Secure Provisioning of Cloud Services
based on SLA Management

SPECS Project - Deliverable 5.1.3

Description of the validation scenarios and identification of common elements

Version no.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.:	D5.1.3
Deliverable title:	Description of the validation scenarios and identification of common elements
Deliverable nature:	Prototype
Dissemination level:	Public
Contractual delivery:	15 February 2016
Actual delivery date:	15 February 2016
Author(s):	Stefano Marrone (CeRICT), Giancarlo Capone (CeRICT)
Contributors:	Massimiliano Rak (CeRICT)
Reviewers:	Andrew Byrne (EMC), Adrian Spataru (IeAT)
Total number of pages:	33

Executive summary

This deliverable reports the status of development activities of the web container and metric catalogue application. These applications are in charge of demonstrating the feasibility of the SPECS approach; they also have the purpose of testing the entire SPECS platform and modules implementing the Validation Scenarios according to the methodology expressed in D5.1.2.

In this document, we present the final status of the development activities related to T5.1 and references to install and use the prototype applications developed in the framework of this task. All the software artefacts are available, they cover a very large number of Validation Scenarios and they solicit a great part of the SPECS platform and modules, although most of them are transparent to the SPECS end-users. In this deliverable, we focus on implementation status, provide links to repositories, and present guides for installation and usage of the components. Furthermore, in accordance with the methodology proposed in D4.5.2, unit test results are be presented.

Using SPECS, an EU of the web container application can instantiate a VM containing a generic web application and negotiate, monitor and enforce the security of such application in a transparent way. The web container application solicits the greatest part of the components demonstrating, with a single application, the feasibility and the effectiveness of the proposed SPECS solution. Being part of the Solution Portfolio, this application highlights a strong industrial interest in the results of these development and validation activities. Moreover, it is in charge of implementing the subset of the Validation Scenarios covering the greatest part of the Key Concern Items as discussed in D5.1.2.

Using SPECS, an EU of the metric catalogue application can define new security metric to adopt in existing/future SPECS applications. The metric catalogue application implements some important Validation Scenarios representing one of the most requested application in SPECS. Notwithstanding this application is not in the SPECS solution portfolio, the need of having the metric catalogue is witnessed by the number of the other SPECS applications which use it.

Table of contents

Index of figures

Index of tables

1. Introduction

This document focuses on the description of the status of development activities of the web container and the metric catalogue applications. With respect to other prototypal deliverables, this one describes SPECS applications rather than software components. SPECS applications are not in the SPECS solution but they use it to provide advanced features to the final user. Hence, describing SPECS applications as they were software components is not correct: rather than, the description and the checkpoint on the development process should focus on the following items:

- covered Validation Scenarios;
- solicited SPECS components;
- relations with SPECS portfolio solution.

This deliverable represents a complete guide to install and use both the web container and the metric catalogue. Details about the implementation, installation, usage and testing of the components are given in proper sections of this document. Furthemore, it also reports the methodological steps which can guide a developer in the costruction of a SPECS Application.

This deliverable is structured as follows. In Section 2, relationships between this deliverable and other deliverables of the project are discussed. Section 3 briefly describes the main concepts of the SPECS validation and testing process, which are fully reported in D5.1.1 and D.5.1.2. Section 4 provides the methodology to build a SPECS application. Section 5 reports the status of the development on the Web Container and Section 6 on the Metric Catalogue. Section 7 draws conclusions. Appendix A and Appendix B provide the details of the unit testing campaign respectively on the Web Container and the Metric Catalogue applications.

2. Relationship with Other Deliverables

Figure 1 represents the dependency relationships among this deliverable and other deliverables.

