA Manager

The SLA Manager component is responsible for handling the SLA life cycle. It stores and manages the SLAs signed by End-users and enables to perform the CRUD[6] operations on them.

4.1.1. Repository

The SLA Manager component consists of two sub-components: the *backend*, which provides SLA management and storing features, and the *frontend*, which exposes a REST API. Both sub-components are available on Bitbucket at the following URLs:

- the backend:
  - https://bitbucket.org/specs-team/specs-core-sla_platform-sla_manager
- the frontend:
  - https://bitbucket.org/specs-team/specs-core-sla_platform-sla_manager-api

4.1.2. Description and design

As mentioned in the previous section, the SLA Manager consists of two sub-components. The *backend* implements all the operations that enable the management of SLAs and grants the persistence of the SLAs and of associated information.

The *frontend* sub-component contains the implementation of the SLA API, described in deliverable D1.3 and in deliverable D1.4.1.
The main exposed resources are:

- **SLAs Resource:** represents a SLAs collection and the management operations associated with it;
- **SLA Resource:** represents a particular SLA and the management operations associated with it;

The description and the design of the SLA Manager component are available in deliverable D1.4.1.

4.1.3. Installation

In this section we report the installation guide, which covers two scenarios:

- Installing by using precompiled binaries (SPECS recommended);
- compiling and installing from source (for advanced users);

4.1.3.1. Installing by using precompiled binaries

The precompiled binaries are available under the SPECS Artifact Repository [4].

Requirements

- Oracle Java JDK 7;
- Apache Tomcat 7.0.x;

[6]CRUD stands for create, read, update and delete and represents the four basic functions of persistent storage;

Installation steps

1. download the web application archive (war) file from the artifact repository : http://ftp.specs-project.eu/public/artifacts/sla-platform/sla-manager/sla-manager-api-STABLE.war
2. deploy the war in the java servlet/web container;

If Apache Tomcat 7.0.x is used, the war file needs to be copied into the "/webapps" folder inside the home directory (CATALINA_HOME) of Apache Tomcat 7.0.x.

4.1.3.2. Compiling and installing from source

In order to compile and install the SLA Manager, it is mandatory first to process the *backend*, and afterwards the *frontend*.

Requirements

- a Git client [5];
- Apache Maven 3.3.x [6];
- Oracle Java JDK 7;
- Apache Tomcat 7.0.x [7]

Backend installation steps:

1. clone the Bitbucket repository with the following command:

git clone git@bitbucket.org:specs-team/specs-core-sla_platform-sla_manager.git

2. under *specs-core-sla_platform-sla_manager* run:

mvn install

Frontend installation steps:

1. clone the Bitbucket repository with the following command:

git clone git@bitbucket.org:specs-team/specs-core-sla_platform-sla_manager-api.git

2. under *specs-core-sla_platform-sla_manager-api* run:

mvn package

The *backend* installation generates the artifact used by the *frontend*. The *frontend* installation generates a web application archive (war) file, under the "/target" subfolder. In order to use the component, the war file has to be deployed in the java servlet/web container. If Apache Tomcat 7.0.x is used, the war file needs to be copied into the "/webapps" folder inside the home directory (CATALINA_HOME) of Apache Tomcat 7.0.x.

4.1.4. Usage

The SLA Manager component exposes a REST API interface. All the exposed REST resources are mapped under the path "/cloud-sla/*". The mapping rules are defined in the *web.xml* file under WEB-INF folder (*CATALINA_HOME/webapps/sla-manager-api/WEB-INF/*).

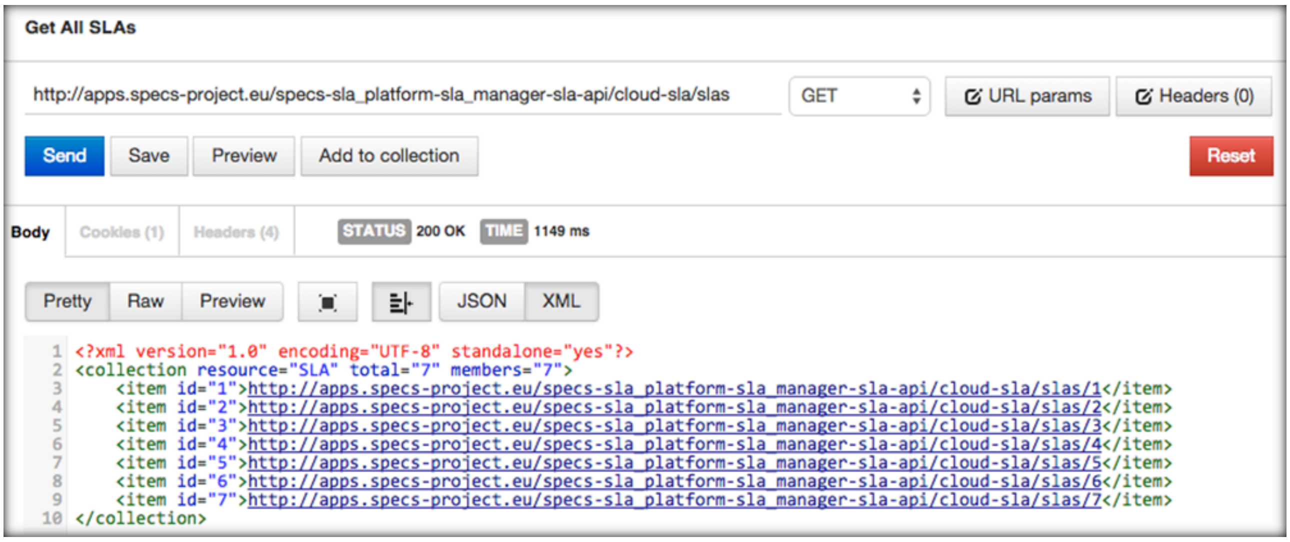
```
<servlet-mapping>
     <servlet-name>Jersey REST Service</servlet-name>
     <url-pattern>/cloud-sla/*</url-pattern>
</servlet-mapping>
```

Moreover, all the REST resources are represented by specific java classes under the package "*eu.specsproject.slaplatform.slamanager.restfrontend*", defined in the *web.xml* file under WEB-INF folder:

```
        <init-param>
        <param-name>jersey.config.server.provider.packages</param-name>
        <param-value>
eu.specsproject.slaplatform.slamanager.restfrontend</param-value>
        </init-param>
```

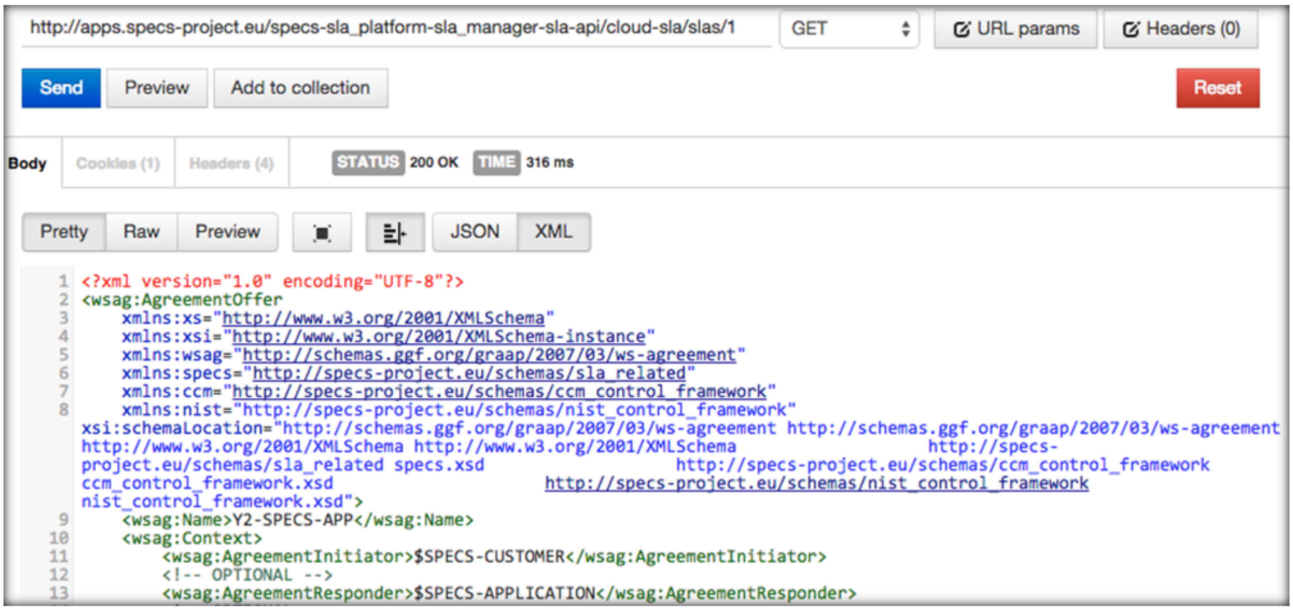
At the start-up of the component, all the REST resources are configured based on the configuration parameters defined in the *web.xml* file.

Figure 5 and Figure 6 describe two examples of the REST API calls used to retrieve SLAs information.



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<collection resource="SLA" total="7" members="7">
    <item id="1">http://apps.specs-project.eu/specs-sla_platform-sla_manager-sla-api/cloud-sla/slas/1</item>
    <item id="2">http://apps.specs-project.eu/specs-sla_platform-sla_manager-sla-api/cloud-sla/slas/2</item>
    <item id="3">http://apps.specs-project.eu/specs-sla_platform-sla_manager-sla-api/cloud-sla/slas/3</item>
    <item id="4">http://apps.specs-project.eu/specs-sla_platform-sla_manager-sla-api/cloud-sla/slas/4</item>
    <item id="5">http://apps.specs-project.eu/specs-sla_platform-sla_manager-sla-api/cloud-sla/slas/5</item>
    <item id="6">http://apps.specs-project.eu/specs-sla_platform-sla_manager-sla-api/cloud-sla/slas/6</item>
    <item id="7">http://apps.specs-project.eu/specs-sla_platform-sla_manager-sla-api/cloud-sla/slas/7</item>
</collection>
```

Figure 5: SLA Manager – *Get a collection of SLAs* API call example



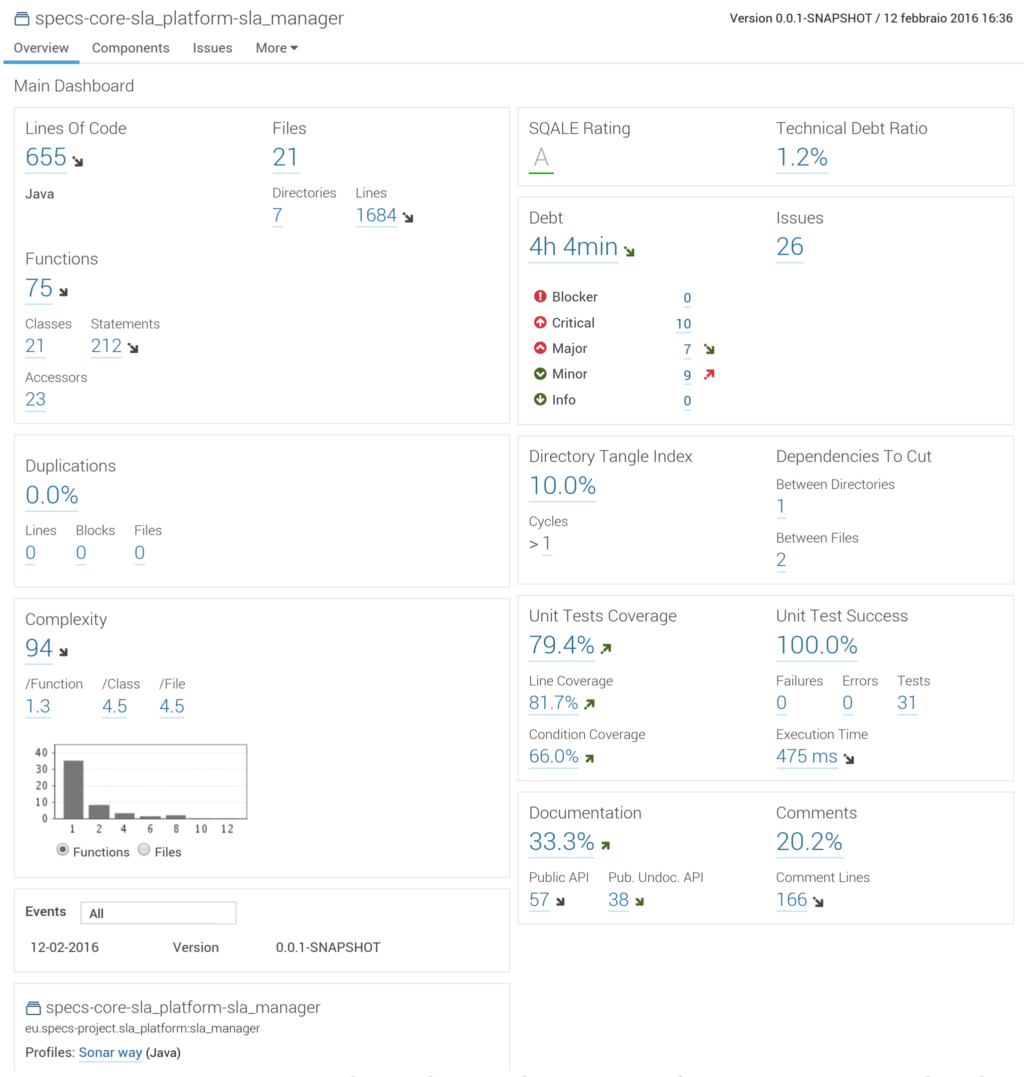
```
<?xml version="1.0" encoding="UTF-8"?>
<wsag:AgreementOffer
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:wsag="http://schemas.ggf.org/graap/2007/03/ws-agreement"
    xmlns:specs="http://specs-project.eu/schemas/sla_related"
    xmlns:ccm="http://specs-project.eu/schemas/ccm_control_framework"
    xmlns:nist="http://specs-project.eu/schemas/nist_control_framework"
xsi:schemaLocation="http://schemas.ggf.org/graap/2007/03/ws-agreement http://schemas.ggf.org/graap/2007/03/ws-agreement
http://www.w3.org/2001/XMLSchema http://www.w3.org/2001/XMLSchema                    http://specs-
project.eu/schemas/sla_related specs.xsd                    http://specs-project.eu/schemas/ccm_control_framework
ccm_control_framework.xsd               http://specs-project.eu/schemas/nist_control_framework
nist_control_framework.xsd">
    <wsag:Name>Y2-SPECS-APP</wsag:Name>
    <wsag:Context>
        <wsag:AgreementInitiator>$SPECS-CUSTOMER</wsag:AgreementInitiator>
        <!-- OPTIONAL -->
        <wsag:AgreementResponder>$SPECS-APPLICATION</wsag:AgreementResponder>
```

Figure 6: SLA Manager – *Get a single SLA* API call example

The complete details of the REST API are described in Deliverable 1.3.

4.1.5. Tests

The JUnit[7] framework has been used to test the component. Figure 7 and Figure 8 illustrates the code quality analysis report produced for SLA Manager backend and frontend. In Annex A, the tests conducted for this component are detailed with the list of the methods that have been covered and the code coverage percentage currently reached.



specs-core-sla_platform-sla_manager

Version 0.0.1-SNAPSHOT / 12 febbraio 2016 16:36

Overview Components Issues More

Main Dashboard

Lines Of Code 655 ↘ Java

Files 21 — Directories 7, Lines 1684 ↘

Functions 75 ↘ — Classes 21, Statements 212 ↘, Accessors 23

SQALE Rating A

Technical Debt Ratio 1.2%

Debt 4h 4min ↘

Issues 26

Blocker	0
Critical	10
Major	7 ↘
Minor	9 ↗
Info	0

Duplications 0.0% — Lines 0, Blocks 0, Files 0

Directory Tangle Index 10.0% — Cycles > 1

Dependencies To Cut — Between Directories 1, Between Files 2

Complexity 94 ↘ — /Function 1.3, /Class 4.5, /File 4.5

Functions Files

Unit Tests Coverage 79.4% ↗ — Line Coverage 81.7% ↗, Condition Coverage 66.0% ↗

Unit Test Success 100.0% — Failures 0, Errors 0, Tests 31, Execution Time 475 ms ↘

Documentation 33.3% ↗ — Public API 57 ↘, Pub. Undoc. API 38 ↘

Comments 20.2% — Comment Lines 166 ↘

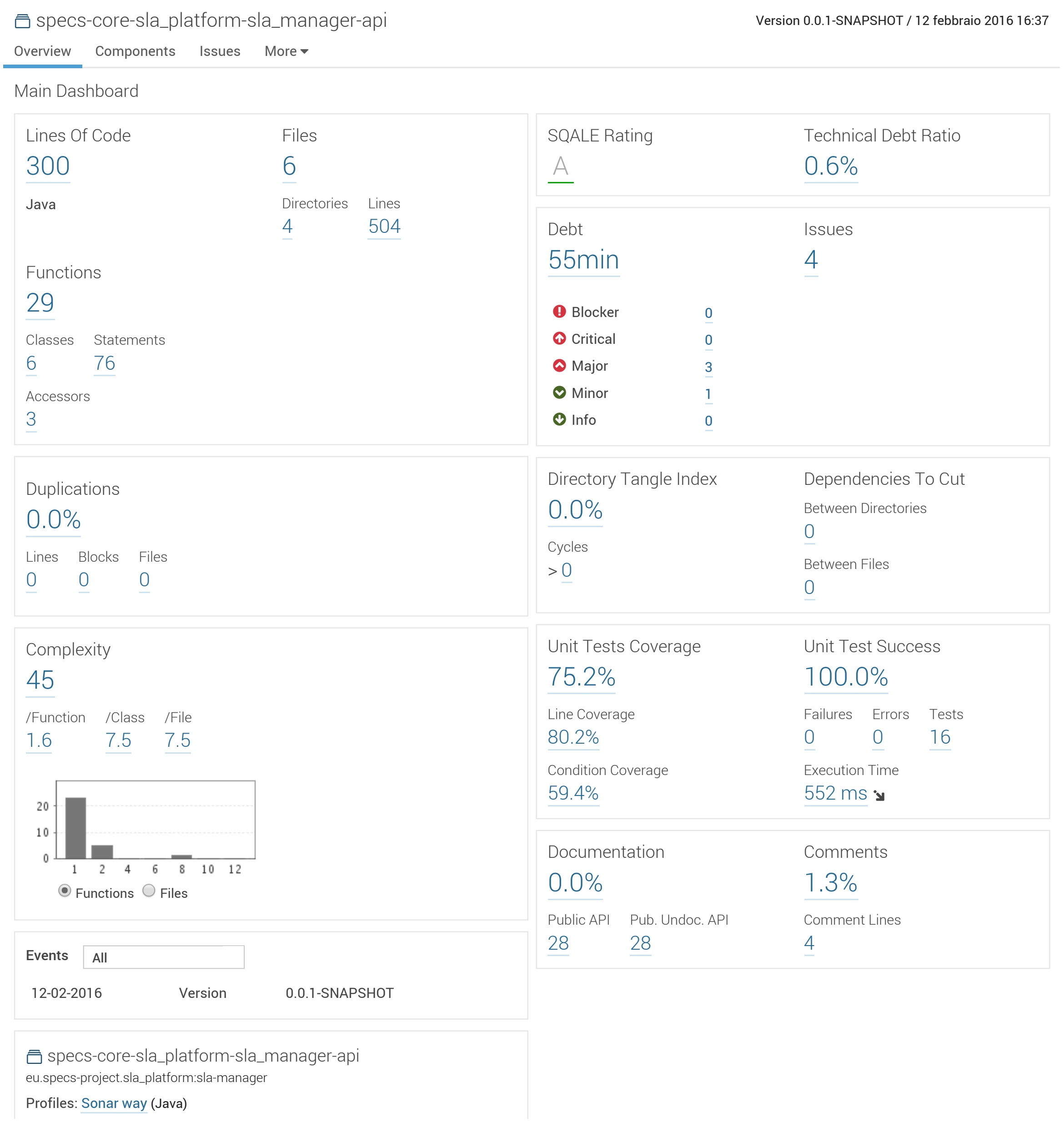
Events All

12-02-2016 Version 0.0.1-SNAPSHOT

specs-core-sla_platform-sla_manager
eu.specs-project.sla_platform:sla_manager
Profiles: Sonar way (Java)

Figure 7: Code Quality Analysis Report for SLA Manager Backend

[7] jUnit – Java unit testing framework (http://junit.org/)


specs-core-sla_platform-sla_manager-api — Version 0.0.1-SNAPSHOT / 12 febbraio 2016 16:37

Overview · Components · Issues · More

Main Dashboard

Lines Of Code 300 (Java)
Files 6 — Directories 4, Lines 504
Functions 29 — Classes 6, Statements 76, Accessors 3

SQALE Rating A
Technical Debt Ratio 0.6%
Debt 55min
Issues 4 — Blocker 0, Critical 0, Major 3, Minor 1, Info 0

Duplications 0.0% — Lines 0, Blocks 0, Files 0

Directory Tangle Index 0.0% — Cycles > 0
Dependencies To Cut — Between Directories 0, Between Files 0

Complexity 45 — /Function 1.6, /Class 7.5, /File 7.5

Unit Tests Coverage 75.2% — Line Coverage 80.2%, Condition Coverage 59.4%
Unit Test Success 100.0% — Failures 0, Errors 0, Tests 16, Execution Time 552 ms

Documentation 0.0% — Public API 28, Pub. Undoc. API 28
Comments 1.3% — Comment Lines 4

Events All
12-02-2016 Version 0.0.1-SNAPSHOT

specs-core-sla_platform-sla_manager-api
eu.specs-project.sla_platform:sla-manager
Profiles: Sonar way (Java)

Figure 8: Code Quality Analysis Report for SLA Manager Frontend

4.2. Service Manager

The Service Manager is responsible for managing the Security Mechanisms (SMs) (see Deliverable D4.2.2 and Deliverable D4.3.2) by using a persistent data storage that is CRUD enabled.

The main design requirement imposes the ability to query the Security Mechanisms and Security Capabilities database in order to obtain Mechanisms that are able to enforce or monitor specific metrics and satisfy specific capabilities. Moreover, the component should allow the End-user to get the associated metadata of each Security Mechanism.

4.2.1. Repository

The component consists of two sub-components: the *backend*, which provides Security Metrics and Security Capabilities management and storing features, and the *frontend*, which exposes a REST API. Both sub-components are available on Bitbucket at the following URLs:

- the backend:
  - https://bitbucket.org/specs-team/specs-core-sla_platform-service_manager
- the front-end:
  - https://bitbucket.org/specs-team/specs-core-sla_platform-service_manager-api

4.2.2. Description and design

As mentioned in the previous section, the Services Manager consists of two sub-components. The *backend* offers the implementation of all the functions that enable the management of the Security Mechanisms and Security Capabilities database (CRUD enabled functions). These are provided by:

- **Service Manager Abstract Implementation**, which offers the methods for managing the Security Mechanisms and the Security Capabilities;
- **Service Manager SQl JPA**, which provides the functionalities to manage the persistence of the Security Mechanisms and the Security Capabilities.

The *frontend* sub-component contains the implementation of the Services API, described in deliverables D1.3 and D1.4.1.
The main exposed resources are:

- **SMs Resource**: represents a collection of Security Mechanisms and the management operations associated with it;
- **SM Resource**: represents a specific created Security Mechanism and the management operations associated with it;
- **SM Metadata Resource:** represents the metadata associated with a specific Security Mechanism and the management operations associated with it;
- **SCs Resource**: represents the collection of Security Capabilities and the management operations associated with it;
- **SC Resource**: represents a specific created Security Capability and and the management operations associated with it.

The description and the design of the Services Manager component are available in Deliverable D1.4.1.

4.2.3. Installation

In this section we provide an installation guide for the Service manager that covers two scenarios:

- Installing by using precompiled binaries (SPECS recommended);
- compiling and installing from source (for advanced users);

4.2.3.1. Installing by using precompiled binaries

The precompiled binaries are available under the SPECS Artifact Repository [4].

Requirements

- Oracle Java JDK 7;
- SQLite 3.9.x;
- Apache Tomcat 7.0.x;

Installation steps

1. download the web application archive (war) file from the artifact repostiry :

http://ftp.specs-project.eu/public/artifacts/sla-platform/service-manager/service-manager-STABLE.war

2. deploy the war file in the java servlet/web container
3. if Apache Tomcat 7.0.x is used, the war file needs to be copied into the "/webapps" folder inside the home directory (CATALINA_HOME) of Apache Tomcat 7.0.x.

4.2.3.2. Compiling and installing from source

In order to compile and install the Services Manager it is mandatory first to process the *backend* and afterwards the *frontend*.

Requirements

- a Git client;
- Apache Maven 3.3.x;
- Oracle Java JDK 7;
- SQLite 3.9.x;
- Apache Tomcat 7.0.x;

Backend installation steps:

1. clone the Bitbucket repository:

git clone git@bitbucket.org:specs-team/specs-core-sla_platform-service_manager.git

2. change the configuration of the database in the *persistence.xml* file (the file is located in the folder "src/main/resource" and "/src/test/resources") in order to define the correct path where the SQLite database will be created;
3. under *specs-core-sla_platform-service_manager* run:

mvn install

Frontend installation steps:

4. clone the Bitbucket repository:

git clone git@bitbucket.org:specs-team/specs-core-sla_platform-service_manager-api.git

5. under *specs-core-sla_platform-service_manager-api* run:

mvn package

The *backend* installation generates the artifact used by the *frontend*. The *frontend* installation generates a web application archive (war) file, under the "/target" subfolder. In order to use the component, the war file has to be deployed in the java servlet/web container. If Apache Tomcat 7.0.x is used, the war file needs to be copied into the "/webapps" folder inside the home directory (CATALINA_HOME) of Apache Tomcat 7.0.x.

4.2.4. Usage

The Service Manager component exposes a REST API interface. All the exposed REST resources are mapped under the path "/cloud-sla/*". The mapping rules are defined in the *web.xml* file under WEB-INF folder (*CATALINA_HOME/webapps/service-manager-api/WEB-INF/*):

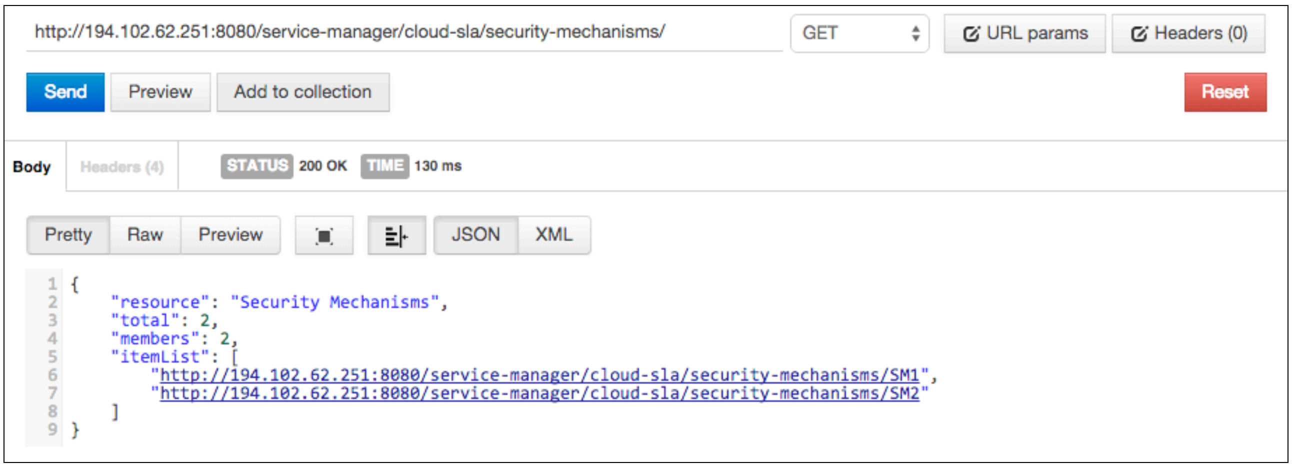
```
	<servlet-mapping>
		<servlet-name>Jersey REST Service</servlet-name>
		<url-pattern>/cloud-sla/*</url-pattern>
	</servlet-mapping>
```

Moreover, all the REST resources are represented by specific java classes under the package "*eu.specsproject.slaplatform.servicemanager.restfrontend*", defined in the *web.xml* file under WEB-INF folder:

```
	<init-param>
	<param-name>jersey.config.server.provider.packages</param-name>
	<param-
value>eu.specsproject.slaplatform.servicemanager.restfrontend</param-
value>
	</init-param>
```

At the start-up of the component, all the REST resources are configured based on the configuration parameters defined in the *web.xml* file.

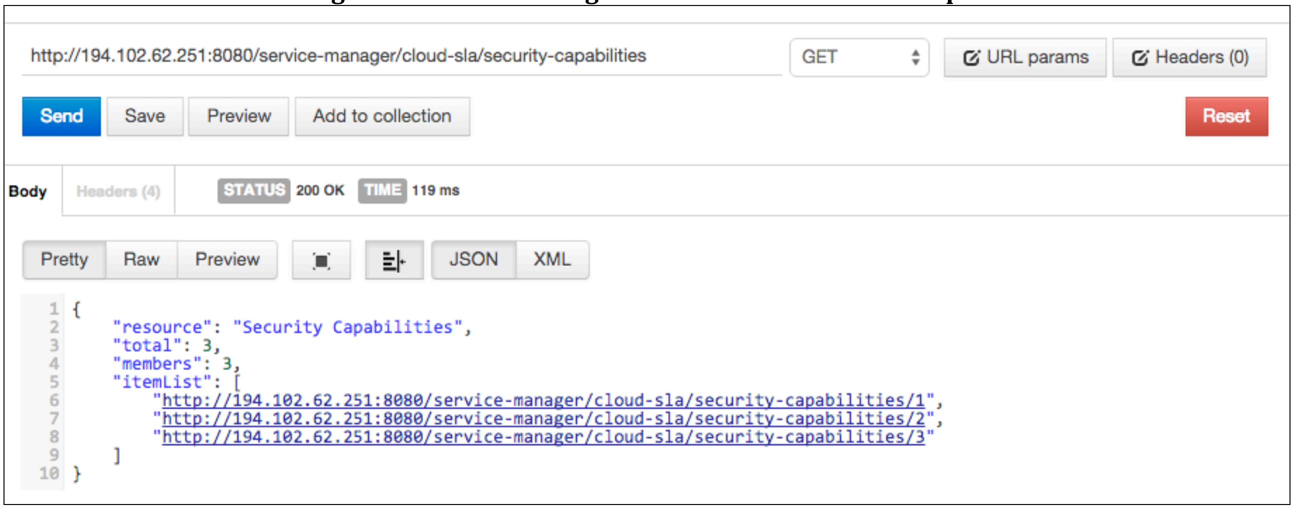
Figure 9, Figure 10 and Figure 11 describe three examples of the REST API calls used to retrieve SMs and SCs information.



http://194.102.62.251:8080/service-manager/cloud-sla/security-mechanisms/

```
{
    "resource": "Security Mechanisms",
    "total": 2,
    "members": 2,
    "itemList": [
        "http://194.102.62.251:8080/service-manager/cloud-sla/security-mechanisms/SM1",
        "http://194.102.62.251:8080/service-manager/cloud-sla/security-mechanisms/SM2"
    ]
}
```

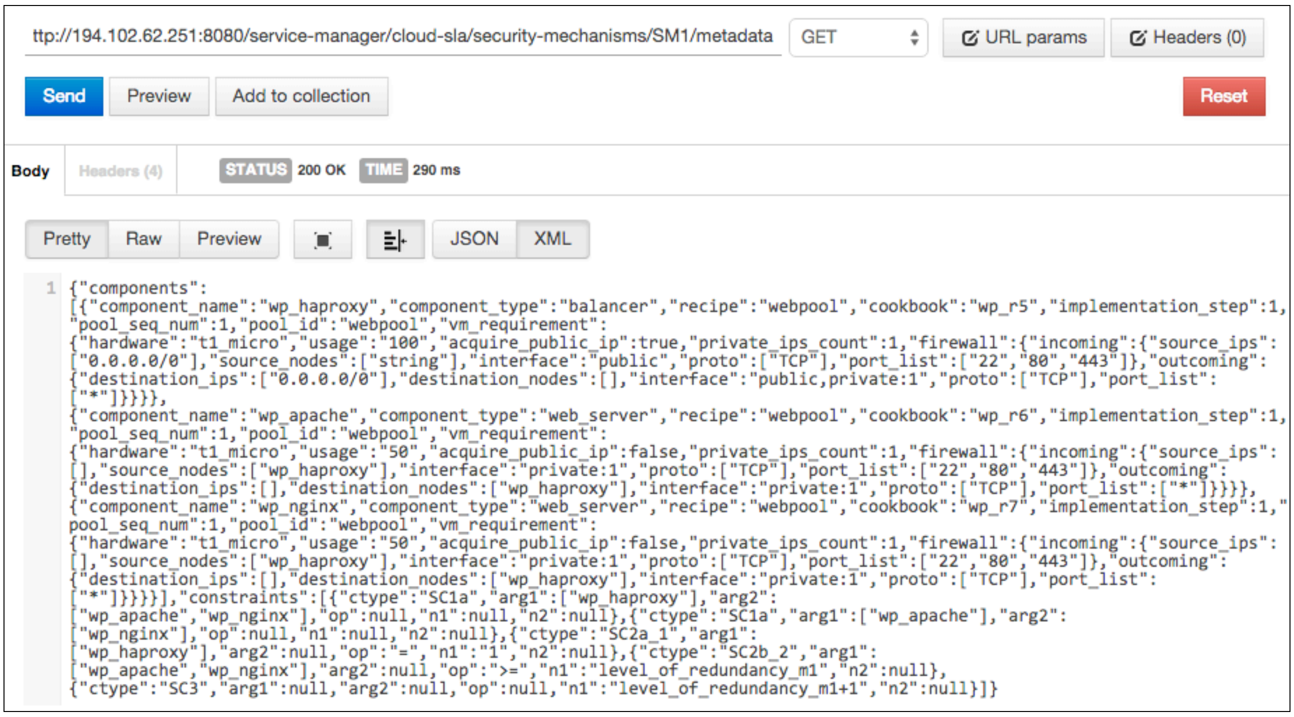
Figure 9: Service Manager - *Get all SMs* API call example



http://194.102.62.251:8080/service-manager/cloud-sla/security-capabilities

```
{
    "resource": "Security Capabilities",
    "total": 3,
    "members": 3,
    "itemList": [
        "http://194.102.62.251:8080/service-manager/cloud-sla/security-capabilities/1",
        "http://194.102.62.251:8080/service-manager/cloud-sla/security-capabilities/2",
        "http://194.102.62.251:8080/service-manager/cloud-sla/security-capabilities/3"
    ]
}
```

Figure 10: Service Manager - *Get SM Metadata* API call example



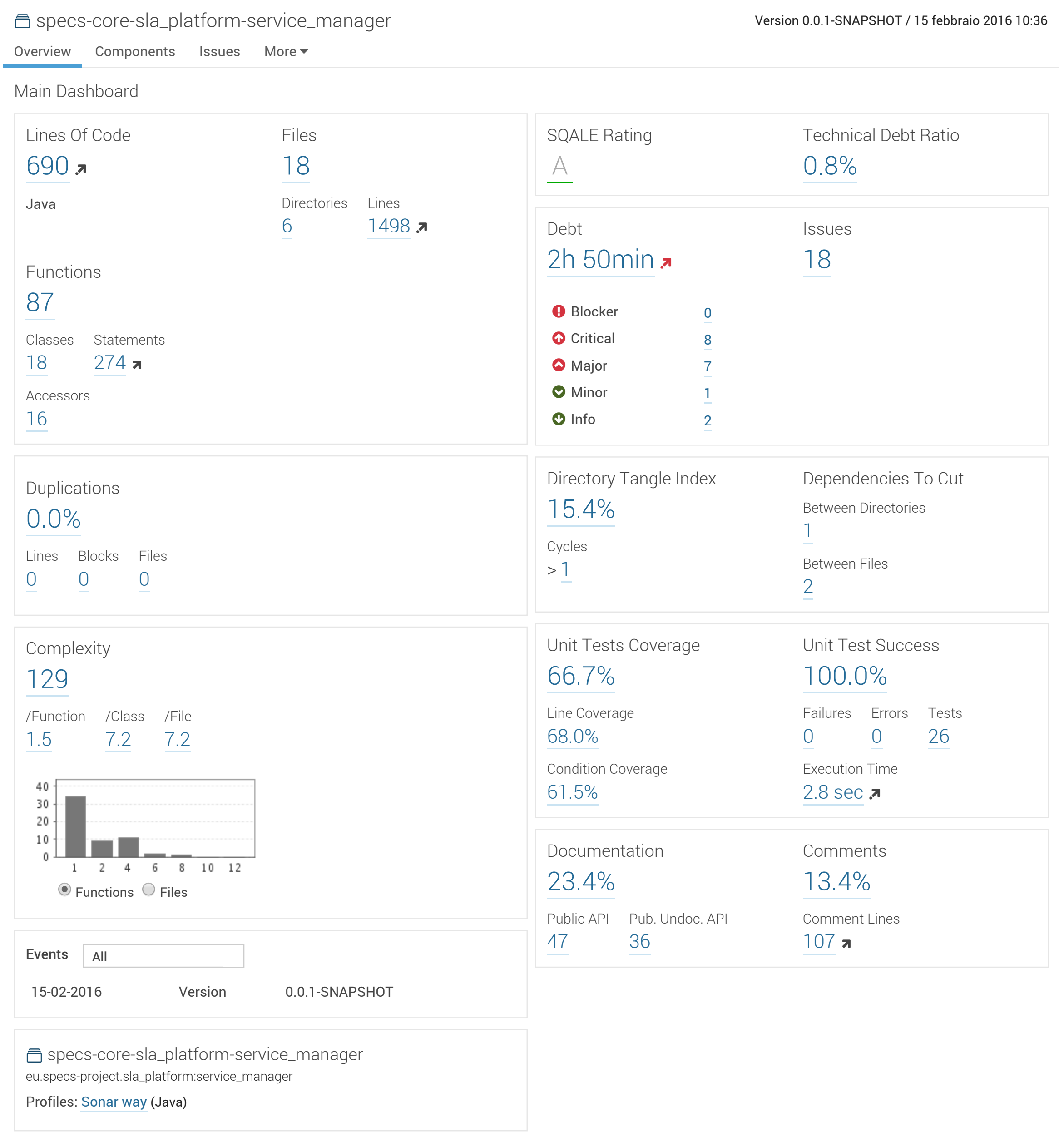
```
{"components":
[{"component_name":"wp_haproxy","component_type":"balancer","recipe":"webpool","cookbook":"wp_r5","implementation_step":1,
"pool_seq_num":1,"pool_id":"webpool","vm_requirement":
{"hardware":"t1_micro","usage":"100","acquire_public_ip":true,"private_ips_count":1,"firewall":{"incoming":{"source_ips":
["0.0.0.0/0"],"source_nodes":["string"],"interface":"public","proto":["TCP"],"port_list":["22","80","443"]},"outcoming":
{"destination_ips":["0.0.0.0/0"],"destination_nodes":[],"interface":"public,private:1","proto":["TCP"],"port_list":
["*"]}}}},
{"component_name":"wp_apache","component_type":"web_server","recipe":"webpool","cookbook":"wp_r6","implementation_step":1,
"pool_seq_num":1,"pool_id":"webpool","vm_requirement":
{"hardware":"t1_micro","usage":"50","acquire_public_ip":false,"private_ips_count":1,"firewall":{"incoming":{"source_ips":
[],"source_nodes":["wp_haproxy"],"interface":"private:1","proto":["TCP"],"port_list":["22","80","443"]},"outcoming":
{"destination_ips":[],"destination_nodes":["wp_haproxy"],"interface":"private:1","proto":["TCP"],"port_list":["*"]}}}},
{"component_name":"wp_nginx","component_type":"web_server","recipe":"webpool","cookbook":"wp_r7","implementation_step":1,"
pool_seq_num":1,"pool_id":"webpool","vm_requirement":
{"hardware":"t1_micro","usage":"50","acquire_public_ip":false,"private_ips_count":1,"firewall":{"incoming":{"source_ips":
[],"source_nodes":["wp_haproxy"],"interface":"private:1","proto":["TCP"],"port_list":["22","80","443"]},"outcoming":
{"destination_ips":[],"destination_nodes":["wp_haproxy"],"interface":"private:1","proto":["TCP"],"port_list":
["*"]}}}}],"constraints":[{"ctype":"SC1a","arg1":["wp_haproxy"],"arg2":
["wp_apache","wp_nginx"],"op":null,"n1":null,"n2":null},{"ctype":"SC1a","arg1":["wp_apache"],"arg2":
["wp_nginx"],"op":null,"n1":null,"n2":null},{"ctype":"SC2a_1","arg1":
["wp_haproxy"],"arg2":null,"op":"=","n1":"1","n2":null},{"ctype":"SC2b_2","arg1":
["wp_apache","wp_nginx"],"arg2":null,"op":">=","n1":"level_of_redundancy_m1","n2":null},
{"ctype":"SC3","arg1":null,"arg2":null,"op":null,"n1":"level_of_redundancy_m1+1","n2":null}]}
```

Figure 11: Service Manager - *Get all SCs* API call example

The complete details of the REST API are described in Deliverable 1.3.

4.2.5. Tests

The JUnit framework has been used to test the component. Figure 12 and Figure 13Figure 7 and Figure 8 illustrates the code quality analysis report produced for SLA Manager backend and frontend. In Annex A, the tests conducted for this component are detailed with the list of the methods that have been covered and the code coverage percentage currently reached.



specs-core-sla_platform-service_manager

Version 0.0.1-SNAPSHOT / 15 febbraio 2016 10:36

Overview Components Issues More

Main Dashboard

Lines Of Code 690

Java

Files 18

Directories 6 Lines 1498

Functions 87

Classes 18 Statements 274

Accessors 16

SQALE Rating A

Technical Debt Ratio 0.8%

Debt 2h 50min

Issues 18

Blocker	0
Critical	8
Major	7
Minor	1
Info	2

Duplications 0.0%

Lines 0 Blocks 0 Files 0

Directory Tangle Index 15.4%

Cycles > 1

Dependencies To Cut

Between Directories 1

Between Files 2

Complexity 129

/Function 1.5 /Class 7.2 /File 7.2

Unit Tests Coverage 66.7%

Line Coverage 68.0%

Condition Coverage 61.5%

Unit Test Success 100.0%

Failures 0 Errors 0 Tests 26

Execution Time 2.8 sec

Documentation 23.4%

Public API 47 Pub. Undoc. API 36

Comments 13.4%

Comment Lines 107

Events All

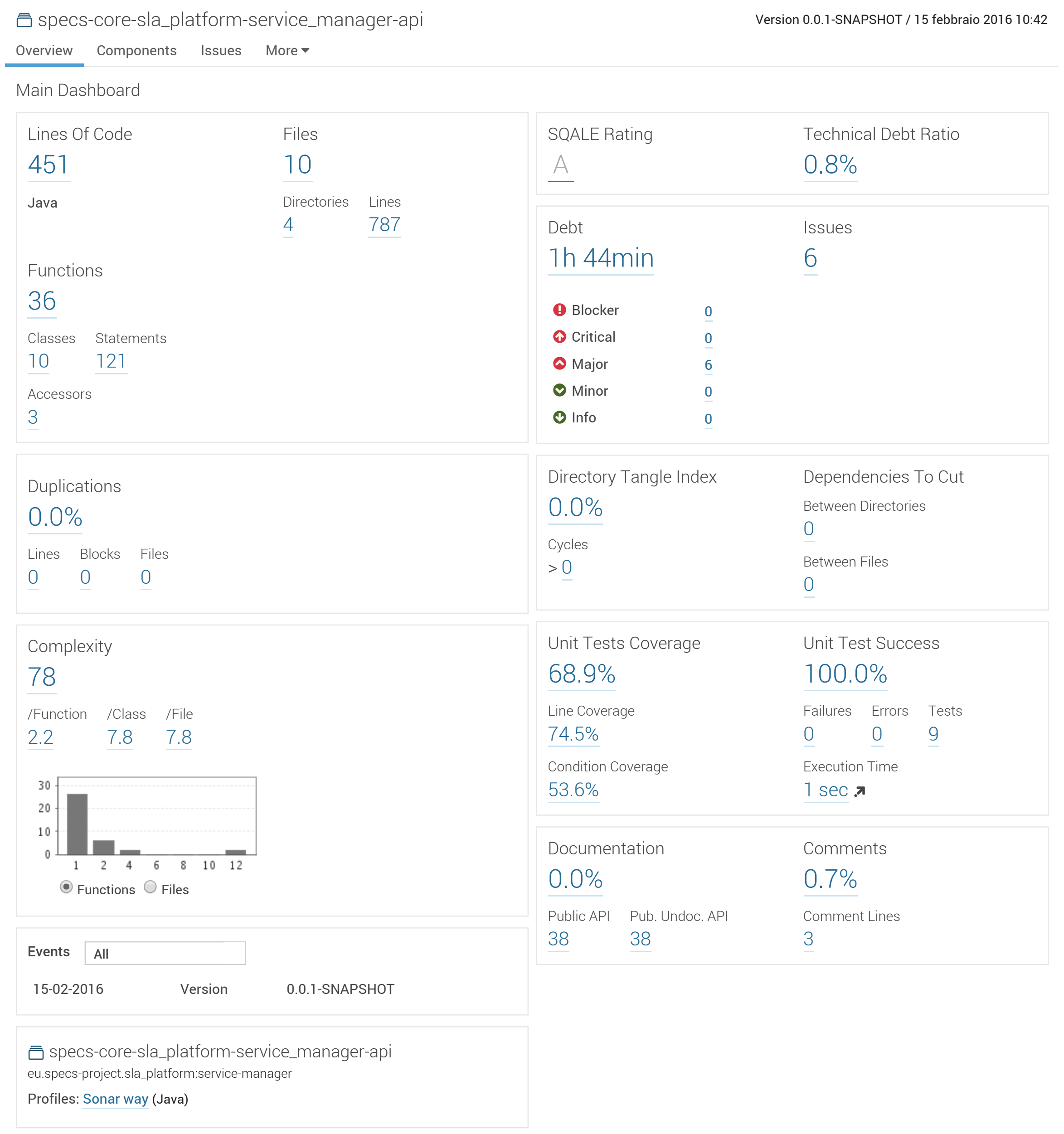
15-02-2016 Version 0.0.1-SNAPSHOT

specs-core-sla_platform-service_manager

eu.specs-project.sla_platform:service_manager

Profiles: Sonar way (Java)

Figure 12: Code Quality Analysis Report for Service Manager Backend



specs-core-sla_platform-service_manager-api

Version 0.0.1-SNAPSHOT / 15 febbraio 2016 10:42

Overview Components Issues More

Main Dashboard

Lines Of Code 451
Java

Files 10
Directories 4 Lines 787

Functions 36
Classes 10 Statements 121
Accessors 3

SQALE Rating A
Technical Debt Ratio 0.8%

Debt 1h 44min
Issues 6

Blocker	0
Critical	0
Major	6
Minor	0
Info	0

Duplications 0.0%
Lines 0 Blocks 0 Files 0

Directory Tangle Index 0.0%
Cycles > 0

Dependencies To Cut
Between Directories 0
Between Files 0

Complexity 78
/Function 2.2 /Class 7.8 /File 7.8

Functions Files

Unit Tests Coverage 68.9%
Line Coverage 74.5%
Condition Coverage 53.6%

Unit Test Success 100.0%
Failures 0 Errors 0 Tests 9
Execution Time 1 sec

Documentation 0.0%
Public API 38 Pub. Undoc. API 38

Comments 0.7%
Comment Lines 3

Events All

15-02-2016 Version 0.0.1-SNAPSHOT

specs-core-sla_platform-service_manager-api
eu.specs-project.sla_platform:service-manager
Profiles: Sonar way (Java)

Figure 13: Code Quality Analysis Report for Service Manager Frontend

4.3. Security Metrics Catalogue

The Security Metrics Catalogue is responsible for managing the Security Metrics, which are defined in Deliverable D2.2.2 and in Deliverable D4.3.2, by using a persistent data storage that is CRUD enabled.
The main design requirement imposed the possibility to:

- query the Security Metrics catalogue in order to obtain all the metrics based on their unique identifier (Reference ID);
- the EU must be allowed to obtain a metric specification represented in XML;
- the EU must be allowed to download the entire database in form of a file (raw database download).

4.3.1. Repository

The component consists of two sub-components: the *backend*, which provides SLAs management and storing features, and the *frontend*, which exposes a REST API. Both sub-components are available on Bitbucket at the following URLs:

- the backend:
  - https://bitbucket.org/specs-team/specs-core-sla_platform-security-metric-catalogue
- the frontend:
  - https://bitbucket.org/specs-team/specs-core-sla_platform-security_metric_catalogue-api

4.3.2. Description and design

As mentioned in the previous section, the Security Metrics Catalogue component consists of two sub-components. The *backend* offers the implementation of all the functions that fulfil the management of the Security Metrics (CRUD enabled functions), provided by:

- **Metric Manager Abstract Implementation**, which implements the methods for managing the Security Metrics;
- **Metric Manager SQL JPA**, which provides the functions to manage the persistence of the Security Metrics;

The *frontend* sub-component contains the implementation of the exposed REST API. As specified in D1.1.3, the Security Metrics Catalogue offers a subset of the Services API (offered by Service Manager). Such subset, i.e. the Metric Catalogue API, can be invoked to manage a catalogue of security metrics represented according to current standards. The main exposed resources are:

- **SMTs Resource:** represents a collection of Security Metrics and the management operations associated with it. In addition, it contains the REST API to manage the database file (backup and restore);
- **SMT Resource:** represents a specific created Security Metric, the management operations associated with it and the XML representation of the Security Metric.

The description and the design of the Security Metrics Catalogue component are available in Deliverable D1.4.1.

4.3.3. Installation

The installation guide covers two scenarios:

- install using precompiled binaries (SPECS recommended);
- compile and install from source (for advanced users);

4.3.3.1. Installing by using precompiled binaries

The precompiled binaries are available under the SPECS Artifact Repository [4].

Requirements

- Oracle Java JDK 7;
- SQLite 3.9.x;

- Apache Tomcat 7.0.x;

Installation steps

1. download the web application archive (war) file from the artifact repostiry :

http://ftp.specs-project.eu/public/artifacts/sla-platform/metric-catalogue/metric-catalogue-STABLE.war

2. the war file has to be deployed in the java servlet/web container
3. if Apache Tomcat 7.0.x is used, the war file needs to be copied into the "/webapps" folder inside the home directory (CATALINA_HOME) of Apache Tomcat 7.0.x.

4.3.3.2. Compiling and installing from source

In order to compile and install the Services Manager it is mandatory first to process the *backend* and afterwards the *frontend*.

Requirements

- a Git client;
- Apache Maven 3.3.x;
- Oracle Java JDK 7;
- SQLite 3.9.x;
- Apache Tomcat 7.0.x;

Backend installation steps:

1. clone the Bitbucket repository:

git clone git@bitbucket.org:specs-team/specs-core-sla_platform-security-metric-catalogue.git

2. change the configuration of the database in the *persistence.xml* file (the file is located in the folder "src/main/resource" and "/src/test/resources") in order to define the correct path where the SQLite database will be created;
3. under *specs-core-sla_platform-security-metric-catalogue* run:

mvn install

***Frontend installation steps*:**

1. clone the Bitbucket repository:

git clone git@bitbucket.org:specs-team/specs-core-sla_platform-security-metric-catalogue-api.git

2. under *specs-core-sla_platform-security-metric-catalogue-api* run:

mvn package

The *backend* installation generates the artifact used by the *frontend*. The *frontend* installation generates a web application archive (war) file, under the "/target" subfolder. In order to use the component, the war file has to be deployed in the java servlet/web container. If Apache Tomcat 7.0.x is used, the war file needs to be copied into the "/webapps" folder inside the home directory (CATALINA_HOME) of Apache Tomcat 7.0.x.

4.3.4. Usage

The Security Metrics Catalogue exposes a REST API interface. All the exposed REST resources are mapped under the path "/cloud-sla/*". The mapping rules are defined in the *web.xml* file under WEB-INF folder (*CATALINA_HOME/webapps/service-manager-api/WEB-INF/*):