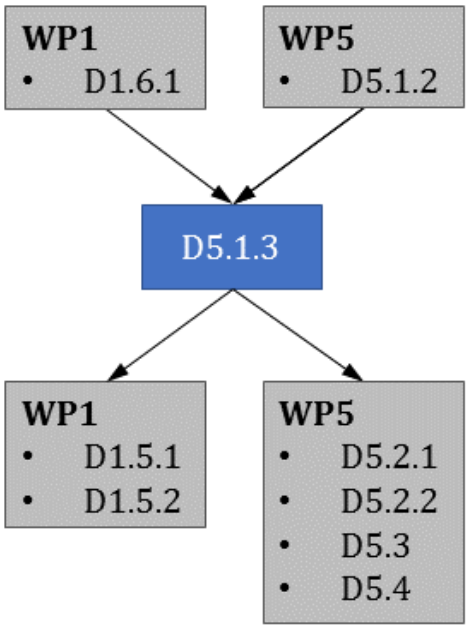

Figure 1. Relationships with other deliverables

The following deliverables are inputs for this deliverable:

- D1.6.1 provides a description of the SPECS testbed. The information about the testbed is necessary to make the testing campaign of the described applications compliant with the SPECS testbed.
- D5.1.2 fully describes the testing approach as well as providing the set of Validation Scenarios. The information contained in this deliverable are needed to map the applications described in this paper onto Validation Scenarios as well as measuring the impact of the single application on the coverage of the SPECS Key Concerns.

The following deliverables are affected by the results of this deliverable:

- D1.5.x focus on Integration Testing and, hence, could use the developed applications here described as "test script" to conduct such testing activities.
- D5.2.x, D5.3 and D5.4 focus on the industrial SPECS applications. The development process of these applications should follow the building guidelines and the way in which D5.1.3 reports application description. Moreover, the development and execution of the applications in D5.2.x, D5.3 and D5.4 may benefit from the results of this deliverable.

3. Background on the Validation Approach

The SPECS project considers different levels of testing (user, integration, unit): an overview of such levels is reported in D5.1.2. D5.1.2 also provides details about the testing methodology and the definition of the Validation Scenarios.

In brief, T5.1 defines some Validation Scenarios (VSs) used to test the SPECS platform and modules from a user perspective. Then, some Validation Applications (VAs) are defined to implement such VSs and to run them in order to verify their assumptions. Such VAs have been chosen refining the SPECS User Stories also by comparing such stories with the SPECS solution portfolio. Such comparison is very important since it validates the set of the SPECS requirements by the experience and the needs of industrial stakeholders. This deliverable focuses on two of these applications: Web Container and Metric Catalogue.

The quality of the validation approach is measured by means of ten KPIs defined in D5.1.2. Five of them are related to the execution of the Validation Applications and this deliverable uses these five KPIs to evaluate the impact of each Validation Applications among all the others. These KPIs measure which is the percentage of the covered Key Concern Items by the single VA. The KPIs are:

- EC_U: the percentage of the covered User Key Concern Items by executed VAs;
- EC_{TS}: the percentage of the covered Target Services Key Concern Items by executed VAs;
- EC_{IC}: the percentage of the covered Invocation Chain Key Concern Items by executed VAs;
- EC_{SS}: the percentage of the covered SPECS Services Key Concern Items by executed VAs;
- EC_{SLA}: the percentage of the covered SLA lifecycle Key Concern Items by executed VAs.

For more details on the entire validation approach, the Key Concern and on the validation related KPIs, see D51.2.

In this deliverable, moreover, unit tests are conducted to demonstrate not only the coverage of the Key Concerns but also of the developed source code. In particular, a testing approach based on the use of existing frameworks and oriented to branch coverage is used. The frameworks used in this deliverables are: JUnit which provide a framework for creating and executing suites of unit tests [6]; WireMock that is a library for stubbing and mocking web services [7]. Branch coverage is a widespread coverage technique which aims to test at least one time each branch of the code[4][5]: this technique requires more test with respect simpler coverage techniques as statement coverage but provide more accurate information about the expected and unexpected behaviour of the software. The coverage level has been measured using another widespread tool: SonarQube which provides a full comprehensive and user friendly dashboard of the results of a testing campaign [8]. An instance of SonarQube (i.e., an installation of the tool on a website connected to actual metrics and coverage data) is available for SPECS project [14].

4. Building a SPECS Application

SPECS applications orchestrate the SPECS core services to enable the cloud services delivery based on the SLA life cycle phases. SPECS applications are built by developers by customizing a *default SPECS application,* which includes all the functionalities needed to negotiate, implement and monitor an SLA. Since the default SPECS application automates the most part of the cloud service delivery based on SLAs, the customization consists in developing and deploying the mechanisms needed to grant the features that the SPECS Owner is willing to offer to End-users through the application.

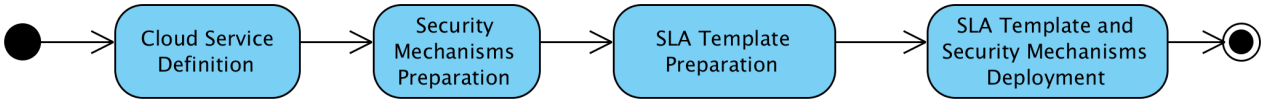

Figure 2. SPECS Application development process

Figure 2 summarizes the four main steps for the development of a new SPECS application: (i) Cloud Service Definition, (ii) Security Mechanism Preparation, (iii) SLA Template Preparation, and (iv) SLA Template and Security Mechanisms Deployment. These steps are briefly described in the following.

4.1.1. Cloud Service Definition

During this phase, the developer defines what types of cloud services to deliver through the SPECS application (e.g., web container services, storage services) and prepares related cookbooks. In order to accomplish this task, the developer has to define proper mechanisms to automatically deploy and configure the offered target services. In SPECS, such mechanisms are developed as *security mechanisms* and offer, in addition to basic cloud services, some security features that can be negotiated by the End-user.

As an example, the SPECS Secure Web Container application [2] requires the deployment of the WebPool mechanism (see D4.2.2 and D4.3.2 for details), which offers a pool of web servers over a set of virtual machines acquired from an external CSP (we used Amazon in our implementation) and is also able to provide the web resilience capability.

4.1.2. Security Mechanisms Preparation

During this phase, the developer prepares (or selects among those already available) the security mechanisms to offer on top of the cloud services defined at previous step. Each security mechanism comes with some elements: its cookbook, declaring the granted security capabilities and related security controls; the enforced/monitored security metrics: its metadata which includes all related recipes and deployment constraints. The related recipes deal with the software components implementing the mechanism while deployment constraints include incompatibility or dependency of software components implementing the mechanism.

Details on the design and implementation of security mechanisms currently available in SPECS can be found in D4.2.2, D4.3.2 and D4.4.2.

4.1.3. SLA Template Preparation

The SPECS application negotiates the services with End-users by following a process based on the WS-Agreement standard, which adopts proper templates summarizing the features that can be offered to customers. To enable negotiation, the SPECS developer has to build a WS-Agreement-compliant SLA template, which summarizes what kind of service is to be

delivered, what are the security capabilities that can be offered to End-users through the application and what are the related guarantees.

SPECS SLA templates must be written according to the Security SLA model described in Deliverable D2.2.2, which also specifies the related machine-readable format. The reported XML-based representation is an extension of the WS-Agreement schema with security-related concepts, whose definition can be found at http://www.specs-project.eu/resources/schemas/xml/SLAtemplate.xsd.

4.1.4. SLA Template and Security Mechanisms Deployment

The last development step is the deployment of the security mechanisms as well as of the SLA Template to make them available to the SPECS application.

All the prepared cookbooks must be registered with the Chef Server in order to enable the enforcement module to implement the SLA, and the mechanisms' metadata must be registered in the SLA Platform in order to enable the SPECS application to retrieve the information and to implement the SLA.

The SPECS Owner, which will offer the SPECS application services to End-Users, uses the SPECS Core Enforcement repository to share the cookbooks with the Chef Server and to enable the SPECS application execution (further details on this process can be found in D4.3.2).