```
<servlet-mapping>
    <servlet-name>Jersey REST Service</servlet-name>
```

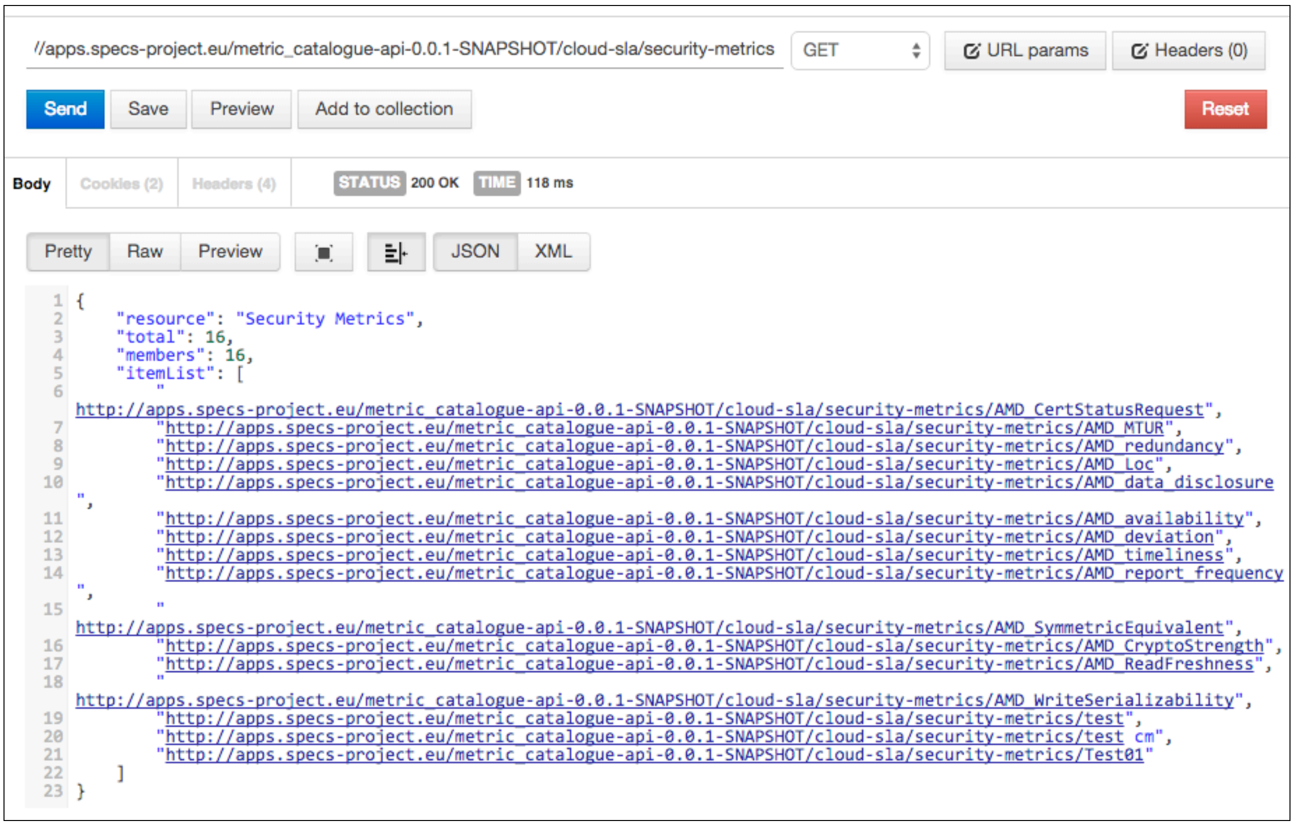
```
            <url-pattern>/cloud-sla/*</url-pattern>
        </servlet-mapping>
```

Moreover, all the REST resources are represented by specific java classes under the package "*eu.specsproject.slaplatform.metriccatalogue.restfrontend*", defined in the *web.xml* file under WEB-INF folder:

```
        <init-param>
        <param-name>jersey.config.server.provider.packages</param-name>
        <param-
value>eu.specsproject.slaplatform.metriccatalogue.restfrontend</param-
value>
        </init-param>
```

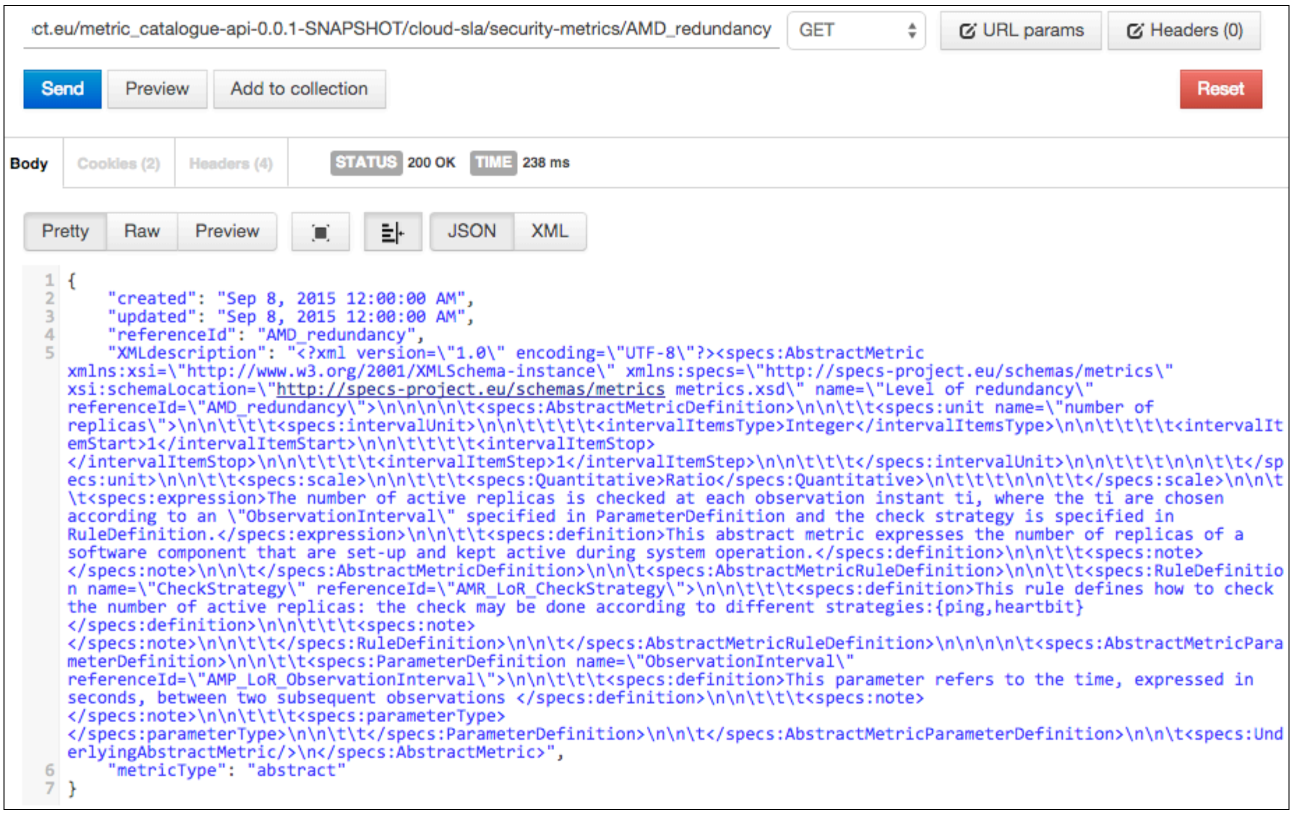
At the startup of the component, all the REST resources are configured based on the configuration parameters defined in the *web.xml* file.

Figure 14 and Figure 15 describe two examples of the REST API calls used to retrieve Security Metrics information.