To conclude, it is worth noticing that the SPECS application development mainly focuses on the development of ad-hoc Chef cookbooks for the security mechanisms to be offered. When cookbooks are already available, the only additional work consists in the preparation of the metadata and of the SLA templates used to automate the SLA implementation.

Mechanisms' metadata is defined according to the format discussed in D1.3 (related to the Enforcement API). An example of metadata, defined for the WebPool mechanism, can be found in D4.3.2, while an example of SLA template related to the SPECS Secure Web Container Application is reported online at the address http://www.specs-project.eu/resources/specs-security-sla-model/.

5. The Web Container

A web developer, representing the End-User of this user story, aims at acquiring a web container, to run his/her own application, which fulfils some security requirements. The web container is represented by one or more Virtual Machines (VMs) provided by one or more IaaS CSPs. It is reasonable to suppose that the End-User is not an expert in security field: she/he is aware of the technologies that may be involved (SSL, authentication and authorization protocols and so on), but she/he is not aware of the best practices and of how to protect her/his application from malicious attacks. For this reason, the acquisition of VMs and the enforcement of security features are accomplished through SPECS.

5.1. Development Final Results

The development of the application is completed, and a preliminary testing campaign has been successfully accomplished. It is now possible to use the "alpha" release of the application, deploying it from the repository website.

According to D5.1.2, the web container application covers the Validation Scenarios reported in Table 1 with the indication of the percentage of coverage.

	% of coverage
Secure_Web_Container_Selection	100%
Secure_Web_Container_Brokering	100%
Secure_Web_Container_TLS_enhanced	100%
Secure_Web_Container_SVA_enhanced_alert	100%
Secure_Web_Container_TLS_SVA_enhanced_violation	100%
Secure_Web_Container_TLS_multitenancy	100%
Secure_Web_Container_Web_Pool_Replication_enhanced_alert	100%
Secure_Web_Container_Web_Pool_Replication_enhanced_violation	100%

Table 1. Mapping Web Container on VSs (excerpt from D5.1.2)

The SPECS components solicited by the VA are here listed (see D5.12 for details):

- Core Modules
  - Negotiation: SLO Manager, Supply Chain Manager, Security Reasoner
  - Enforcement: Planning, Implementation, Diagnosis, RDS, Broker
  - Monitoring: Event Hub, Event Aggregator, Event Archiver, Monitoring Policy Filter, SLO Metric Exporter
- Security Mechanisms: Web Pool, TLS, SVA, DoS.

Table 2 reports an excerpt form a table of D5.1.2 showing how the Web Container application is a refinement of the homonymous User Story and is into one-to-one relation with the Secure Web Container in the solution portfolio.

Validation Applications	User Stories	solution portfolio
Web Container	Web Container	Secure Web Container

Table 2. Mapping among VAs, solution portfolio and User Stories (excerpt from D5.1.2)

Here, a brief discussion of the impact of this Validation Application is reported. The objective is to evaluate how much of the validation-related KPIs (defined in D5.1.2) is covered by the Web Container. To accomplish to this task, Table 3 reports on the row the KPIs that are related to the execution of VAs. The table has two columns: the first reports which should be the value of the KPIs if Web Container was the only available application; the second reports these values considering all the VAs.

	Only Web Container	**All VAs**
EC_U	16.67%	66.67%
EC_{TS}	85.7%	100%
EC_{IC}	33.33%	100%
EC_{SLA}	73.6%	78.9%
EC_{SS}	58%	100%

Table 3. Impact of Web Container on the Validation KPIs

5.2. Application Description

A web developer, representing the End-User of this user story, aims at acquiring a web container, to run his/her own application, which fulfils some security requirements. The web container is represented by one or more Virtual Machines (VMs) provided by one or more IaaS CSPs. It is reasonable to suppose that the End-User is not an expert in security field: she/he is aware of the technologies that may be involved (SSL, authentication and authorization protocols and so on), but she/he is not aware of the best practices and of how to protect her/his application from malicious attacks. For this reason, the acquisition of VMs and the enforcement of security features are accomplished through SPECS.