```
{
    "resource": "Security Metrics",
    "total": 16,
    "members": 16,
    "itemList": [
        "
http://apps.specs-project.eu/metric_catalogue-api-0.0.1-SNAPSHOT/cloud-sla/security-metrics/AMD_CertStatusRequest",
        "http://apps.specs-project.eu/metric_catalogue-api-0.0.1-SNAPSHOT/cloud-sla/security-metrics/AMD_MTUR",
        "http://apps.specs-project.eu/metric_catalogue-api-0.0.1-SNAPSHOT/cloud-sla/security-metrics/AMD_redundancy",
        "http://apps.specs-project.eu/metric_catalogue-api-0.0.1-SNAPSHOT/cloud-sla/security-metrics/AMD_Loc",
        "http://apps.specs-project.eu/metric_catalogue-api-0.0.1-SNAPSHOT/cloud-sla/security-metrics/AMD_data_disclosure
",
        "http://apps.specs-project.eu/metric_catalogue-api-0.0.1-SNAPSHOT/cloud-sla/security-metrics/AMD_availability",
        "http://apps.specs-project.eu/metric_catalogue-api-0.0.1-SNAPSHOT/cloud-sla/security-metrics/AMD_deviation",
        "http://apps.specs-project.eu/metric_catalogue-api-0.0.1-SNAPSHOT/cloud-sla/security-metrics/AMD_timeliness",
        "http://apps.specs-project.eu/metric_catalogue-api-0.0.1-SNAPSHOT/cloud-sla/security-metrics/AMD_report_frequency
",
        "
http://apps.specs-project.eu/metric_catalogue-api-0.0.1-SNAPSHOT/cloud-sla/security-metrics/AMD_SymmetricEquivalent",
        "http://apps.specs-project.eu/metric_catalogue-api-0.0.1-SNAPSHOT/cloud-sla/security-metrics/AMD_CryptoStrength",
        "http://apps.specs-project.eu/metric_catalogue-api-0.0.1-SNAPSHOT/cloud-sla/security-metrics/AMD_ReadFreshness",
        "
http://apps.specs-project.eu/metric_catalogue-api-0.0.1-SNAPSHOT/cloud-sla/security-metrics/AMD_WriteSerializability",
        "http://apps.specs-project.eu/metric_catalogue-api-0.0.1-SNAPSHOT/cloud-sla/security-metrics/test",
        "http://apps.specs-project.eu/metric_catalogue-api-0.0.1-SNAPSHOT/cloud-sla/security-metrics/test cm",
        "http://apps.specs-project.eu/metric_catalogue-api-0.0.1-SNAPSHOT/cloud-sla/security-metrics/Test01"
    ]
}
```

Figure 14: Security Metrics Catalogue - *Get all metrics* API call example



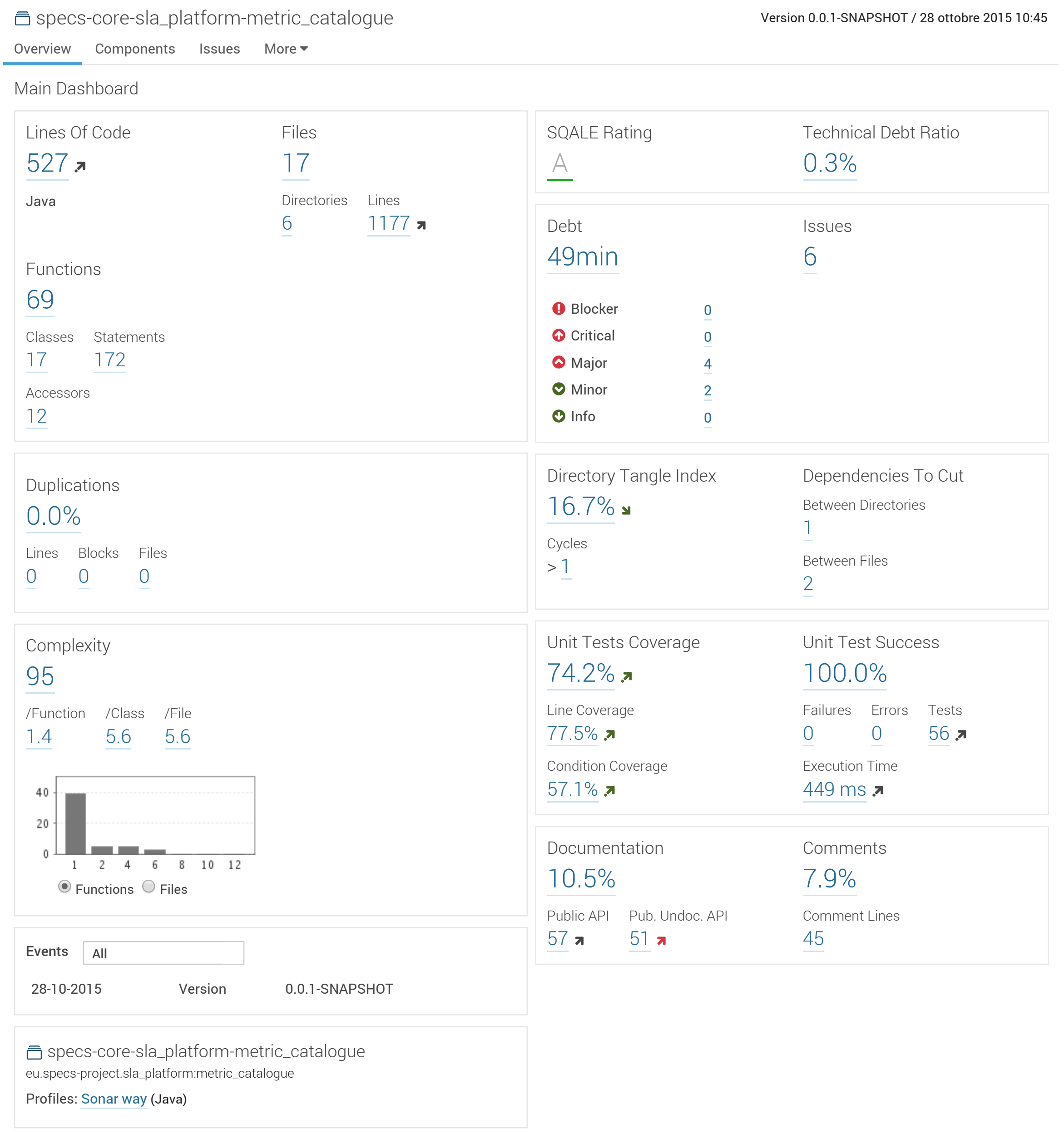
```
{
    "created": "Sep 8, 2015 12:00:00 AM",
    "updated": "Sep 8, 2015 12:00:00 AM",
    "referenceId": "AMD_redundancy",
    "XMLdescription": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><specs:AbstractMetric
xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" xmlns:specs=\"http://specs-project.eu/schemas/metrics\"
xsi:schemaLocation=\"http://specs-project.eu/schemas/metrics metrics.xsd\" name=\"Level of redundancy\"
referenceId=\"AMD_redundancy\">\n\n\n\n\t<specs:AbstractMetricDefinition>\n\n\t\t<specs:unit name=\"number of
replicas\">\n\n\t\t\t<specs:intervalUnit>\n\n\t\t\t\t<intervalItemsType>Integer</intervalItemsType>\n\n\t\t\t\t<intervalIt
emStart>1</intervalItemStart>\n\n\t\t\t\t<intervalItemStop>
</intervalItemStop>\n\n\t\t\t\t<intervalItemStep>1</intervalItemStep>\n\n\t\t\t</specs:intervalUnit>\n\n\t\t\t\n\n\t\t</sp
ecs:unit>\n\n\t\t<specs:scale>\n\n\t\t\t<specs:Quantitative>Ratio</specs:Quantitative>\n\t\t\t\n\n\t\t</specs:scale>\n\n\t
\t<specs:expression>The number of active replicas is checked at each observation instant ti, where the ti are chosen
according to an \"ObservationInterval\" specified in ParameterDefinition and the check strategy is specified in
RuleDefinition.</specs:expression>\n\n\t\t<specs:definition>This abstract metric expresses the number of replicas of a
software component that are set-up and kept active during system operation.</specs:definition>\n\n\t\t<specs:note>
</specs:note>\n\n\t</specs:AbstractMetricDefinition>\n\n\t<specs:AbstractMetricRuleDefinition>\n\n\t\t<specs:RuleDefinitio
n name=\"CheckStrategy\" referenceId=\"AMR_LoR_CheckStrategy\">\n\n\t\t\t<specs:definition>This rule defines how to check
the number of active replicas: the check may be done according to different strategies:{ping,heartbit}
</specs:definition>\n\n\t\t\t<specs:note>
</specs:note>\n\n\t\t</specs:RuleDefinition>\n\n\t</specs:AbstractMetricRuleDefinition>\n\n\n\n\t<specs:AbstractMetricPara
meterDefinition>\n\n\t\t<specs:ParameterDefinition name=\"ObservationInterval\"
referenceId=\"AMP_LoR_ObservationInterval\">\n\n\t\t\t<specs:definition>This parameter refers to the time, expressed in
seconds, between two subsequent observations </specs:definition>\n\n\t\t\t<specs:note>
</specs:note>\n\n\t\t\t<specs:parameterType>
</specs:parameterType>\n\n\t\t</specs:ParameterDefinition>\n\n\t</specs:AbstractMetricParameterDefinition>\n\n\t<specs:Und
erlyingAbstractMetric/>\n</specs:AbstractMetric>",
    "metricType": "abstract"
}
```

Figure 15: Security Metrics Catalogue - *Get a specific metric* API call example

The complete details of the REST API are described in Deliverable 1.3.

4.3.5. Tests

The JUnit framework has been used to test the component. Figure 16 and Figure 17 illustrate the code quality analysis report produced for Metric Catalogue backend and frontend. In Annex A, the tests conducted for this component are detailed with the list of the methods that have been covered and the code coverage percentage currently reached.



specs-core-sla_platform-metric_catalogue

Version 0.0.1-SNAPSHOT / 28 ottobre 2015 10:45

Overview | Components | Issues | More

Main Dashboard

Lines Of Code 527 ↗ — Java

Files 17 — Directories 6, Lines 1177 ↗

Functions 69 — Classes 17, Statements 172, Accessors 12

SQALE Rating A

Technical Debt Ratio 0.3%

Debt 49min

Issues 6

Blocker	0
Critical	0
Major	4
Minor	2
Info	0

Duplications 0.0% — Lines 0, Blocks 0, Files 0

Directory Tangle Index 16.7% ↘ — Cycles > 1

Dependencies To Cut — Between Directories 1, Between Files 2

Complexity 95 — /Function 1.4, /Class 5.6, /File 5.6

Unit Tests Coverage 74.2% ↗ — Line Coverage 77.5% ↗, Condition Coverage 57.1% ↗

Unit Test Success 100.0% — Failures 0, Errors 0, Tests 56 ↗, Execution Time 449 ms ↗

Documentation 10.5% — Public API 57 ↗, Pub. Undoc. API 51 ↗

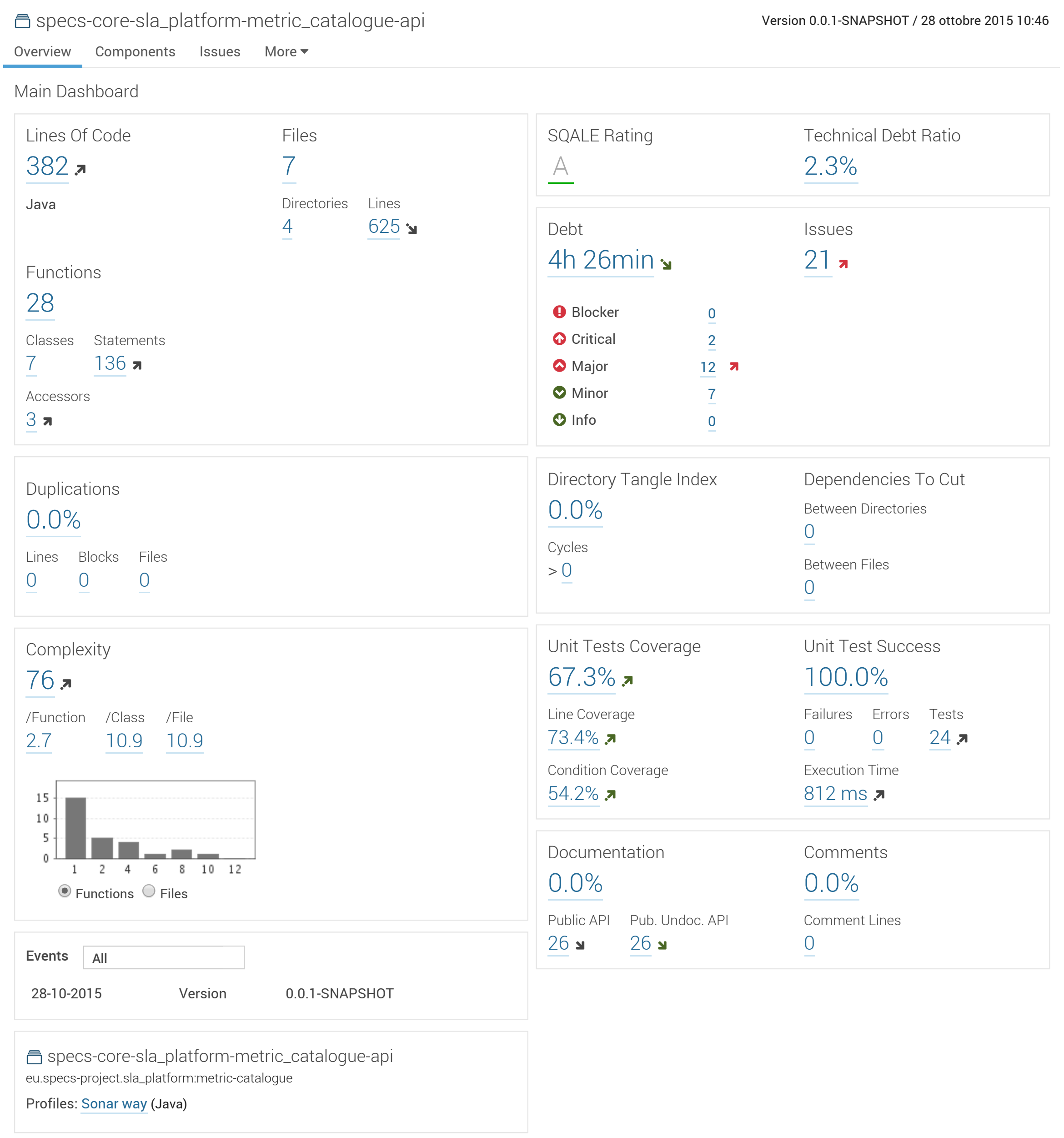
Comments 7.9% — Comment Lines 45

Events: All

28-10-2015	Version	0.0.1-SNAPSHOT

specs-core-sla_platform-metric_catalogue
eu.specs-project.sla_platform:metric_catalogue
Profiles: Sonar way (Java)

Figure 16: Code Quality Analysis Report for Metric Catalogue Backend



specs-core-sla_platform-metric_catalogue-api

Version 0.0.1-SNAPSHOT / 28 ottobre 2015 10:46

Overview Components Issues More

Main Dashboard

Lines Of Code
382
Java

Files
7
Directories 4
Lines 625

Functions
28
Classes 7
Statements 136
Accessors 3

SQALE Rating
A

Technical Debt Ratio
2.3%

Debt
4h 26min

Issues
21

Blocker	0
Critical	2
Major	12
Minor	7
Info	0

Duplications
0.0%
Lines 0
Blocks 0
Files 0

Directory Tangle Index
0.0%
Cycles
> 0

Dependencies To Cut
Between Directories
0
Between Files
0

Complexity
76
/Function 2.7
/Class 10.9
/File 10.9

Unit Tests Coverage
67.3%
Line Coverage
73.4%
Condition Coverage
54.2%

Unit Test Success
100.0%
Failures 0
Errors 0
Tests 24
Execution Time
812 ms

Documentation
0.0%
Public API 26
Pub. Undoc. API 26

Comments
0.0%
Comment Lines
0

Events All

28-10-2015 Version 0.0.1-SNAPSHOT

specs-core-sla_platform-metric_catalogue-api
eu.specs-project.sla_platform:metric-catalogue
Profiles: Sonar way (Java)

Figure 17: Code Quality Analysis Report for Metric Catalogue Frontend

4.4. *Interoperability Layer*

The Interoperability layer offers functionalities for enabling transparent communication among different modules. In practice, it acts as a gateway by intercepting all API calls and by redirecting them to the right component. This is accomplished by defining a suitable virtual interface that associates a set of API calls to a specific end-point (i.e., to a specific URL).

4.4.1. Repository

The component consists of four sub-components: *Config*, *Manager*, *RestFrontend* and *Utility*. All sub-components are maintained in the same software repository. The repository is available on Bitbucket at the following URL:

- https://bitbucket.org/specs-team/specs-core-sla_platform-interoperability

4.4.2. Description and Design

The Interoperability Layer is composed of four subcomponents: *RestFrontend*, *Config*, *Manager* and *Utility*.
The *RestFrontend* component offers the *Interoperability API* described in D1.3, needed to manage the REST resources of the Interoperability module.
Config and Manager cooperate in management of resources offered through the REST interface: in particular, the *Config* component manages the configuration parameters (e,g,, logs location, the size of the log files and the log rotation policy), while the *Manager* component offers the functionalities to configure the virtual interfaces.
The *Utility* component contains the event definition and the logic to manage it.

A detailed description of the design and architecture of the Interoperability layer is available in deliverable D1.4.1.

4.4.3. Installation

The installation guide covers two scenarios:

- Installing by using precompiled binaries (SPECS recommended);
- compiling and installing from source (for advanced users);

4.4.3.1. Installing by using precompiled binaries

The precompiled binaries are available under the SPECS Artifact Repository [4].

Requirements

- Oracle Java JDK 7;
- Apache Tomcat 7.0.x

Installation steps

1. download the web application archive (war) file from the artifact repostiry :

http://ftp.specs-project.eu/public/artifacts/sla-platform/interoperability/interoperability-api-STABLE.war

2. deploy the war file in the java servlet/web container
3. if Apache Tomcat 7.0.x is used, the war file needs to be copied into the "/webapps" folder inside the home directory (CATALINA_HOME) of Apache Tomcat 7.0.x.

4.4.3.2. Compiling and installing from source

The following are the prerequisites to compile and install the Interoperability Layer:

- a Git client;
- Apache Maven 3.3.x;
- Oracle Java JDK 7;
- Apache Tomcat 7.0.x;

Installation steps:

1. clone the Bitbucket repository:

git clone git@bitbucket.org:specs-team/specs-core-sla_platform-interoperability.git

2. under *specs-core-sla_platform-interoperability* run:

mvn package

The installation steps generate a web application archive (war) file, under the "/target" subfolder. In order to use the component, the war file has to be deployed in the java servlet/web container. If Apache Tomcat 7.0.x is used, the war file needs to be copied into the "/webapps" folder inside the home directory (CATALINA_HOME) of Apache Tomcat 7.0.x.

The installation instructions are also available on the Bitbucket repository:

- https://bitbucket.org/specs-team/specs-core-sla_platform-interoperability

4.4.4. Usage

The Interoperability component exposes the Interoperability API interface. All the exposed REST resources are mapped under the path "/cloud-sla/*". The mapping rules are defined in the *web.xml* file under WEB-INF folder (*CATALINA_HOME/webapps/service-manager-api/WEB-INF/*):

```
<servlet-mapping>
     <servlet-name>Jersey REST Service</servlet-name>
     <url-pattern>/*</url-pattern>
</servlet-mapping>
```

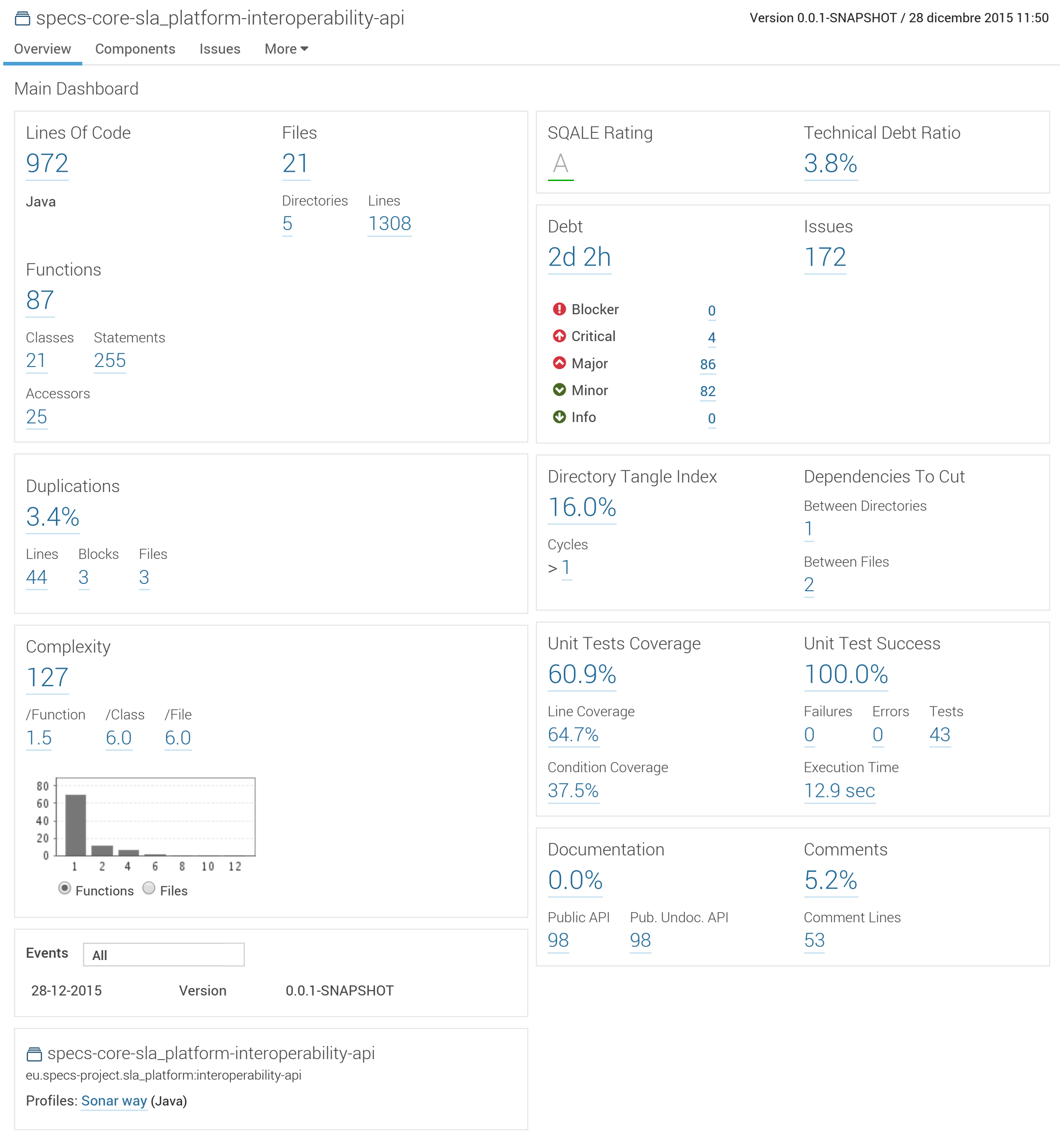
Moreover, all the REST resources are represented by specific java classes under the package "*eu.specsproject.slaplatform.slainteroperability.restfrontend*", defined in the *web.xml* file under WEB-INF folder:

```
     <init-param>
     <param-name>jersey.config.server.provider.packages</param-name>
     <param-
value>eu.specsproject.slaplatform.slainteroperability.restfrontend,
org.codehaus.jackson.jaxrs
</param-value>
     </init-param>
```

At the startup of the component, all the REST resources are configured based on the configuration parameters defined in the *web.xml* file.

4.4.5. Tests

The JUnit framework has been used to test the component. Figure 18 illustrates the code quality analysis report produced for Interoperability component. In Annex A, the tests conducted for this component are detailed with the list of the methods that have been covered and the code coverage percentage currently reached.


specs-core-sla_platform-interoperability-api

Version 0.0.1-SNAPSHOT / 28 dicembre 2015 11:50

Overview Components Issues More

Main Dashboard

Lines Of Code 972

Java

Files 21

Directories 5 Lines 1308

Functions 87

Classes 21 Statements 255

Accessors 25

SQALE Rating A

Technical Debt Ratio 3.8%

Debt 2d 2h

Issues 172

Blocker	0
Critical	4
Major	86
Minor	82
Info	0

Duplications 3.4%

Lines 44 Blocks 3 Files 3

Directory Tangle Index 16.0%

Cycles > 1

Dependencies To Cut

Between Directories 1

Between Files 2

Complexity 127

/Function 1.5 /Class 6.0 /File 6.0

Unit Tests Coverage 60.9%

Line Coverage 64.7%

Condition Coverage 37.5%

Unit Test Success 100.0%

Failures 0 Errors 0 Tests 43

Execution Time 12.9 sec

Documentation 0.0%

Public API 98 Pub. Undoc. API 98

Comments 5.2%

Comment Lines 53

Events All

28-12-2015 Version 0.0.1-SNAPSHOT

specs-core-sla_platform-interoperability-api

eu.specs-project.sla_platform:interoperability-api

Profiles: Sonar way (Java)

Figure 18: Code Quality Analysis Report for Interoperability

Vertical Layer Components

In this section, the details on implementation, installation, usage and testing of the single components belonging to the Vertical layer is given.

4.5. Auditing

The Auditing component (designed together with security mechanisms of the Enforcement module) serves as a logging/auditing component for the entire SPECS framework.

It should be noted that the Auditing component has been developed on the basis of the Contrail Auditing component[8], which has been extended and adjusted to the SPECS project's needs.

4.5.1. Repository

The component is implemented as a Maven project with two modules: audit-server and audit-client. The source code can be found on the project's BitBucket repository at the URL:
https://bitbucket.org/specs-team/specs-core-sla_platform-auditing

The component depends on the common data model classes, defined in the `specs-utility-data-model` project.

4.5.2. Description and Design

In summary, the Auditing component is made up of two subcomponents that orchestrate all logging activities:

- **Audit Server**. Provides a REST API for dealing with audit events and interacts with the underlying audit database;
- **Audit Client**. Offers Java API with convenience methods for creating and publishing audit events, which interacts in the background with the audit-server.

The initial design has been reported in D4.2.2 at M12; the final design and its validation is available in D1.4.1.

4.5.3. Installation

The project can be built from source code using Apache Maven 3 tool:

- clone the project from the BitBucket repository using a Git client:

```
git clone git@bitbucket.org:specs-team/specs-utility-auditing.git
```

- go in the specs-utility-auditing directory and run:

```
mvn package
```

4.5.3.1. Audit Server

Requirements

- Apache Tomcat 7.0.x;
- MongoDB 2.x [14];
- Oracle Java JDK 7;

[8]"*Contrail*", 2014. [Online]. Available: http://contrail-project.eu/.

The project is packaged as a web application archive (war) file with name `audit-api.war`, which has to be deployed to a Java web container. For example, to deploy the application to Apache Tomcat, just copy the war file to the Tomcat webapps directory.

The application configuration is located in the file audit-server.properties in the Java properties format. The file contains the following configuration properties:

```
mongodb.host=localhost
mongodb.port=27017
mongodb.database=auditing
```

Make the necessary changes if needed and restart the web container for changes to take effect. The Audit server should now be available at https://<host>:<port>/audit-api.

4.5.3.2. Audit Client

The Audit client can be used by other SPECS components to create and publish audit events. When using Apache Maven, the Audit client can be simply added to the project's dependencies:

```
<dependency>
     <groupId>eu.specs-project.utility</groupId>
     <artifactId>audit-client</artifactId>
     <version>0.1-SNAPSHOT</version>
</dependency>
```

Otherwise the jar file audit-client.jar has to be added to the project's library. If project was built from the source code, the audit-client.jar is located in the audit-client/target directory.
In non-Java projects where the Audit client library cannot be used, the audit events can be created in accordance with the audit event schema and published by calling Audit server REST API.
The Audit client requires the following configuration properties:

- *audit_client.audit_server_address*: audit server address
- *audit_client.keystore.path*: path of the Java key store with the component certificate
- *audit_client.keystore.password*: key store password
- *audit_client.truststore.path*: path of the Java trust store with the SPECS certificate authority certificate chain
- *audit_client.truststore.password*: trust store password

4.5.4. Usage

At the component startup, initialize the audit-client using the AuditorFactory init method by providing the properties file path:

```
voidAuditorFactory.init(StringconfigFilePath);
```

The AuditorFactory creates an Auditor instance and caches it in memory. The Auditor is now ready to accept audit events and to publish them to the audit-server.
Auditor can be retrieved from the AuditorFactory:

```
Auditorauditor=AuditorFactory.getAuditor();
```

If the Spring framework[9] is used, the Auditor can be defined in the Spring context file instead:

[9]http://projects.spring.io/spring-framework/

```
<bean id="auditor" class="eu.specsproject.utility.auditing.auditclient">
<constructor-arg name="auditServerAddress" value="
https://localhost/audit-api"/>
   ...
</bean>
```

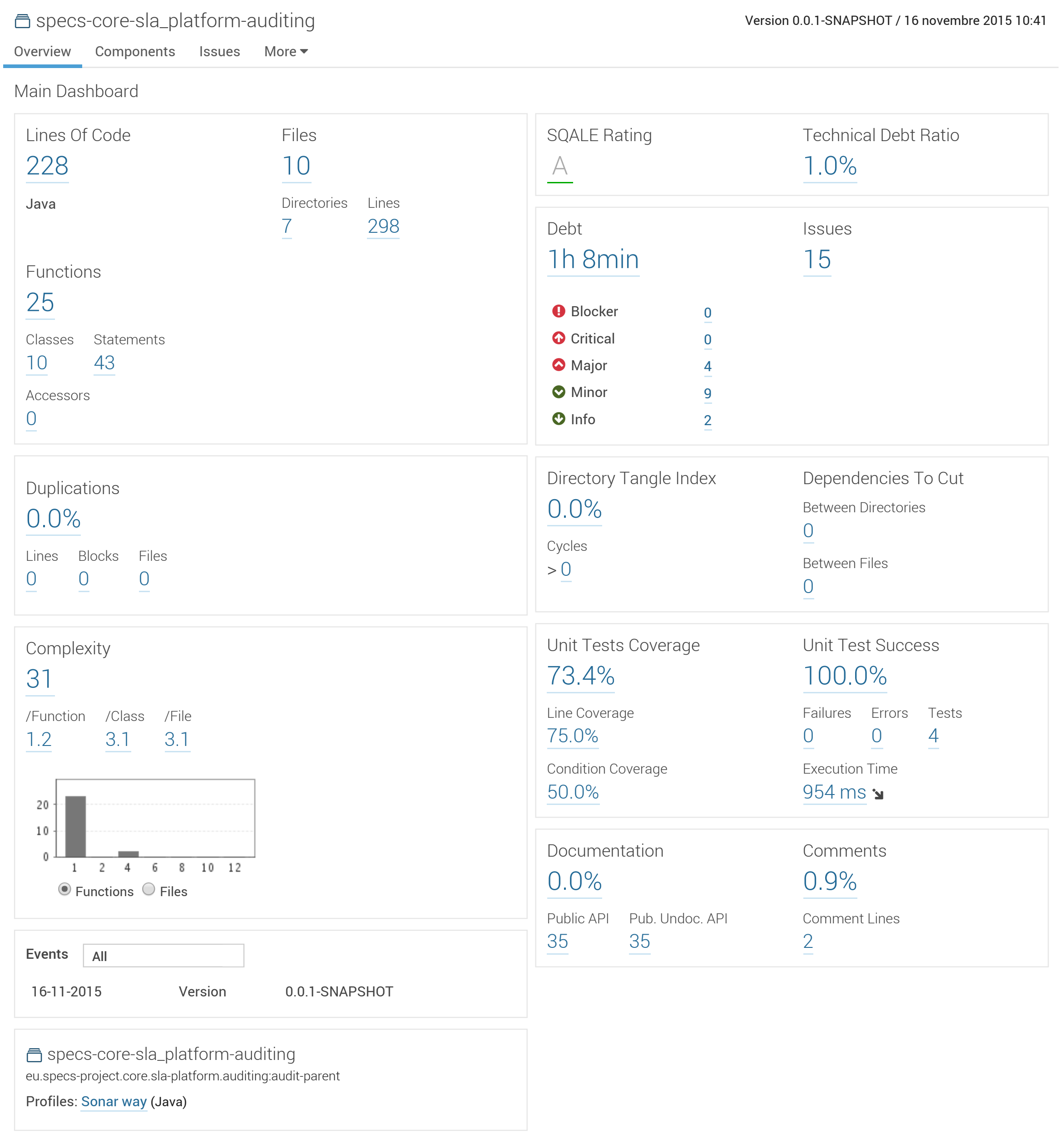
The auditor bean can then be injected using the @Autowired stereotype.

To audit an event, create an AuditEvent object, populate it with relevant data and publish it using the Auditor's audit method:

```
AuditEventauditEvent=newAuditEvent();
auditEvent.set...// set relevant data
auditor.audit(auditEvent);
```

4.5.5. Tests

The JUnit and the Spring Test frameworks have been used to test the component, as shown in Annex A. Figure 19 illustrates the code quality analysis report produced for Audit component.



specs-core-sla_platform-auditing

Version 0.0.1-SNAPSHOT / 16 novembre 2015 10:41

Overview Components Issues More

Main Dashboard

Lines Of Code
228
Java

Files
10

Directories	Lines
7	298

Functions
25

Classes	Statements
10	43

Accessors
0

SQALE Rating
A

Technical Debt Ratio
1.0%

Debt
1h 8min

Issues
15

Blocker	0
Critical	0
Major	4
Minor	9
Info	2

Duplications
0.0%

Lines	Blocks	Files
0	0	0

Directory Tangle Index
0.0%

Cycles
> 0

Dependencies To Cut
Between Directories
0
Between Files
0

Complexity
31

/Function	/Class	/File
1.2	3.1	3.1

Unit Tests Coverage
73.4%

Line Coverage
75.0%

Condition Coverage
50.0%

Unit Test Success
100.0%

Failures	Errors	Tests
0	0	4

Execution Time
954 ms

Documentation
0.0%

Public API	Pub. Undoc. API
35	35

Comments
0.9%

Comment Lines
2

Events All

16-11-2015	Version	0.0.1-SNAPSHOT

specs-core-sla_platform-auditing
eu.specs-project.core.sla-platform.auditing:audit-parent
Profiles: Sonar way (Java)

Figure 19: Code Quality Analysis Report for Audit

4.6. User Manager

The User Manager component provides a shared access control mechanism to all SPECS applications. In particular, it integrates a common authentication mechanism based on LDAP (OpenLDAP [10]) and a role-based authorization mechanism (XACML [13]).

4.6.1. Repository

The User Manager component is available on Bitbucket at the following URL:

- https://bitbucket.org/specs-team/specs-core-vertical_layer-user_manager

4.6.2. Description and Design

The User Manager component is implemented as a web application, the *SPECS Apps Manager*, which can be integrated within any other SPECS Application in order to support user management. The application uses an external authentication mechanism to manage accounts and a *Repository policy* to store the files containing the access control policies.
A detailed description and the design of the User Manager component are available in Deliverable 1.4.1.

4.6.3. Installation

In order to install the component on top of the SPECS SLA Platform, a Chef recipe is provided in the Platform Chef repository. Such recipe automates the installation and configuration of a dedicated external authentication mechanism (LDAP server) and of the policy repository.

The following paragraph covers the installation steps needed to install the component independently of the SPECS SLA Platform and provides usage information on how to test it locally. The installation guide covers two scenarios:

- Installing by using precompiled binaries (SPECS recommended);
- Compiling and installing by using sources (for advanced users);

4.6.3.1. Installing by using precompiled binaries

The precompiled binaries are available under the SPECS Artifact Repository [4].

Requirements

- Oracle Java JDK 7;
- Apache Tomcat 7.0.x

Installation steps:

1. download the web application archive (war) file from the artifact repository :

http://ftp.specs-project.eu/public/artifacts/vertical-layer/user-manager/user-manager-STABLE.war

2. the war file has to be deployed in the java servlet/web container

if Apache Tomcat 7.0.x is used, the war file needs to be copied into the "/webapps" folder inside the home directory (CATALINA_HOME) of Apache Tomcat 7.0.x.

4.6.3.2. Compiling and installing from source

Requirements for the component setup

- a Git client;
- Apache Maven 3.3.x;
- Oracle Java JDK 7;
- Apache Tomcat 7.0.x

Requirements for the authorization mechanisms

- Java JNDI [11],
- Sun XACML [13].

Requirements for the authentication mechanism

- an instance of OpenLDAP Server [10];

Installation steps for the User Manager component

1. clone the Bitbucket repository:

git clone git@bitbucket.org:specs-team/specs-core-vertical_layer-user_manager.git

2. under *specs-core-vertical_layer-user_manager* run:

mvn package

The installation steps put a web application archive (war) file under the "/target" subfolder. In order to use the component, the war file has to be deployed in the java servlet/web container. If Apache Tomcat 7.0.x is used, the war file needs to be copied into the "/webapps" folder inside the home directory (CATALINA_HOME) of Apache Tomcat 7.0.x.

After the installation of the User Manager the next step refers to the OpenLDAP Server installation.

This paragraph explains briefly what are the requirements to setup the OpenLDAP Server under the Windows and Linux environments. For tutorials that are more comprehensive, please visit OpenLDAP official website. For OpenLDAP Server installation we recommend to use the Chef based installation.

Installing a standalone OpenLDAP Server under Windows

Prerequisites:

- Visual C++ Redistributable Package [9],
- MIT Kerberos [10].

Installation package:

- the installation package available at [1];
- execute the installer file and follow the steps displayed:
  - choose the installation folder in C:\OpenLDAP;
  - check all the boxes to make sure that every feature is enabled;
- when the installation is over proceed with the configuration steps;

Installing a standalone OpenLDAP Server under Linux/Unix

- Download the appropriate binary archive from [2];
- Use the installation guide at [3];

Configuration steps

- once installed the LDAP schemas need to be customized to support SPECS configuration template:
- the *slapd.conf* file needs to be updated by including the following schema files:

```
include ./schema/cosine.schema

include ./schema/inetorgperson.schema

include ./schema/java.schema
```

- the schemas explanations:
  - *inetorgperson.schema*, in which users attributes will be defined;
  - *cosine.schema*, that it is required by *inetorperson.schema*;

  - *java.schema*, that permits to map java objects into the server.
- also update LDAP specific parameters like:

```
suffix           "dc=specs,dc=eu"
      rootdn           "cn=Manager,dc=specs,dc=eu"
      rootpw           password
      directory        ./data
```

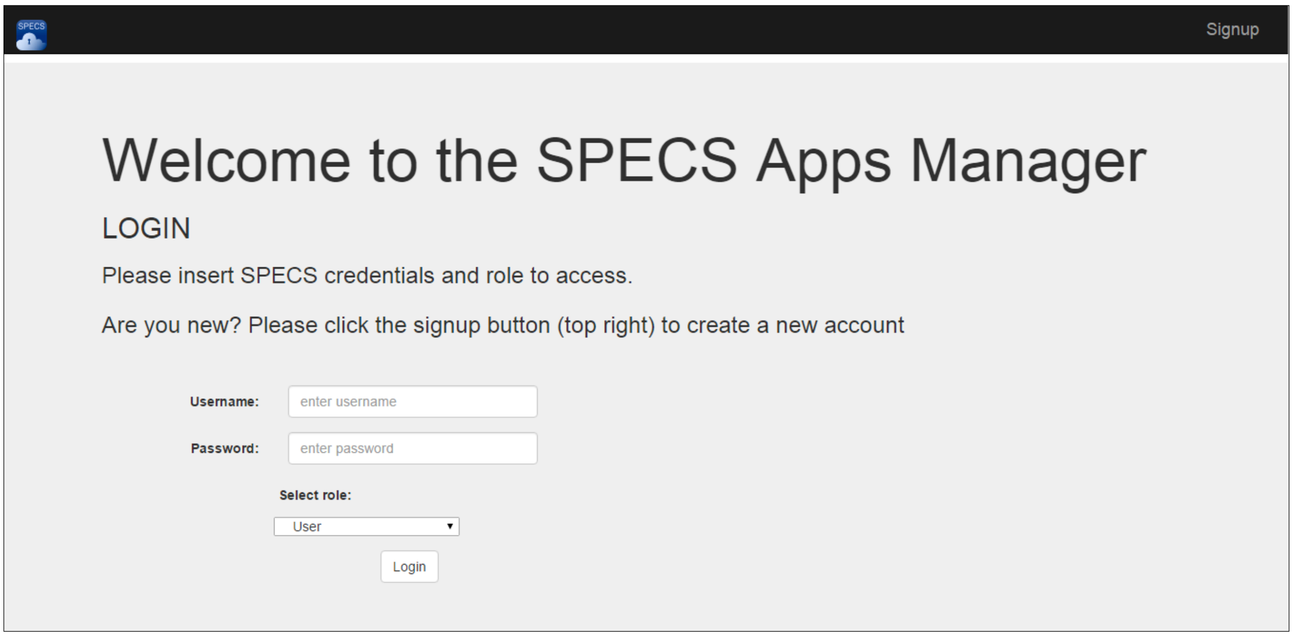
- save the configuration file and start OpenLDAP server daemon;
- create the LDIF files for each SPECS actor (users, developer and owners) using following content as an example:

```
dn: dc=specs,dc=eu
dc: specs
objectClass: domain
objectClass: dcObject
objectClass: top