By means of the SPECS framework, it would be possible to:

- Offer a single interface to select among different offerings on different providers;
- Enable web developer to specify the needed security capabilities on the target web container explicitly, selecting the security controls
- Automatically configure the VM(s) in order to enforce the security controls requested
- Offers a set of security metrics in order to concretely monitor the respect of the security requests
- Automatically remediate to (some of) alerts and violation that may occur to the SLA associated with the web container

To develop this application, the phases of the process described in Section 4 have been followed:

1. **Cloud Service Definition,** Figure 3 shows the UML Activity Diagram that explains the flow starting from the selection of the services the End-User wants to acquire to the monitoring of the metrics he has negotiated and signed into the SLA;

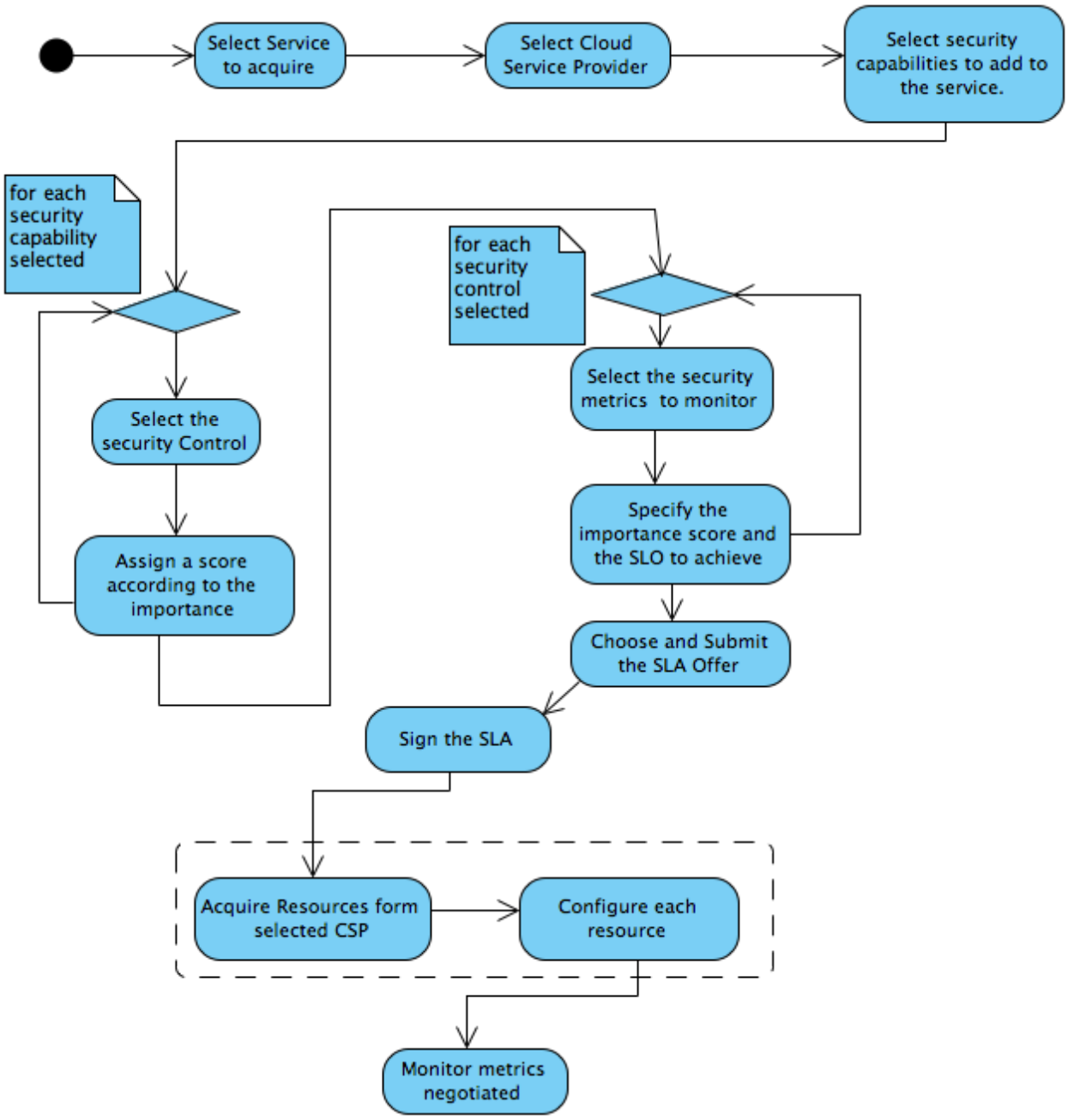

Figure 3. UML Activity Diagram of the Web Container

2. **Security Mechanisms Preparation**, the web container application uses the following Security Mechanisms: WebPool, TLS, SVA, E2EE, DBB, DOS;
3. **SLA Template Preparation,** the SLA Template can be found at https://bitbucket.org/specs-team/specs-app-webcontainer/src/180e033efedb3b20302729a46f7a13713d84ffdb/src/main/resources/sla_template/wsag_SecureWebContainer_ClientEncryption_Replication_v3clean.xml;
4. **SLA Template and Security Mechanisms Deployment**, to deploy the application three Virtual Machines are needed; the chef scripts that automate the installation of the Security Mechanisms as well of all the needed artefacts and executables:
    - Apache Server https://bitbucket.org/specs-team/specs-mechanism-enforcement-

webpool/src/f45e81aa50ebe733253d69d257795578ca956e9a/recipes/apache.rb?at=master
  - HA Proxy https://bitbucket.org/specs-team/specs-mechanism-enforcement-webpool/src/f45e81aa50ebe733253d69d257795578ca956e9a/recipes/haproxy.rb?at=master