dn: ou=users,dc=specs,dc=eu
objectClass: top
objectClass: organizationalUnit
ou: users
```

4.6.4. Usage

The SPECS Apps Manager allows to manage the access to SPECS Applications by providing authentication and authorization functionalities. Via the SPECS Apps Manager, the SPECS Owner can configure the access rights to the available applications. In this way, the access requests of Platform's users can be processed based on the users' roles and permissions.


Signup

Welcome to the SPECS Apps Manager

LOGIN

Please insert SPECS credentials and role to access.

Are you new? Please click the signup button (top right) to create a new account

Username: enter username

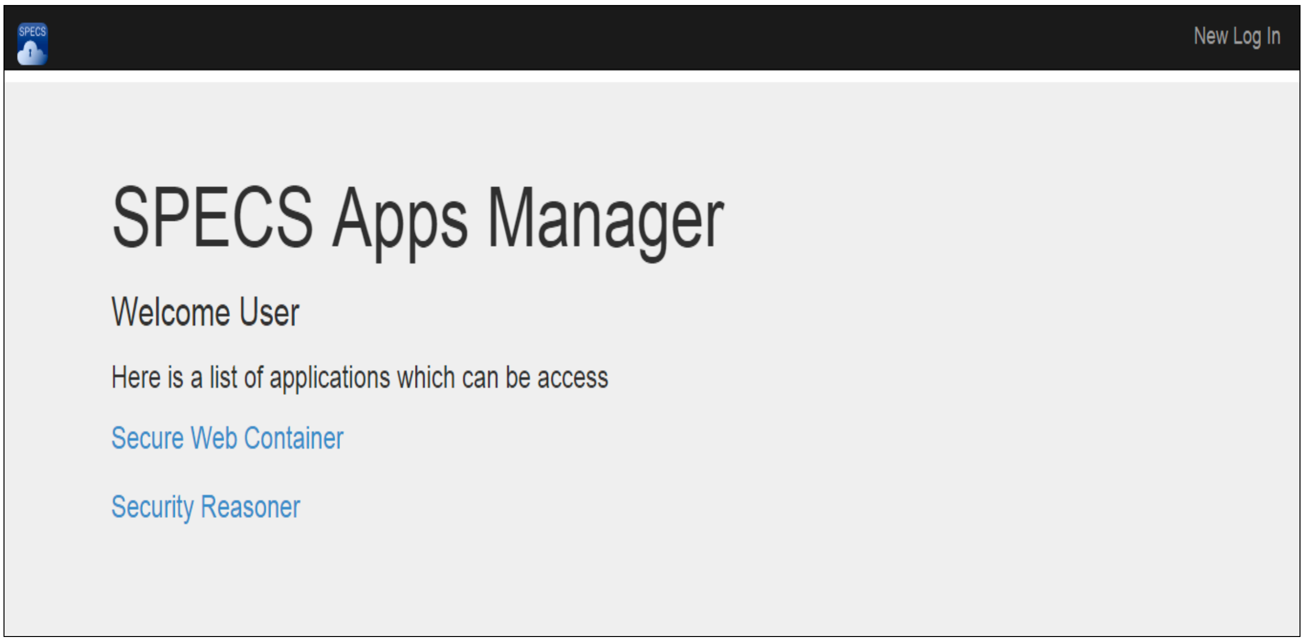
Password: enter password

Select role: User

Login

Figure 20: User Manager – The landing page

In order to access the application interface, a user must insert the SPECS Credentials (available in the LDAP repository) in the login form and click on the "Login" button. The application redirects the user to a reserved page, depending on the role, with a list of applications that the user is entitled to access [Figure 21].



SPECS Apps Manager

Welcome User

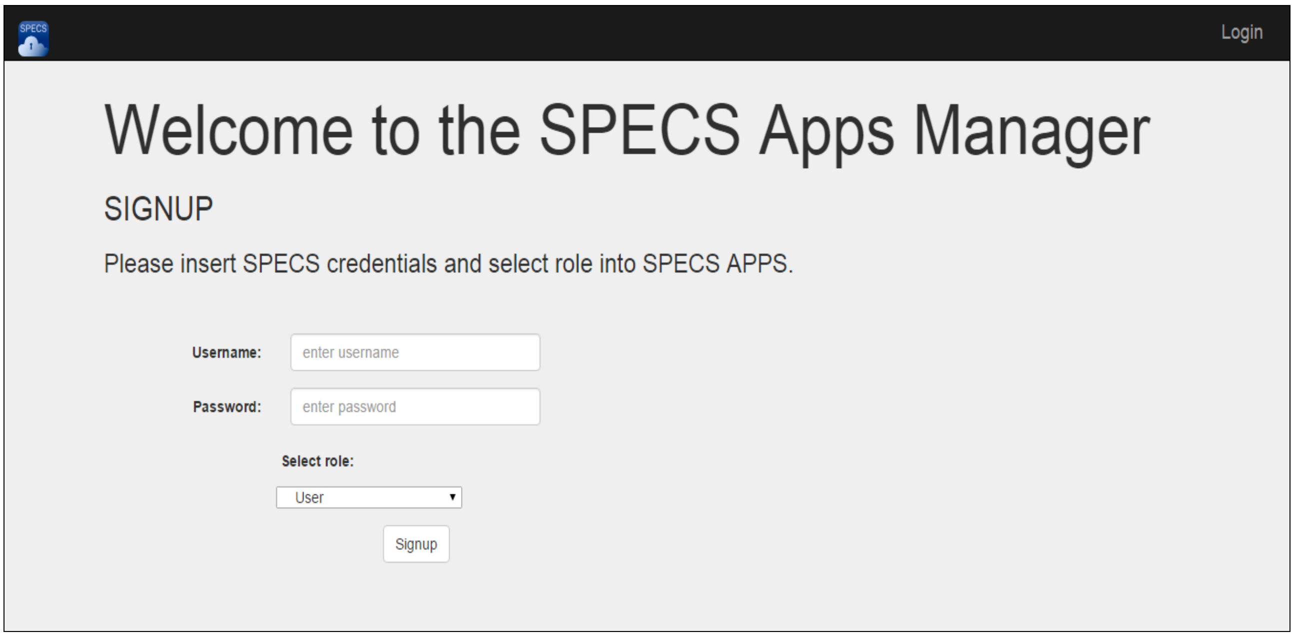
Here is a list of applications which can be access

Secure Web Container

Security Reasoner

Figure 21: User Manager - User allowed resources WUI

Moreover, a new user can click the "Signup" button (located at the top left corner) to ask for the creation of new credentials. This redirects the user to the signup module shown in **Errore. L'origine riferimento non è stata trovata.**,.



Welcome to the SPECS Apps Manager

SIGNUP

Please insert SPECS credentials and select role into SPECS APPS.

Username: enter username

Password: enter password

Select role: User

Figure 22: User Manager - New account WUI

4.6.5. Tests

The JUnit framework has been used to test the component. In Annex A, the tests conducted for this component are detailed with the list of the methods that have been covered and the code coverage percentage currently reached.

5. Conclusions

This document has presented the prototypes of the SLA Platform module and its vertical layer, as they result after the updates to requirements and components made in year 2, after the feedback from implementation and validation activities.

It should be noted that the two components of the Vertical Layer are intended to secure interactions among SPECS components, namely Credential Service and Security Tokens, although part of the Vertical Layer, are discussed in deliverables D4.4.1 and D4.4.2 of the dedicated task T4.4.

All design changes, which were due to updates in other tasks and to the feedback received from developers and integrators, are discussed in deliverable D1.4.1 and included in the current implementation. In particular, we have reported here:

- Prototypes of all SLA Platform components;
- Prototypes of two Vertical layer components.

The implementation has been completed and the four main components cover about 100% of the requirements, as reported in Section 3.1. As for the two vertical layer components, the current implementation covers about 75% of the requirements, except for the User Manager component, whose administration related functionalities are not yet implemented, as discussed in Section 3.1. At this stage we do not plan to make further improvements but, if any, they will be presented in the deliverables associated with SPECS applications within WP5.

6. References

[1] OpenLDAP Server for Windows Environments, [Online], http://www.userbooster.de/en/download/openldap-for-windows.aspx

[2] OpenLDAP Server for Linux/Unix binary repository. [Online]. ftp://ftp.openldap.org/pub/OpenLDAP/openldap-release/

[3] OpenLDAP Server for Linux/Unix Installation Guide, [Online], http://www.openldap.org/doc/admin22/install.html

[4] SPECS Artifact Repository, [Online], http://ftp.specs-project.eu/public/artifacts/

[5] Git – a version control system for software development, [Online], https://git-scm.com/

[6] Apache Maven – software project management, [Online], https://maven.apache.org/

[7] Apache Tomcat – java servlet container, [Online], https://tomcat.apache.org/

[8] jUnit – Java unit testing framework, [Online], http://junit.org/

[9] Visual C++ Redistributable Package, [Online], http://www.microsoft.com/en-us/download/details.aspx?id=5555

[10] MIT Kerberos – authentication protocol, [Online], http://web.mit.edu/kerberos/

[11] OpenLDAP – open-source directory protocol, [Online], www.openldap.org

[12] Java JNDI, [Online], http://docs.oracle.com/javase/jndi/tutorial/

[13] SUN XACML implementation, [Online], http://sunxacml.sourceforge.net/

[14] MongoDB – NoSQL database implementation, [Online], https://www.mongodb.org/

[15] Amazon AWS Documentation – Access keys, [Online], http://docs.aws.amazon.com/general/latest/gr/managing-aws-access-keys.html

[16] SPECS Testbed connection information, [Online], http://wiki.specs-project.eu/SilviuPanica/Notes/DashboardSecurityRequiredData

[17] Amazon AWS Documentation – Network and security, [Online], http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_Network_and_Security.html

[18] HP Helion Eucalyptus (formerly known as Eucalyptus Cloud), [Online], http://www8.hp.com/us/en/cloud/helion-eucalyptus-overview.html

Annex A
SLA Platform and Vertical Layer components - Unit Test

This annex briefly summarizes the results of the tests performed on the SLA Platform and on Vertical Layers. The SPECS Continuous Integration System (https://bamboo.services.ieat.ro), introduce in deliverable D4.5.2x, performs continuously these tests, updating the results every time the code changes.

A.1. SLA Manager Tests

The JUnit and the Jersey Test frameworks have been used to test the component. The following tables provide a collection of tests, which currently cover the 89,3% of the entire *specs-core-sla_platform-sla-manager* project.

Test ID	**createTest**
Test objective	The goal is to verify that the "create" method of the interface SLAManager creates and saves in persistence a SLA
Verified requirements	*SLAPL_R11, SLAPL_R27, SLAPL_R26, SLAPL_R28*
Inputs	Not null SLA
Expected results	SLA created , saved in persistence and in state of negotiating
Outputs	none
Comments	All operations executed successfully

Test ID	**retrieveTest**
Test objective	The goal is to verify that the "retrieve" method of the interface SLAManager gets a SLA from the persistence
Verified requirements	*SLAPL_R19, SLAPL_R21, SLAPL_R22*
Inputs	Not null and valid SLAIdentifier
Expected results	The SLA is obtained form the persistence
Outputs	none
Comments	All operations executed successfully

Test ID	**observedSignSlaTest**
Test objective	The goal is to verify that the SLA gets from signed to observed, so it tests the methods sign and observe
Verified requirements	SLAPL_R20, SLAPL_R31
Inputs	Not null and valid SLAIdentifier, valid SLADocument
Expected results	The state of the SLA is observed
Outputs	none
Comments	All operations executed successfully

Test ID	**observedNotSignSlaTest**
Test objective	The goal is to verify that the SLA cannot get to observed if it wasn 't in signed state. So the method returns an exception.
Verified requirements	*SLAPL_R20, SLAPL_R31*
Inputs	Not null and valid SLAIdentifier, valid SLADocument
Expected results	IllegalStateException
Outputs	none
Comments	All operations executed successfully

Test ID	**signTest**
Test objective	The goal is to verify that the SLA state gets to signed. So it tests the method sign
Verified	*SLAPL_R20, SLAPL_R31, SLAPL_R33*

requirements	
Inputs	Not null and valid SLAIdentifier, valid SLADocument
Expected results	SLA in state of signed
Outputs	none
Comments	All operations executed successfully

Test ID	**testTerminate**
Test objective	The goal is to verify that SLA gets from observed to terminated. So it tests the method sign, observe and terminate
Verified requirements	*SLAPL_R20, SLAPL_R31, SLAPL_R32*
Inputs	Valid SLAIdentifier
Expected results	SLA in state of terminated
Outputs	none
Comments	All operations executed successfully

Test ID	**testSignalAlert**
Test objective	The goal is to verify that SLA gets from observed to alerted. So it tests the method signalAlert
Verified requirements	*SLAPL_R20, SLAPL_R31*
Inputs	Valid SLAIdentifier
Expected results	SLA in state of alerted
Outputs	none
Comments	All operations executed successfully

Test ID	**testSignalViolation**
Test objective	The goal is to verify that SLA gets from observed to violated So it. So it tests the method renegotiate
Verified requirements	*SLAPL_R20, SLAPL_R31*
Inputs	Not null and valid SLAIdentifier, valid SLADocument
Expected results	SLA in state of violated
Outputs	none
Comments	All operations executed successfully

Test ID	**testComplete**
Test objective	The goal is to verify that SLA gets from observed to completed. So it tests the method complete
Verified requirements	*SLAPL_R20, SLAPL_R31, SLAPL_R29*
Inputs	Valid SLAIdentifier
Expected results	SLA in state of completed
Outputs	none
Comments	All operations executed successfully

Test ID	**testGetState**
Test objective	The goal is to verify that current SLA state is not null. So it tests the method getState
Verified requirements	*SLAPL_R20, SLAPL_R31*
Inputs	Valid SLAIdentifier
Expected results	Current state of SLA
Outputs	none
Comments	All operations executed successfully

Test ID	testGetAnnotations
Test objective	The goal is to verify that the method getAnnotations returns a list of annotations. So it tests the methods annotate and getAnnotations of the interface SLAManager
Verified requirements	*SLAPL_R19, SLAPL_R21*
Inputs	Valid SLAIdentifier
Expected results	Lista of annotations
Outputs	none
Comments	All operations executed successfully

Test ID	testSearch
Test objective	The goal is to verify that the list of SLAIdentifiers saved in persistence is not null. So it tests the method search of the interface SLAManager
Verified requirements	*SLAPL_R23*
Inputs	
Expected results	Lista of SLAidentifier
Outputs	
Comments	All operations executed successfully

Test ID	testRetrieveAndLock
Test objective	The goal is to verify that the pair SLA-Lock is not null and SLA is in the state of negotianting. So it tests the method retrieveAndLock
Verified requirements	*SLAPL_R19, SLAPL_R21, SLAPL_R22*
Inputs	Not null and valid SLAIdentifier, valid SLADocument
Expected results	Pair SLA-Lock e SLA in state of negotiating
Outputs	none
Comments	All operations executed successfully

Test ID	testUpdate
Test objective	The goal is to verify that the updated SLA is saved in persistence. So it tests the method update
Verified requirements	*SLAPL_R34*
Inputs	Valid SLAIdentifier
Expected results	SLA updated and saved in persistence
Outputs	none
Comments	All operations executed successfully

Test ID	testAnnotateException
Test objective	The goal of the test Is to verify that the method annotate generates an exception if the annotation passed as parameter is null. So it tests the method annotate
Verified requirements	*SLAPL_R25*
Inputs	Valid SLAIdentifier
Expected results	IllegalArgumentException
Outputs	none
Comments	All operations executed successfully

Test ID	testUpdateException
Test objective	The goal of the test Is to verify that the method update generates an exception if the SLA passed as parameter is null.
Verified	*SLAPL_R25*

requirements	
Inputs	Not null SLAIdentifier, null SLA
Expected results	IllegalArgumentException
Outputs	none
Comments	All operations executed successfully

Test ID	**testCreateException**
Test objective	The goal of the test is verify that the method create generates an exception id the SLA passed as parameter is null.
Verified requirements	*SLAPL_R11, SLAPL_R27, SLAPL_R26, SLAPL_R28*
Inputs	Not null SLA
Expected results	IllegalArgumentException
Outputs	none
Comments	All operations executed successfully

Test ID	**testSLADocument**
Test objective	The goal is to verify that the method "create" of the interface SLAManager creates a SLADocument
Verified requirements	*SLAPL_R11, SLAPL_R27, SLAPL_R26, SLAPL_R28*
Inputs	Valid SLADocument
Expected results	SLADocument created
Outputs	none
Comments	All operations executed successfully

Test ID	**testLock**
Test objective	The goal is to control the creation of a Lock and that his methods work correctly. So it tests the class Lock and the methods hashcode and equals.
Verified requirements	/
Inputs	
Expected results	Lock created
Outputs	none
Comments	All operations executed successfully

Test ID	**testPersistenceCreate**
Test objective	The goal is to verify that the method persistenceCreate generates an exceptions if the SLA passed as parameter is null
Verified requirements	*SLAPL_R11, SLAPL_R27, SLAPL_R26, SLAPL_R28*
Inputs	Null SLA
Expected results	IllegalArgumentException
Outputs	none
Comments	All operations executed successfully

Test ID	**testPersistenceGetByID**
Test objective	The goal is to verify that the method persistenceGetByID generates an exception if the SLAIdentifier is null
Verified requirements	*SLAPL_R19, SLAPL_R21, SLAPL_R22, , SLAPL_R23*
Inputs	Not valid SLAIdentifier
Expected results	IllegalArgumentException
Outputs	none
Comments	All operations executed successfully

Test ID	**testPersistenceGerByIDNotValid**
Test objective	The goal is to verify that the method persistenceGetByID generates an exception if the SLAIdentifier is not valid
Verified requirements	*SLAPL_R19, SLAPL_R21, SLAPL_R22*
Inputs	Not valid SLAIdentifier
Expected results	IllegalArgumentException
Outputs	none
Comments	All operations executed successfully

Test ID	**testPersistenceReleaseLock**
Test objective	The goal is to verify that the method persistenceReleaseLock releases a Lock from a SLA saved in persistence. So it tests the method persistenceReleaseLock
Verified requirements	/
Inputs	Not valid SLAIdentifier valido, valid Lock
Expected results	False
Outputs	none
Comments	All operations executed successfully

Test ID	**testCollection**
Test objective	The goal is to control the creation of a Collection and that his methods work correctly. So it tests the class Collection
Verified requirements	/
Inputs	
Expected results	Collection created
Outputs	none
Comments	All operations executed successfully

Test ID	**testItem**
Test objective	The goal is to control the creation of an Item and that his methods work correctly. So it tests the class Item
Verified requirements	/
Inputs	
Expected results	Item created
Outputs	none
Comments	All operations executed successfully

Test ID	**unMarshalTest**
Test objective	The goal is to verify that the method unmarshal converts a JSON into a MarshallingInterface.
Verified requirements	/
Inputs	JSON
Expected results	null
Outputs	none
Comments	All operations executed successfully

Test ID	**marshalTest**
Test objective	The goal is to verify that the method marshal converts a MarshallingInterface into JSON
Verified	/

requirements	
Inputs	MarshallingInterface
Expected results	Not null
Outputs	none
Comments	All operations executed successfully

Test ID	**XMLunMarshalTest**
Test objective	The goal is to verify that the method unmarshal of the class XMLEntityBuilder converts an xml into MarhallingInterface
Verified requirements	/
Inputs	String
Expected results	Null
Outputs	none
Comments	All operations executed successfully

Test ID	**XMLmarshalTest**
Test objective	The goal is to verify that the method marshal of the class XMLMarshaller converts a MarhallingInterface into xml.
Verified requirements	/
Inputs	MarshallingInterface
Expected results	Null
Outputs	none
Comments	All operations executed successfully

As for the frontend component, the following tables provide a collection of tests, which currently cover 78,5% of the entire *specs-core-sla_platform-sla_manager-api* project.

Test ID	**testSLAAPI**
Test objective	The goal is to verify the creation of SLAAPI.
Verified requirements	/
Inputs	
Expected results	SLAAPI
Outputs	none
Comments	All operations executed successfully

Test ID	**testAnnotation**
Test objective	The goal is to verify that the list of annotation is not null
Verified requirements	*SLAPL_R19, SLAPL_R21, SLAPL_R22*
Inputs	Valid SLAIdentifier, string
Expected results	List of annotations
Outputs	none
Comments	All operations executed successfully

Test ID	**testCreateAnnotation**
Test objective	The goal is to verify that the method createAnnotation generates an exception if the annotation passed as parameter is null
Verified requirements	*SLAPL_R25*
Inputs	Valid SLAIdentifier, annotation null
Expected results	IllegalArgumentException
Outputs	none
Comments	All operations executed successfully

Test ID	**testAlerted**
Test objective	The goal is to verify that SLA gets from a status to another without errors. SLA states controlled are negotiating, signed, observed and alerted. So it tests the method getSLAStatus
Verified requirements	*SLAPL_R20, SLAPL_R31*
Inputs	Valid SLAIdentifier
Expected results	SLA in state of alerted
Outputs	none
Comments	All operations executed successfully

Test ID	**testTerminate**
Test objective	The goal is to verify that SLA gets from a status to another without errors. SLA states controlled are negotiating, signed, observed and terminated. So it tests the method getSLAStatus
Verified requirements	*SLAPL_R20, SLAPL_R31, SLAPL_R32*
Inputs	Valid SLAIdentifier
Expected results	SLA in state of terminated
Outputs	none
Comments	All operations executed successfully

Test ID	**testRenegotiate**
Test objective	The goal is to verify that SLA gets from a status to another without errors. SLA states controlled are negotiating, signed, observed and violated. So it tests the method getSLAStatus
Verified requirements	*SLAPL_R30, SLAPL_R31*
Inputs	Valid SLAIdentifier
Expected results	SLA in state of violated
Outputs	none
Comments	All operations executed successfully

Test ID	**testComplete**
Test objective	The goal is to verify that SLA gets from a status to another without errors. SLA states controlled are negotiating, signed, observed and completed. So it tests the method getSLAStatus
Verified requirements	*SLAPL_R20, SLAPL_R31, SLAPL_R29*
Inputs	Valid SLAIdentifier
Expected results	SLA in state of completed
Outputs	none
Comments	All operations executed successfully

Test ID	**testAnnotationResource**
Test objective	The goal is to verify that the method getAnnotationResource returns a not null list of annotations
Verified requirements	*SLAPL_R19, SLAPL_R21*
Inputs	Valid SLA
Expected results	List of annotations
Outputs	none
Comments	All operations executed successfully

Test ID	**testUpdateSLA**

Test objective	The goal is to verify that the method update work correctly. So it test the method updateSLA
Verified requirements	*SLAPL_R34*
Inputs	Valid SLAIdentifier, string
Expected results	SLA created and saved in persistence
Outputs	none
Comments	All operations executed successfully

Test ID	**testIsWritaeble**
Test objective	The goal is to verify that the method isWriteable returns true if MarshallingInterface is passed as parameter
Verified requirements	/
Inputs	
Expected results	True
Outputs	none
Comments	All operations executed successfully

Test ID	**testIsReadable**
Test objective	The goal is to verify that the method isReadable returns true if MarshallingInterface is passed as parameter
Verified requirements	/
Inputs	
Expected results	True
Outputs	none
Comments	All operations executed successfully

Test ID	**testGetSize**
Test objective	The goal is to verify that the method getSize returns 0
Verified requirements	/
Inputs	
Expected results	0 (zero)
Outputs	none
Comments	All operations executed successfully

Test ID	**testReadFromXMLEntityBuilder**
Test objective	The goal is to verify that the method readFrom of the class SecurityProvider is called correctly. In this test case, the return will be null because there is an exception.
Verified requirements	/
Inputs	String
Expected results	null
Outputs	None
Comments	All operations executed successfully

Test ID	**testSLAsResource**
Test objective	The goal is to verify that a SLAResource is created correctly, and its then called with the method getSLAResource
Verified requirements	/
Inputs	String
Expected results	SLAResource retrieved from persistence
Outputs	None
Comments	All operations executed successfully

Test ID	**testSLAsResourceGetSLAS**
Test objective	The goal is to verify that the method getSLAS returns a not null list of SLA retrieved from persistence
Verified requirements	*SLAPL_R23*
Inputs	Valid SLAIdentifier
Expected results	List of SLA retrieved from persistence
Outputs	None
Comments	All operations executed successfully

A.2. Service Manager Tests

The JUnit and the Jersey Test frameworks have been used to test the component. The following tables provide a collection of tests,which currently cover the 71,8% of the entire *specs-core-sla_platform-service_manager* project.

Test ID	**createSMDTest**
Test objective	The goal is to verify the proper SecurityMechanismDocument creation and its possible research or updating. It covers some of declared methods of ServiceManager interface, implemented in EUServiceManagerAbstractImpl class (createSM, retrieveSM, retrieveSMMetadata, updateSM, removeSM).
Verified requirements	*SLAPL_R6*
Inputs	Json files
Expected results	It will create a new persistent *SecurityMechanismDocument.*
Outputs	The *SecurityMechanismDocument* has been created and reachable into the database.
Comments	All operations executed successfully.

Test ID	**exceptionCreateSmTest**
Test objective	The goal is to verify that the *createSM* method exception raised because of the *SecurityMechanismDocument* does not exist.
Verified requirements	*SLAPL_R6*
Inputs	No required input.
Expected results	It will raise the *IllegalArgumentException*.
Outputs	The *IllegalArgumentException* exception raised.
Comments	All operations executed as expected.

Test ID	**exceptionUpdateSmTest**
Test objective	The goal is to verify that the *updateSM* method exception raised because of the *SecurityMechanismDocument* does not exist.
Verified requirements	*SLAPL_R8*
Inputs	No required input.
Expected results	It will raise the *IllegalArgumentException*.
Outputs	The *IllegalArgumentException* exception raised.
Comments	All operations executed as expected.

Test ID	**annotateSMTest**
Test objective	The goal is to verify that the *annotateSM* method of the *EUServiceManagerAbstractImpl* class called correctly.
Verified requirements	*SLAPL_R8*
Inputs	No required input.
Expected results	It will raise the *IllegalArgumentException*.
Outputs	The *IllegalArgumentException* exception raised.
Comments	All operations executed as expected.

Test ID	**getAnnotationsSMTest**
Test objective	The goal is to verify that the *getAnnotationsSM* method of the *EUServiceManagerAbstractImpl* class called correctly.
Verified	*SLAPL_R5*

requirements	
Inputs	No required input.
Expected results	The method will be called correctly.
Outputs	The method called correctly.
Comments	All operations executed as expected.

Test ID	**updateAnnotationsSMTest**
Test objective	The goal is to verify that the *updateAnnotationsSM* method of the *EUServiceManagerAbstractImpl* class called correctly.
Verified requirements	*SLAPL_R8*
Inputs	No required input.
Expected results	The method will be called correctly.
Outputs	The method called correctly.
Comments	All operations executed as expected.

Test ID	**searchSMsTest**
Test objective	The goal is to verify that the *searchSMs* returns a correct list of *SecurityMechanismIdentifier*.
Verified requirements	*SLAPL_R9*
Inputs	No required input.
Expected results	A result list will be created and it will contain the right *SecurityMechanismIdentifier s.*
Outputs	The result list has been created and it actually contains the right *SecurityMechanismIdentifier s.*
Comments	All operations executed as expected.

Test ID	**createSCTest**
Test objective	The goal is to verify the proper *SecurityCapability* creation and its possible research or updating. It covers some declared methods of *ServiceManager* interface, implemented in *EUServiceManagerAbstractImpl* class (*createSC, retrieveSC, updateSC, removeSC*). Moreover the test checks the correct creation of a *SecurityCapalityDocument* calling all the methods of *SecurityCapalityDocument* class.
Verified requirements	*SLAPL_R6*
Inputs	No required input.
Expected results	It will create a new persistent *SecurityCapability.*
Outputs	The *SecurityCapability* has been created and reachable into a *SecurityMechanismDocument.*
Comments	All operations executed as expected.

Test ID	**exceptionCreateScTest**
Test objective	The goal is to verify that the *createSC* method exception raised because of the *SecurityCapability* does not exist.
Verified requirements	*SLAPL_R6*
Inputs	No required input.
Expected results	It will raise the *IllegalArgumentException*.
Outputs	The *IllegalArgumentException* exception raised.
Comments	All operations executed as expected.

Test ID	**exceptionUpdateScTest**
Test objective	The goal is to verify that the *updateSC* method exception raised because

	of the *SecurityCapability* does not exist.
Verified requirements	*SLAPL_R8*
Inputs	No required input.
Expected results	It will raise the *IllegalArgumentException*.
Outputs	The *IllegalArgumentException* exception raised.
Comments	All operations executed as expected.

Test ID	**annotateSCTest**
Test objective	The goal is to verify that the a*nnotateSC* method of the *EUServiceManagerAbstractImpl* class called correctly.
Verified requirements	*SLAPL_R8*
Inputs	No required input.
Expected results	The method will be called correctly.
Outputs	The method called correctly.
Comments	All operations executed as expected.

Test ID	**getAnnotationsSCTest**
Test objective	The goal is to verify that the getAnnotationsSC method of the EUServiceManagerAbstractImpl class called correctly.
Verified requirements	SLAPL_R5
Inputs	No required input.
Expected results	The method will be called correctly.
Outputs	The method called correctly.
Comments	All operations executed as expected.

Test ID	**updateAnnotationsSCTest**
Test objective	The goal is to verify that the update*AnnotationsSC* method of the *EUServiceManagerAbstractImpl* class called correctly.
Verified requirements	*SLAPL_R8*
Inputs	No required input.
Expected results	The method will be called correctly.
Outputs	The method called correctly.
Comments	All operations executed as expected.

Test ID	**unMarshalTest**
Test objective	The goal is to verify that the *unmarshal* method of *JSONentityBuilder* class acts well.
Verified requirements	/
Inputs	No required input.
Expected results	It will create a new, not-null, *MarshallingInterface.*
Outputs	A not-null *MarshallingInterface.*
Comments	All operations executed as expected.

Test ID	**marshalTest**
Test objective	The goal is to verify that the *marshal* method of *JSONmarshaller* class acts well.
Verified requirements	/
Inputs	No required input.
Expected results	It will return a new, not-null, String representing an entity.
Outputs	A not-null String.
Comments	All operations executed as expected.

Test ID	**XMLunMarshalTest**
Test objective	The goal is to verify that the un*marshal* method of *XMLentityBuilder* class acts well.
Verified requirements	/
Inputs	No required input.
Expected results	It will create a new, null, *MarshallingInterface.*
Outputs	A null *MarshallingInterface.*
Comments	All operations executed as expected.

Test ID	**XMLmarshalTest**
Test objective	The goal is to verify that the *marshal* method of *XMLmarshaller* class acts well.
Verified requirements	/
Inputs	No required input.
Expected results	It will return a new, null, String representing an entity.
Outputs	A null String.
Comments	All operations executed as expected.

The following tables provide a collection of tests, which currently cover the 68,9% of the entire *specs-core-sla_platform-service_manager-api* project.

Test ID	**smResourceTest**
Test objective	The goal is to verify the correct calling to the following *SMResource* methods: *getSM, updateSM, removeSM, getMetadata, getAnnotationResource.*
Verified requirements	*SLAPL_R5, SLAPL_R7, SLAPL_R8, SLAPL_R6*
Inputs	A *SecurityMechanismIdentifier* declared and initialized.
Expected results	The *getSM* method returns a String corresponding to a specific *SecurityMechanism.* The *updateSM* actually modified the *SecurityMechanism*. The *getMetadata* returns a *SMMetadataResource* object corresponding to a *SecurityMechanismIdentifier's* SecurityMechanism metadata. The *getAnnotationResource* returns a *SMAnnotationsResource* object corresponding to a *SecurityMechanismIdentifier's* SecurityMechanism annotations.
Outputs	The *String* has been created and not null, the *SecurityMechanism* has been modified, the returned metadata and annotations are the metadata and annotations of the *SecurityMechanism.*
Comments	All operations executed successfully.

Test ID	**smsResourceTest**
Test objective	The goal is to verify the correct calling of the *http* methods through the *SMsResource* class methods.
Verified requirements	*SLAPL_R5*
Inputs	*JerseyEmbeddedHTTPServerCrunchify* started.
Expected results	The post method (reached by url with a string representing the *SecurityMechanismIdentifier* of desidered *SecurityMechanism*) create a persistent *SecurityMechanismDocument.* The get method (reached by url with a string of desired *SecurityMechanismIdentifier*) returns a String representing the found *SecurityMechanismIdentifier*
Outputs	A string (reachable through a *Response* object) of the stored *SecurityMechanismDocument.* A string, representing a *SecurityMechanismIdentifier,* of found *SecurityMechanismDocument.*

Comments	All operations executed successfully.

Test ID	**smsResourceGetTest**
Test objective	The goal is to verify the correct calling of the *gethttp* methods through the *SMsResource* class methods.
Verified requirements	*SLAPL_R5*
Inputs	*JerseyEmbeddedHTTPServerCrunchify* started.
Expected results	The get method (reached by url) returns a String representing the found *SecurityMechanismIdentifier*
Outputs	A string, representing a *SecurityMechanismIdentifier,* of found *SecurityMechanismDocument/s.*
Comments	All operations executed successfully.

Test ID	**scResourceTest**
Test objective	The goal is to verify the correct calling to the following *SCResource* methods: *getSC, updateSC, getAnnotationResource, removeSC.*
Verified requirements	*SLAPL_R5, SLAPL_R7, SLAPL_R8, SLAPL_R6*
Inputs	A *SecurityCapabilityIdentifier* declared and initialized.
Expected results	The *getSC* method returns a String corresponding to a specific *SecurityCapability.* The *updateSC* actually modified the *SecurityCapability*. The *getAnnotationResource* returns a *SCAnnotationsResource* object corresponding to a *SecurityCapabilityIdentifier's* SecurityCapability annotations.
Outputs	The *String* has been created and not null, the *SecurityCapability* has been modified, the annotations are the *SecurityCapability*'s annotations.
Comments	All operations executed successfully.

Test ID	**scsResourceTest**
Test objective	The goal is to verify the correct calling of the *http* methods through the *SCsResource* class methods.
Verified requirements	*SLAPL_R5*
Inputs	*JerseyEmbeddedHTTPServerCrunchify* started.
Expected results	The post method (reached by url) create a persistent *SecurityCapability.* The get method (reached by url with a string representing the *SecurityCapabilityIdentifier* of desidered *SecurityCapability*) returns a String representing the found *SecurityCapabilityIdentifier.*
Outputs	A string (reachable through a *Response* object) of the stored *SecurityMechanismDocument.* A string, representing a *SecurityMechanismIdentifier,* of found *SecurityMechanismDocument.*
Comments	All operations executed successfully.

Test ID	**scsResourceGetTest**
Test objective	The goal is to verify the correct calling of the *gethttp* methods through the *SCsResource* class methods.
Verified requirements	*SLAPL_R5*
Inputs	*JerseyEmbeddedHTTPServerCrunchify* started.
Expected results	The get method (reached by url) returns a String representing the found *SecurityCapabilityIdentifier/s.*
Outputs	A string, representing a *SecurityCapabilityIdentifier/s* of found *SecurityCapability.*
Comments	All operations executed successfully.

Test ID	**scAnnotationResourceTest**
Test objective	The goal is to verify the correct calling of the *SCAnnotationResource* class methods.
Verified requirements	*SLAPL_R5*
Inputs	A *SecurityCapabilityIdentifier* declared and initialized.
Expected results	It should return *Annotations* of the *SecurityCapability.* It should update the annotation.
Outputs	An *Annotation* object representing *SecurityCapability* annotations.
Comments	All operations executed successfully.

Test ID	**smAnnotationResourceTest**
Test objective	The goal is to verify the correct calling of the *SMAnnotationResource* class methods.
Verified requirements	*SLAPL_R5*
Inputs	A *SecurityMechanismIdentifier* declared and initialized.
Expected results	It should return *Annotations* of the *SecurityMechanism.* It should update the annotation.
Outputs	An *Annotation* object representing *SecurityMechanism* annotations.
Comments	All operations executed successfully.

Test ID	**serializationProviderTest**
Test objective	The goal is to verify the correct calling of *SerializationProvider* class methods (isReadable, readFrom, getSize).
Verified requirements	/
Inputs	A *SerializationProvider* objectdeclared and initialized.
Expected results	It should return *true* at *isReadable* calling and *0* to *getSize* calling. It should return a *MarshallingInterface* at *readFrom* calling.
Outputs	*True, 0* and *a MarshallingInterface.*
Comments	All operations executed successfully.

A.3. Security Metrics CatalogueTests

The JUnit and the Jersey Test frameworks have been used to test the component. The following tables provide a collection of tests,which currently cover 74,2% of the entire *specs-core-sla_platform-metric_catalogue* project.

Test ID	**getMetricManagerIstanceTest**
Test objective	The goal is to verify the proper MetricManagerFactory creation.
Verified requirements	/
Inputs	
Expected results	It will create a new MetricManagerFactory.
Outputs	The istance of the class MetricManagerFactory is created.
Comments	All operations executed successfully

Test ID	**getMetricManagerIstanceTestWithParameters**
Test objective	The goal is to verify the proper MetricManagerFactory creation when a correct parameter is passed.
Verified requirements	/
Inputs	String unitName
Expected results	It will create a new MetricManagerFactory.
Outputs	The istance of the class MetricManagerFactory is created.
Comments	All operations executed successfully

Test ID	**EUMetricManagerSQLJPATest**
Test objective	The goal is to verify the proper EUMetricManagerSQLJPA creation.
Verified requirements	/
Inputs	String unitName
Expected results	It will create a new EUMetricManagerSQLJPA
Outputs	The istance of the class EUMetricManagerSQLJPA is created.
Comments	All operations executed successfully

Test ID	**createSMTTest**
Test objective	The goal is to verify that the method createSMT of the class EUMetricManagerAbstractImpl is called correctly.
Verified requirements	*SLAPL_R6*
Inputs	A SecurityMetricDocument (Created from an XML file)
Expected results	To create a SMT and save it in the database, returns an instance of SecurityMetricIdentifier
Outputs	An istance of SecurityMetricIdentifier
Comments	All operations executed successfully

Test ID	**retrieveSMTTest**
Test objective	The goal is to verify that the method retrieveSMT of the class EUMetricManagerAbstractImpl is called correctly. This method calls persistenceGetSMTbyID from the class EUMetricManagerSQLJPA.
Verified requirements	*SLAPL_R5*
Inputs	SecurityMetricIdentifier id
Expected results	To return the SecurityMetric related to the SecurityMetricIdentifier passed as parameter

Outputs	A SecurityMetric istance.
Comments	All operations executed successfully

Test ID	**removeSMTTest**
Test objective	The goal is to verify that the method removeSMT of the class EUMetricManagerAbstractImpl is called correctly.
Verified requirements	*SLAPL_R7*
Inputs	SecurityMetricIdentifier ID
Expected results	To remove the SecurityMetric ,related to the ID passed as a parameter, from the database
Outputs	none
Comments	All operations executed successfully

Test ID	**updateSMTTest**
Test objective	The goal is to verify that the method updateSMT of the class EUMetricManagerAbstractImpl is called correctly.
Verified requirements	*SLAPL_R8*
Inputs	A securityMetricIdentifier, a SecurityMetricDocument
Expected results	To update the securityMetric related to the ID passed as parameter, to the new metric contained in the XML file.
Outputs	None
Comments	All operations executed successfully

Test ID	**annotateSMTTest**
Test objective	The goal is to verify that the method annotateSMT of the class EUMetricManagerAbstractImpl is called correctly.
Verified requirements	*SLAPL_R8*
Inputs	SecurityMetricIdentifier ID, Object Object
Expected results	None, method is not yet implemented
Outputs	none
Comments	All operations executed successfully

Test ID	**getAnnotationsSMTtest**
Test objective	The goal is to verify that the method getAnnotationsSMT of the class EUMetricManagerAbstractImpl is called correctly.
Verified requirements	*SLAPL_R5*
Inputs	SecurityMetricIdentifier id
Expected results	None, method is not yet implemented
Outputs	none
Comments	All operations executed successfully

Test ID	**updateAnnotationsSMT**
Test objective	The goal is to verify that the method updateAnnotationsSMT of the class EUMetricManagerAbstractImpl is called correctly.
Verified requirements	*SLAPL_R8*
Inputs	SecurityMetricIdentifier Id,String AnnotationID,Object Object
Expected results	None, method is not yet implemented

Outputs	none
Comments	All operations executed successfully

Test ID	**searchSMT**
Test objective	The goal is to verify that the method searchSMT of the class EUMetricManagerAbstractImpl is called correctly. This method calls the method persistenceSearchSMT of the class EUMetricManagerSQLJPA
Verified requirements	*SLAPL_R9*
Inputs	String metricType
Expected results	A list of SecurityMetrics, related to the MetricType passed as parameter
Outputs	none
Comments	All operations executed successfully

Test ID	**getMetricsBackupTest**
Test objective	The goal is to verify that the method getMetricsBackup of the the class EUMetricManagerAbstractImpl is called correctly.
Verified requirements	/
Inputs	String DbName
Expected results	Returns null, dbName is wrong.
Outputs	none
Comments	All operations executed successfully

Test ID	**restoreMetricsBackupTest**
Test objective	The goal is to verify that the method restoreMetricsBackup of the class EUMetricManagerAbstractImpl is called correctly.
Verified requirements	/
Inputs	An inputStream dbFile, a String dbName
Expected results	Returns false, wrong input.
Outputs	none
Comments	All operations executed successfully

Test ID	**securityMetricTest**
Test objective	The goal is to verify that the constructor of the class SecurityMetric is called correctly.
Verified requirements	/
Inputs	
Expected results	An instance of the class SecurityMetric
Outputs	none
Comments	All operations executed successfully

Test ID	**getCreatedTest**
Test objective	The goal is to verify that the method getCreated of the class SecurityMetrics is called correctly.
Verified requirements	/
Inputs	
Expected results	Returns the variable “created” of the securityMetric
Outputs	none
Comments	All operations executed successfully

Test ID	**setCreatedTest**

Test objective	The goal is to verify that the method setCreated of the class SecurityMetrics is called correctly.
Verified requirements	/
Inputs	A variable Date
Expected results	To set the value of the variable "created" to the one passed as parameter
Outputs	none
Comments	All operations executed successfully

Test ID	**getUpdatedTest**
Test objective	The goal is to verify that the method getUpdated of the class SecurityMetrics is called correctly.
Verified requirements	/
Inputs	
Expected results	Returns the variable "updated" of the SecurityMetric
Outputs	none
Comments	All operations executed successfully

Test ID	**setUpdatedTest**
Test objective	The goal is to verify that the method setUpdated of the class SecurityMetrics is called correctly.
Verified requirements	/
Inputs	A variable Date
Expected results	To set the value of the variable "updated" to the one passed as parameter
Outputs	none
Comments	All operations executed successfully

Test ID	**getReferenceIdTest**
Test objective	The goal is to verify that the method getReferenceId of the class SecurityMetrics is called correctly.
Verified requirements	/
Inputs	
Expected results	Fails, there is an error in the code.
Outputs	none
Comments	All operations executed successfully

Test ID	**setReferenceIdTest**
Test objective	The goal is to verify that the method setReferenceId of the class SecurityMetrics is called correctly.
Verified requirements	/
Inputs	String referenceId
Expected results	To set the value of the variable "referenceId" to the one passed as parameter
Outputs	none
Comments	All operations executed successfully

Test ID	**getXMLDescriptionTest**
Test objective	The goal is to verify that the method getXMLdescription of the class SecurityMetrics is called correctly.
Verified requirements	/

Inputs	
Expected results	Returns the variable “XMLdescription” of the security metric
Outputs	none
Comments	All operations executed successfully

Test ID	**setXMLDescriptionTest**
Test objective	The goal is to verify that the method setXMLdescription of the class SecurityMetrics is called correctly.
Verified requirements	/
Inputs	String XMLdescription
Expected results	To set the value of the variable “xmlDescription” to the one passed as parameter
Outputs	none
Comments	All operations executed successfully

Test ID	**getMetricTypeTest**
Test objective	The goal is to verify that the method getMetricType of the class SecurityMetrics is called correctly.
Verified requirements	/
Inputs	
Expected results	Returns the variable “metricType” of the security metric
Outputs	None
Comments	All operations executed successfully

Test ID	**setMetricTypeTest**
Test objective	The goal is to verify that the method setMetricType of the class SecurityMetrics is called correctly.
Verified requirements	/
Inputs	String metricType
Expected results	To set the value of the variable “metricType” to the one passed as parameter
Outputs	None
Comments	All operations executed successfully

Test ID	**securityMetricIdentifierTest**
Test objective	The goal is to verify that the consctructor of the class SecurityMetricIdentifier is called correctly.
Verified requirements	/
Inputs	A string containing the ID
Expected results	An instance of the class SecurityMetricIdentifier
Outputs	None
Comments	All operations executed successfully

Test ID	**securityMetricIdentifierNoArgsTest**
Test objective	The goal is to verify that the consctructor of the class SecurityMetricIdentifier is called correctly.
Verified requirements	/
Inputs	String ID
Expected results	An instance of the class SecurityMetricIdentifier
Outputs	None
Comments	All operations executed successfully

Test ID	**getIdTest**
Test objective	The goal is to verify that the method getId of the class SecurityMetricIdentifier is called correctly.
Verified requirements	/
Inputs	
Expected results	Returns the value of the variable "id" of the SecurityMetricIdentifier
Outputs	none
Comments	All operations executed successfully

Test ID	**toStringTest**
Test objective	The goal is to verify that the method toString of the class SecurityMetricIdentifier is called correctly.
Verified requirements	/
Inputs	
Expected results	Returns the value of the variable "id" of the SecurityMetricIdentifier
Outputs	none
Comments	All operations executed successfully

Test ID	**setIdTest**
Test objective	The goal is to verify that the method setId of the class SecurityMetricIdentifier is called correctly.
Verified requirements	/
Inputs	String id
Expected results	To set the value of the variable "id" to the one passed as parameter
Outputs	none
Comments	All operations executed successfully

Test ID	**collectionSchemaTest**
Test objective	The goal is to verify that the constructor of the class CollectionSchema is called correctly.
Verified requirements	/
Inputs	String resource, Integer total,Integer members,List <String > itemList
Expected results	Returns an instance of the class CollectionSchema
Outputs	None
Comments	All operations executed successfully

Test ID	**collectionSchemaNoParamsTest**
Test objective	The goal is to verify that the constructor of the class CollectionSchema is called correctly.
Verified requirements	/
Inputs	
Expected results	Return an instance of the class CollectionSchema
Outputs	none
Comments	All operations executed successfully

Test ID	**getResourceTest**
Test objective	The goal is to verify that the method getResource of the class CollectionSchema is called correctly.
Verified requirements	/
Inputs	
Expected results	Returns the value of the variable "resource" of the CollectionSchema

Outputs	none
Comments	All operations executed successfully

Test ID	**setResourceTest**
Test objective	The goal is to verify that the method setResource of the class CollectionSchema is called correctly.
Verified requirements	/
Inputs	String resource
Expected results	To set the value of the variable “resource” to the one passed as parameter
Outputs	None
Comments	All operations executed successfully

Test ID	**getTotalTest**
Test objective	The goal is to verify that the method getTotal of the class CollectionSchema is called correctly.
Verified requirements	/
Inputs	
Expected results	Returns the value of the variable “total” of the CollectionSchema
Outputs	None
Comments	All operations executed successfully

Test ID	**setTotalTest**
Test objective	The goal is to verify that the method setTotal of the class CollectionSchema is called correctly.
Verified requirements	/
Inputs	Integer total
Expected results	To set the value of the variable “total” to the one passed as parameter
Outputs	none
Comments	All operations executed successfully

Test ID	**getMembersTest**
Test objective	The goal is to verify that the method getMembers of the class CollectionSchema is called correctly.
Verified requirements	/
Inputs	
Expected results	Returns the value of the variable “members” of the CollectionSchema
Outputs	none
Comments	All operations executed successfully

Test ID	**setMembersTest**
Test objective	The goal is to verify that the method setMembers of the class CollectionSchema is called correctly.
Verified requirements	/
Inputs	Integer members
Expected results	To set the value of the variable “members” to the one passed as parameter
Outputs	none
Comments	All operations executed successfully

Test ID	**getItemListTest**
Test objective	The goal is to verify that the method getItemList of the class CollectionSchema is called correctly.
Verified	/

requirements	
Inputs	
Expected results	Returns the list of the values as in the variable "itemList" of the class CollectionSchema
Outputs	none
Comments	All operations executed successfully

Test ID	**setItemListTest**
Test objective	The goal is to verify that the method setItemList of the class CollectionSchema is called correctly.
Verified requirements	/
Inputs	List <String > itemList
Expected results	To set the value of the variable "itemList" to the one passed as parameter
Outputs	none
Comments	All operations executed successfully

Test ID	**getAttributeFromPersistenceTest**
Test objective	The goal is to verify that the method getAttributeFromPersistence of the class PersistenceManager is called correctly.
Verified requirements	/
Inputs	
Expected results	Test not implemented yet
Outputs	none
Comments	All operations executed successfully

Test ID	**getSQLPathByUnitTest**
Test objective	The goal is to verify that the method getSQLPathByUnit of the class PersistenceManager is called correctly.
Verified requirements	/
Inputs	
Expected results	Test not implemented yet
Outputs	none
Comments	All operations executed successfully

Test ID	**getSQLPathByDbNameTest**
Test objective	The goal is to verify that the method getSQLPathByDbName of the class PersistenceManager is called correctly.
Verified requirements	/
Inputs	
Expected results	Test not implemented yet
Outputs	none
Comments	All operations executed successfully

Test ID	**annotationTest**
Test objective	The goal is to verify that the constructor of the class Annotation is called correctly
Verified requirements	/
Inputs	String id,String descr
Expected results	Returns an instance of the class Annotation
Outputs	None
Comments	All operations executed successfully

Test ID	**annotationTestNoParameters**

Test objective	The goal is to verify that the constructor of the class Annotation is called correctly
Verified requirements	/
Inputs	
Expected results	Returns an instance of the class Annotation
Outputs	none
Comments	All operations executed successfully

Test ID	**getIdTest**
Test objective	The goal is to verify that the method getId of the class Annotation is called correctly
Verified requirements	/
Inputs	
Expected results	Returns the value of the variable “id” of the class Annotation
Outputs	none
Comments	All operations executed successfully

Test ID	**setIdTest**
Test objective	The goal is to verify that the method setId of the class Annotation is called correctly
Verified requirements	/
Inputs	String id
Expected results	To set the value of the variable ”id” to the one passed as parameter
Outputs	none
Comments	All operations executed successfully

Test ID	**getDescrTest**
Test objective	The goal is to verify that the method getDescr of the class Annotation is called correctly
Verified requirements	/
Inputs	
Expected results	Returns the value of the variable “descr” of the class Annotation
Outputs	None
Comments	All operations executed successfully

Test ID	**setDescrTest**
Test objective	The goal is to verify that the method setDescr of the class Annotation is called correctly
Verified requirements	/
Inputs	String descr
Expected results	To set the value of the variable ”descr” to the one passed as parameter
Outputs	none
Comments	All operations executed successfully

Test ID	**securityMetricDocumentTest**
Test objective	The goal is to verify that the constructor of the class SecurityMetricDocument is called correctly.
Verified requirements	/
Inputs	String doc
Expected results	Returns an instance of the class SecurityMetricDocument
Outputs	none
Comments	All operations executed successfully

Test ID	**securityMetricDocumentTestNoParams**
Test objective	The goals is to verify that the constructor of the class SecurityMetricDocument is called correctly.
Verified requirements	
Inputs	String doc
Expected results	Returns an instance of the class SecurityMetricDocument
Outputs	none
Comments	All operations executed successfully

Test ID	**getSecurityMetricJsonDocumentTest**
Test objective	The goal is to verify that the method getSecurityMetricJsonDocument of the class SecurityMetricDocument is called correctly
Verified requirements	/
Inputs	
Expected results	Returns the value of the variable “securityMetricJsonDocument” of the class SecurityMetricDocument
Outputs	none
Comments	All operations executed successfully

Test ID	**setSecurityMetricJsonDocumentTest**
Test objective	The goal is to verify that the method setSecurityMetricJsonDocument of the class SecurityMetricDocument is called correctly
Verified requirements	/
Inputs	String document
Expected results	To set the value of the variable “securityMetricJsonDocument” to the one passed as parameter
Outputs	none
Comments	All operations executed successfully

Test ID	**unMarshalTest**
Test objective	The goal is to verify that the method unmarshal of the class JSONentityBuilder is workingCorrectly.
Verified requirements	/
Inputs	A String, MarshallingInterface.class
Expected results	Null, as a null string is passed as parameter.
Outputs	none
Comments	All operations executed successfully

Test ID	**marshalTest**
Test objective	The goal is to verify that the method marshal of the class JSONentityBuilder is workingCorrectly.
Verified requirements	/
Inputs	A marshallingInterface object, the class MarshallingInterface
Expected results	Null, converted as json object, since a null marshallingInterface is passed as parameter.
Outputs	none
Comments	All operations executed successfully

Test ID	**XMLUnMarshalTest**