  - Nginx Server https://bitbucket.org/specs-team/specs-mechanism-enforcement-webpool/src/f45e81aa50ebe733253d69d257795578ca956e9a/recipes/nginx.rb?at=master

A full detailed architectural description of the application is out of the scope of this deliverable. Technical documentation is available in the SPECS repository.

5.3. Repository

The repository containing all the data and artefacts necessary to understand, use and improve the web container application is:

https://bitbucket.org/specs-team/specs-app-webcontainer

This repository contains the *specs-app-webcontainer-demo* which is the application.

5.4. Installation

The installation guide covers two scenarios:

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

5.4.1. Install using precompiled binaries

The precompiled binaries are available under the SPECS Artefact Repository [13].

Prerequisites:

- Oracle Java JDK 7;
- Java Servlet/Web Container (recommended: Apache Tomcat 7.0.x [10]);
- a running SPECS SLA Platform with Monitoring, Negotiation and Enforcement modules.

Installation steps:

- download the web application archive (war) file from the artefact repository: http://ftp.specs-project.eu/public/artifacts/applications/secure-web-container/webcontainer-app-0.0.1-SNAPSHOT.war
- 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.

5.4.2. Compile and install from source

A developer should have the following prerequisites:

- a Git client [12];
- Apache Maven 3.3.x [11];
- a Java 7 Development Kit;
- Java web container (recommended Apache Tomcat 7.0.x [10]);

- a running SPECS SLA Platform with Monitoring, Negotiation and Enforcement modules.

Installation steps:

- clone the Bitbucket repository:
  - *git clone git@bitbucket.org:specs-team/specs-app-webcontainer.git*
- under *specs-app-webcontainer*
  - run: *mvn install*
  - run*: mvn package*

Maven generates a .war file and automatically stores it in the "/target" folder. By following the steps at Section 5.4.1 the application could be deployed and used.