Test objective	The goal is to verify that the method unmarshal of the class XMLentityBuilder is workingCorrectly.
Verified	/

requirements	
Inputs	A string and the MarshallingInterface class
Expected results	Exception is returned, parameters are not correct.
Outputs	none
Comments	All operations executed successfully

Test ID	**XMLMarshalTest**
Test objective	The goal is to verify that the method marshal of the class XMLentityBuilder is workingCorrectly.
Verified requirements	/
Inputs	A MarshallingInterface object, the MarshallingInterface class
Expected results	Null converted as json object, since the object passed as parameter is null.
Outputs	none
Comments	All operations executed successfully

The following tables provide a detailed overview of the tests, which currently cover the 67,3% of the entire *specs-core-sla_platform-metric_catalogue-api* project.

Test ID	**testGetIstance**
Test objective	The goal is to verify that the method getIstance of the class MetricApi is called correctly.
Verified requirements	/
Inputs	
Expected results	To create an instance of the class MetricAPI
Outputs	none
Comments	All operations executed successfully

Test ID	**testGetIstanceParameters**
Test objective	The goal is to verify that the method getIstance of the class MetricApi is called correctly.
Verified requirements	/
Inputs	String dbName
Expected results	To create an instance of the class MetricAPI
Outputs	None
Comments	All operations executed successfully

Test ID	**testGetManager**
Test objective	The goal is to verify that the method getManager of the class MetricApi is called correctly.
Verified requirements	/
Inputs	
Expected results	Returns an istance of the variable "managerEU" of the class MetricAPI
Outputs	none
Comments	All operations executed successfully

Test ID	**testSMTAnnotationResource**
Test objective	The goal is to verify that the constructor of the class SMTAnnotationResource is called correctly.
Verified requirements	/
Inputs	SecurityMetricIdentifier id
Expected results	To create an instance of SMTAnnotationResource

Outputs	none
Comments	All operations executed successfully

Test ID	**testGetAnnotations**
Test objective	The goal is to verify that the method getAnnotations of the class SMTAnnotationResource is called correctly.
Verified requirements	*SLAPL_R5*
Inputs	
Expected results	Uses the method getAnnotationsSMT of the manager, returns an annotation.
Outputs	none
Comments	All operations executed successfully

Test ID	**testcreateAnnotations**
Test objective	The goal is to verify that the method createAnnotations of the class SMTAnnotationResource is called correctly.
Verified requirements	*SLAPL_R6*
Inputs	
Expected results	Uses the method annotateSMT of the manager.
Outputs	none
Comments	All operations executed successfully

Test ID	**testUpdateAnnotations**
Test objective	The goal is to verify that the method updateAnnotations of the class SMTAnnotationResource is called correctly.
Verified requirements	*SLAPL_R8*
Inputs	
Expected results	Uses the method updateSMT of the manager
Outputs	none
Comments	All operations executed successfully

Test ID	**testSMTResource**
Test objective	The goal is to verify that the constructor of the class SMTResource is called correctly.
Verified requirements	*/*
Inputs	SecurityMetricIdentifier id
Expected results	To create an instance of SMTResource
Outputs	none
Comments	All operations executed successfully

Test ID	**testGetSMT**
Test objective	The goal is to verify if the method getSMT of the class SMTResource is called correctly. To get a securityMetric identified by the id of the SMTResource we are calling the method from.
Verified requirements	*SLAPL_R5*
Inputs	
Expected results	Returns a JSON containing the SecurityMetric definition
Outputs	None
Comments	All operations executed successfully

Test ID	**testUpdateSMT**
Test objective	The goal is to verify that the method updateSMT of the class

	SMTResource is called correctly. To update the securityMetric of the resource. This is done calling the manager method updateSMT.
Verified requirements	*SLAPL_R8*
Inputs	String description, an XML file
Expected results	Returns a code Response.ok
Outputs	none
Comments	All operations executed successfully

Test ID	**testRemoveSMT**
Test objective	The goal is to verify that the method removeSMT of the class SMTResource is called correctly. To remove the securityMetric of the SMTResource that calls the method. This is done calling the method removeSMT of the manager.
Verified requirements	*SLAPL_R7*
Inputs	
Expected results	Returns a code Response.ok
Outputs	none
Comments	All operations executed successfully

Test ID	**testGetAnnotationsResource**
Test objective	The goal is to verify that the method getAnnotationsResource of the class SMTResource is called correctly. To get the annotations of the SMTResource that calls the method.
Verified requirements	*SLAPL_R5*
Inputs	
Expected results	Returns a istance of SMTAnntotationResource with the same ID of the SMTResource that calls the method.
Outputs	none
Comments	All operations executed successfully

Test ID	**testMarshall**
Test objective	The goal is to verify that the method marshall of the class CollectionResource is working correctly.
Verified requirements	/
Inputs	ArrayList <String > lista
Expected results	Returns null
Outputs	none
Comments	All operations executed successfully

Test ID	**testUnMarshall**
Test objective	The goal is to verify that the method unmashall of the class CollectionResource is called correctly.
Verified requirements	/
Inputs	String collection
Expected results	Return null.
Outputs	none
Comments	All operations executed successfully

Test ID	**testIsWriteable**
Test objective	The goal is to verify that the method isWriteable of the class SerializationProvider is called correctly.

	Verifies if the class is writeable calling a Marshalling method. This is true only if the class implements MarshallingInterface
Verified requirements	/
Inputs	
Expected results	Returns true
Outputs	none
Comments	All operations executed successfully

Test ID	**testIsReadable**
Test objective	The goal is to verify that the method isReadable of the class SerializationProvider is called correctly. Verifies if the class is writeable calling a Marshalling method. This is true only if the class implements MarshallingInterface
Verified requirements	/
Inputs	
Expected results	Returns true
Outputs	none
Comments	All operations executed successfully

Test ID	**testReadFrom**
Test objective	The goal is to verify that the method readFrom of the class SerializationProvider is called correctly. Reads the inputStream passed as a parameter, then builds an item of the class defined in the parameter mediaType
Verified requirements	/
Inputs	InputStream is, MediaType Application.JSON_TYPE | MediaType Application.XML_TYPE
Expected results	Returns an object of the MediaType passed as parameter.
Outputs	none
Comments	All operations executed successfully

Test ID	**testGetSize**
Test objective	The goal is to verify that the method getSize of the class SerializationProvider is called correctly. This method is not yet implemented.
Verified requirements	/
Inputs	
Expected results	Returns 0
Outputs	None
Comments	All operations executed successfully

Test ID	**testGetSMTs**
Test objective	The goal is to verify that the the method getSMTs of the class SMTsResource is called correctly. To test this class a clientResource is created, then an URI with the query parameters is passed. The path is "/security-metrics"
Verified requirements	*SLAPL_R5*
Inputs	A query containing 4 parameters : metricType = abstract,items = 10,page= 0,length = 2
Expected results	Returns a response.ok code, and a JSON containing the CollectionSchema created.
Outputs	None
Comments	All operations executed successfully

Test ID	testGetSMTsBackup
Test objective	The goal is to verify that the method getSMTsBackup of the class SMTsResource is called correctly. To test this class a client resource is created, then an uri is passed. The path is "/backup/{database}.db" To get the backupFile, the method getMetricsBackup of MetricManager is called, and the file is put in the response.
Verified requirements	/
Inputs	The URI containing the position of the database on the pc.
Expected results	Returns a Response code ok.
Outputs	None
Comments	All operations executed successfully

Test ID	testGetSMTXml
Test objective	The goal is to verify that the method getSMTXml of the class SMTsResource is called correctly. To test this class a client Resource is created, then an uri is passed. The path is "/security-metrics/{id}.xml" To get the XML file, retrieveSMT from metricManager is called. The file path is then determined, and the file is put in the response.
Verified requirements	*SLAPL_R5*
Inputs	An Uri that contains the path to a XML resource .
Expected results	Returns a response code ok
Outputs	none
Comments	All operations executed successfully

Test ID	testGetSMTResource
Test objective	The goal is to verify that the method getSMTResource of the class SMTsResource is called correctly. To test this class a client resource is called, then an uri is passed. In particular, first a SMTResource is created, then the same resource is called through its id. To call the resource, the path "/security-metrics/{id}" is used.
Verified requirements	/
Inputs	An XML file.
Expected results	Returns a SMTResource with the same ID passed in the uri.
Outputs	none
Comments	All operations executed successfully

Test ID	testCreateSMT
Test objective	The goal is to verify that the method createSMT of the class SMTsResource is called correctly. To test this class a client resource is created, then an XML file is read and it is used as body for the post request. The path called is "/security-metrics". To create the resource, the method createSMT of the MetricManager is used, and the XML passed in the body is used to create the securityMetricDocument.
Verified requirements	*SLAPL_R6*
Inputs	An XML file
Expected results	Returns response code Created
Outputs	none

Comments	All operations executed successfully

Test ID	**testRestoreSMTBackup**
Test objective	The goal is to verify that the method restoreSMTbackup of the class SMTsResource is called correctly. To test this class a client resource is created, and the path is "/security-metrics/restore/{dbname}.db". An Inputstream is passed as the body of the post. Then the method restoreMetricsBackup of the MetricManager is called.
Verified requirements	/
Inputs	
Expected results	Returns Response code ok.
Outputs	none
Comments	All operations executed successfully

A.4. Interoperability Layer Tests

The JUnit and the Jersey Test frameworks have been used to test the component. The following tables provide a collection of tests, which currently cover the 70% of the entire *specs-core-sla_platform-sla_interoperability*project.

Test ID	**testEventClass**
Test objective	The goal is to verify that all methods of the Event class works fine. In particular this test verifies that the constructor returns an instance of the class.
Verified requirements	
Inputs	
Expected results	The constructor of the class returns the required instance correctly
Outputs	An instance of the Event class
Comments	All operations executed successfully

Test ID	**testVirtualInterfaceClass**
Test objective	The goal is to verify that all methods of the VirtualInterface class works fine. In particular this test verifies that the constructor returns an instance of the class and that all the get and set methods work properly
Verified requirements	
Inputs	The values of the attributes to call the set methods
Expected results	The constructor of the class returns the required instance correctly and all set and get methods works fine
Outputs	An instance of the VirtualInterface class that has all the attributes set with the required values
Comments	All operations executed successfully

Test ID	**testEventManagerClass**
Test objective	The goal is to verify that all methods of the EventManager class works fine. In particular this test verifies that the constructor returns an instance of the class and that all the get and set methods work properly
Verified requirements	
Inputs	The values of the attributes to call the set methods
Expected results	The constructor of the class returns the required instance correctly and all set and get methods works fine
Outputs	An instance of the EventManager class that has all the attributes set with the required values
Comments	All operations executed successfully

Test ID	**testProxyControllerClass**
Test objective	The goal is to verify that all methods of the ProxyController class works fine. In particular this test verifies that the constructor returns an instance of the class and that all the get and set methods work properly
Verified requirements	
Inputs	The values of the attributes to call the set methods
Expected results	The constructor of the class returns the required instance correctly and all set and get methods works fine
Outputs	An instance of the ProxyController class that has all the attributes set with the required values
Comments	All operations executed successfully

Test ID	getHomeTest
Test objective	The goal is to verify that the API listen on the base path returns a correct value
Verified requirements	
Inputs	Get request to base path
Expected results	The API returns the String "HOME"
Outputs	The String "HOME"
Comments	All operations executed successfully

Test ID	getInterfacesTest
Test objective	The goal is to verify that the API listen on the path "interoperability/interfaces" returns a not null collection of virtual interfaces (get request)
Verified requirements	
Inputs	Get request to "interoperability/interfaces" path
Expected results	The API returns a collection of the virtual interfaces
Outputs	The collection of virtual interfaces
Comments	All operations executed successfully

Test ID	testCreateInterface
Test objective	The goal is to verify that the API listen on the path "interoperability/interfaces" stores a new virtual interface properly (post request)
Verified requirements	
Inputs	Post request to "interoperability/interfaces" path
Expected results	A 201 Created response with the id of the stored virtual interface in the body
Outputs	201 Created
Comments	All operations executed successfully

Test ID	testUpdateInterface
Test objective	The goal is to verify that the API listen on the path "interoperability/interfaces" updates the value of the virtual interface properly (put request)
Verified requirements	
Inputs	Put request to "interoperability/interfaces" path
Expected results	A 201 Created response with the id of the updated virtual interface
Outputs	201 Created
Comments	All operations executed successfully

Test ID	testDeleteInterface
Test objective	The goal is to verify that the API listen on the path "interoperability/interfaces/{id}" deletes the value of the virtual interface properly (delete request)
Verified requirements	
Inputs	Delete request to "interoperability/interfaces/{id}" path
Expected results	A 204 No Content response
Outputs	204 No Content
Comments	All operations executed successfully

Test ID	testGetInterface
Test objective	The goal is to verify that the API listen on the path

	"interoperability/interfaces/{id}" returns the value of the virtual interface properly (get request)
Verified requirements	
Inputs	Get request to "interoperability/interfaces/{id}" path
Expected results	A 200 OK response with the value of the stored virtual interface in the body
Outputs	200 OK
Comments	All operations executed successfully

Test ID	**testEvents**
Test objective	The goal is to verify that the API listen on the path "event/events" returns a not null collection of events (get request)
Verified requirements	
Inputs	Get request to "event / events" path
Expected results	The API returns a collection of the events
Outputs	The collection of events
Comments	All operations executed successfully

Test ID	**testEventId**
Test objective	The goal is to verify that the API listen on the path "events/{id}" returns a null value for an event that doesn't exist (get request)
Verified requirements	
Inputs	Get request to "events/{id}" path
Expected results	A 200 OK response with the value null in the body
Outputs	200 OK
Comments	All operations executed successfully

Test ID	**testGetSLAs**
Test objective	The goal is to verify that the method getSLAs of the class Ins_Resource is called correctly. To test this method, first a Virtual Interface is created and the path ("/interoperability/interfaces") is called with a POST operation, to put the VirtualInterface in the contentProvider of ProxyController,then the path ("/param/slas") is called with a GET operation.
Verified requirements	
Inputs	Get request to "param/slas" path
Expected results	A 200 OK response with the value of the requested slas in the body
Outputs	200 OK
Comments	All operations executed successfully

Test ID	**testPostSLAs**
Test objective	The goal is to verify that the method createSLAs of the class Ins_Resource is called correctly. To test this method, first a Virtual Interface is created and the path ("/interoperability/interfaces") is called with a POST operation, to put the VirtualInterface in the contentProvider of ProxyController,then the path ("/param/slas") is called with a POST operation.
Verified requirements	
Inputs	Post request to "param/slas" path
Expected results	A 201 created response with the value of the id of the created sla in the body
Outputs	201 created

Comments	All operations executed successfully

Test ID	**testResource**
Test objective	The goal is to verify that the method getResource of the class In_Resource is called correctly. To test this method, the path ("/param/slas/{id}") is called with a GET operation.
Verified requirements	
Inputs	Get request to "param/slas/{id}" path
Expected results	A 200 OK response with the value of the requested sla in the body
Outputs	200 OK
Comments	All operations executed successfully

Test ID	**testPostResource**
Test objective	The goal is to verify that the method createResources of the class In_Resource is called correctly. To test this method, the path ("/param/slas/{id}") is called with a PUT operation.
Verified requirements	
Inputs	Put request to "param/slas/{id}" path
Expected results	A 200 OK response with the value of the put sla in the body
Outputs	200 OK
Comments	All operations executed successfully

Test ID	**testAlertsResource**
Test objective	The goal is to verify that the method getAlerts of the class In_AlertsResource is called correctly. To test this method, the path ("/param/slas/{id}/ alerts") is called with a GET operation.
Verified requirements	
Inputs	Get request to "param/slas/{id}/alerts" path
Expected results	A 200 OK response with the value of the alerts in the body
Outputs	200 OK
Comments	All operations executed successfully

Test ID	**testViolationsResource**
Test objective	The goal is to verify that the method getViolations of the class In_ViolationsResource is called correctly. To test this method, the path ("/param/slas/{id}/violations") is called with a GET operation.
Verified requirements	
Inputs	Get request to "param/slas/{id}/violations" path
Expected results	A 200 OK response with the value of the violations in the body
Outputs	200 OK
Comments	All operations executed successfully

Test ID	**testTeamplatesResource**
Test objective	The goal is to verify that the method getTemplates of the class In_TemplatesResource is called correctly. To test this method, the path ("/param/slas/{id}/templates") is called

	with a GET operation.
Verified requirements	
Inputs	Get request to "param/slas/{id}/ templates" path
Expected results	A 200 OK response with the value of the templates in the body
Outputs	200 OK
Comments	All operations executed successfully

Test ID	**testAssociationsResource**
Test objective	The goal is to verify that the method getAssociations of the class In_AssociationsResource is called correctly. To test this method, the path ("/param/slas/{id}/associations") is called with a GET operation.
Verified requirements	
Inputs	Get request to "param/slas/{id}/ associations" path
Expected results	A 200 OK response with the value of the templates in the body
Outputs	200 OK
Comments	All operations executed successfully

Test ID	**testAlertResourceParam**
Test objective	The goal is to verify that the method getAlert of the class In_AlertResource is called correctly. To test this method, the path ("/param/slas/{id}/alerts/{param}") is called with a GET operation.
Verified requirements	
Inputs	Get request to "param/slas/{id}/alerts/{param}" path
Expected results	A 200 OK response with the value of the requested alert in the body
Outputs	200 OK
Comments	All operations executed successfully

Test ID	**testViolationResourceParam**
Test objective	The goal is to verify that the method getViolation of the class In_ViolationResource is called correctly. To test this method, the path ("/param/slas/{id}/violations/{param}") is called with a GET operation.
Verified requirements	
Inputs	Get request to "param/slas/{id}/violations/{param}" path
Expected results	A 200 OK response with the value of the requested violation in the body
Outputs	200 OK
Comments	All operations executed successfully

Test ID	**testTemplateResourceParam**
Test objective	The goal is to verify that the method getTemplate of the class In_TemplateResource is called correctly. To test this method, the path ("/param/slas/{id}/templates/{param}") is called with a GET operation.
Verified requirements	
Inputs	Get request to "param/slas/{id}/templates/{param}" path
Expected results	A 200 OK response with the value of the requested template in the body
Outputs	200 OK
Comments	All operations executed successfully

Test ID	**testAssociationResourceParam**
Test objective	The goal is to verify that the method getAssociation of the class

	In_AssociationResource is called correctly. To test this method, the path ("/param/slas/{id}/associations/{param}") is called with a GET operation.
Verified requirements	
Inputs	Get request to "param/slas/{id}/associations/{param}" path
Expected results	A 200 OK response with the value of the requested association in the body
Outputs	200 OK
Comments	All operations executed successfully

A.5. Auditing Tests

The following tables provide a collection of tests,which currently cover 72% of all source code.

Test ID	**CompActivityServiceTest**
Test objective	Test operations of the ComponentActivity service class (create, retrieve, find Component Activity audit records)
Verified requirements	*ENF_PLAN_R6, ENF_IMPL_R8, ENF_DIAG_R9, ENF_REM_R2, ENF_AUD_R1, ENF_AUD_R2*
Inputs	A test Component Activity object.
Expected results	All operations executed successfully, the retrieved record matches the original one.
Outputs	None.
Comments	All operations executed successfully.

Test ID	**ServiceActivityServiceTest**
Test objective	Test operations of the ServiceActivity service class (create, retrieve, find Service Activity audit records)
Verified requirements	*ENF_IMPL_R6, ENF_AUD_R1, ENF_AUD_R2*
Inputs	A test Service Activity object.
Expected results	All operations executed successfully, the retrieved record matches the original one.
Outputs	None.
Comments	All operations executed successfully.

Test ID	**DiagMonEventServiceTest**
Test objective	Test operations of the DiagnosedMonitoringEvent service class (create, retrieve, find Diagnosed Monitoring Event audit records)
Verified requirements	*ENF_DIAG_R12, ENF_DIAG_R14, ENF_DIAG_R17, ENF_AUD_R1, ENF_AUD_R2*
Inputs	A test Diagnosed Monitoring Event object.
Expected results	All operations executed successfully, the retrieved record matches the original one.
Outputs	None.
Comments	All operations executed successfully.

Test ID	**RemediationResultServiceTest**
Test objective	Test operations of the RemediationResult service class (create, retrieve, find Remediation Result audit records)
Verified requirements	*ENF_AUD_R1, ENF_AUD_R2*
Inputs	A test Remediation Result object.
Expected results	All operations executed successfully, the retrieved record matches the original one.
Outputs	None.
Comments	All operations executed successfully.

Test ID	**CompActivityControllerTest**
Test objective	*ENF_PLAN_R6, ENF_IMPL_R8, ENF_DIAG_R9, ENF_REM_R2, ENF_AUD_R1, ENF_AUD_R2*

Verified requirements	ENF_PLAN_R6, ENF_IMPL_R8, ENF_DIAG_R9, ENF_REM_R2
Inputs	A test Component Activity object.
Expected results	All operations executed successfully, the retrieved record matches the original one.
Outputs	None.
Comments	All operations executed successfully.

Test ID	**ServiceActivityControllerTest**
Test objective	Test operations of the ServiceActivity controller class through REST API (create, retrieve, find Service Activity audit records)
Verified requirements	*ENF_IMPL_R6, ENF_AUD_R1, ENF_AUD_R2*
Inputs	A test Service Activity object.
Expected results	All operations executed successfully, the retrieved record matches the original one.
Outputs	None.
Comments	All operations executed successfully.

Test ID	**DiagMonEventControllerTest**
Test objective	Test operations of the DiagnosedMonitoringEvent controller class through REST API (create, retrieve, find Diagnosed Monitoring Event audit records)
Verified requirements	*ENF_DIAG_R12, ENF_DIAG_R14, ENF_DIAG_R17, ENF_AUD_R1, ENF_AUD_R2*
Inputs	A test Diagnosed Monitoring Event object.
Expected results	All operations executed successfully, the retrieved record matches the original one.
Outputs	None.
Comments	All operations executed successfully.

Test ID	**RemediationResultControllerTest**
Test objective	Test operations of the RemediationResult controller class through REST API (create, retrieve, find Remediation Result audit records)
Verified requirements	*ENF_AUD_R1, ENF_AUD_R2*
Inputs	A test Remediation Result object.
Expected results	All operations executed successfully, the retrieved record matches the original one.
Outputs	None.
Comments	All operations executed successfully.

Test ID	**AppConfigTest**
Test objective	Test application configuration loading from a file.
Verified requirements	*/*
Inputs	A test application configuration file.
Expected results	Application configuration properties are set correctly.
Outputs	None.
Comments	All operations executed as expected.