5.5. Usage

Once the End-User runs the Web Container application, he/she is provided a simple user interface through which he/she can choose first the capabilities he/she wants to add to the service; for each of them the End-User has to select the security controls and finally he/she has to define the Service Level Objectives he/she wants to be guaranteed. Once the End-User has selected all those information, one or more Service Level Agreement offers are shown to the EU, and he/she has to select and sign one of them. The process just described is the Negotiation phase.

Then he/she has to implement the signed SLA so that all the components that are necessary to guarantee the signed SLA are deployed on many virtual machines that are acquired from an external service provider. During the implementation process, whose time varies depending on how many resources have to be acquired and on how many components have to be installed, the user is informed about the current status of the implementation phase; once it has finished, the user has the opportunity to get updated information about each metric defined into the SLA: in particular he/she can access to a web page where there is a summary of all those metrics, and for each of them there is the value signed into the SLA and the values measured using SPECS Monitoring System.

Two screenshots of the application are here reported.

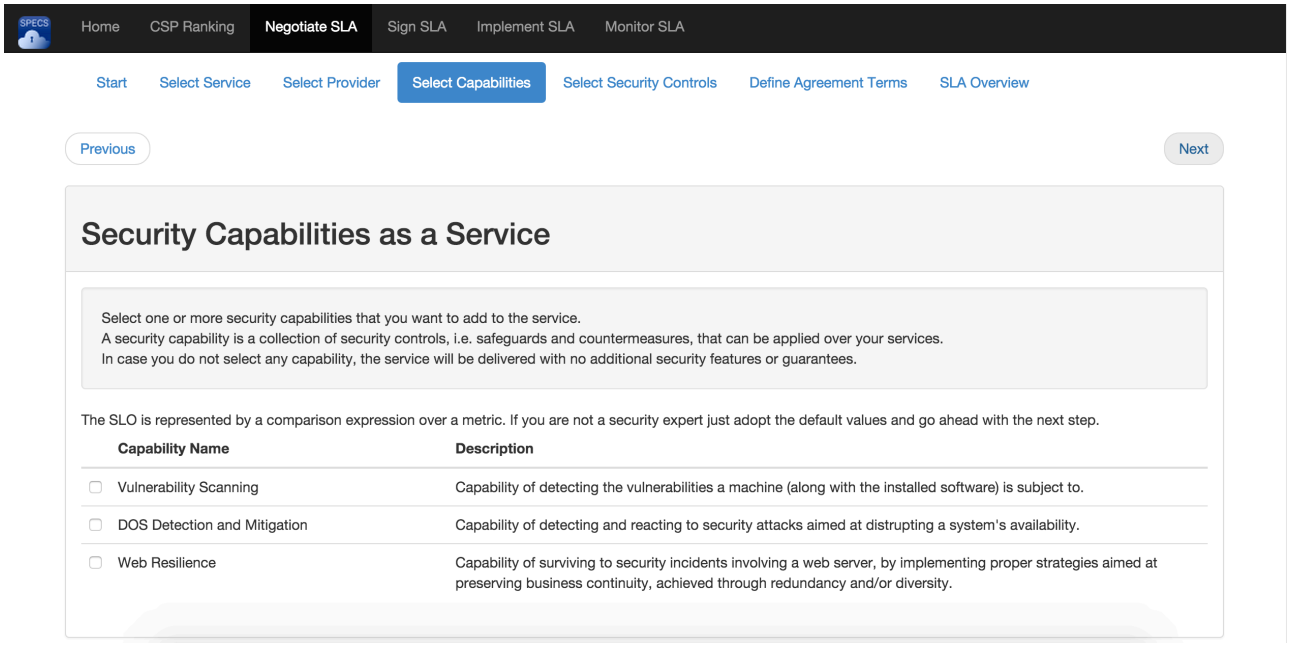

Figure 4. Selection of Capabilities

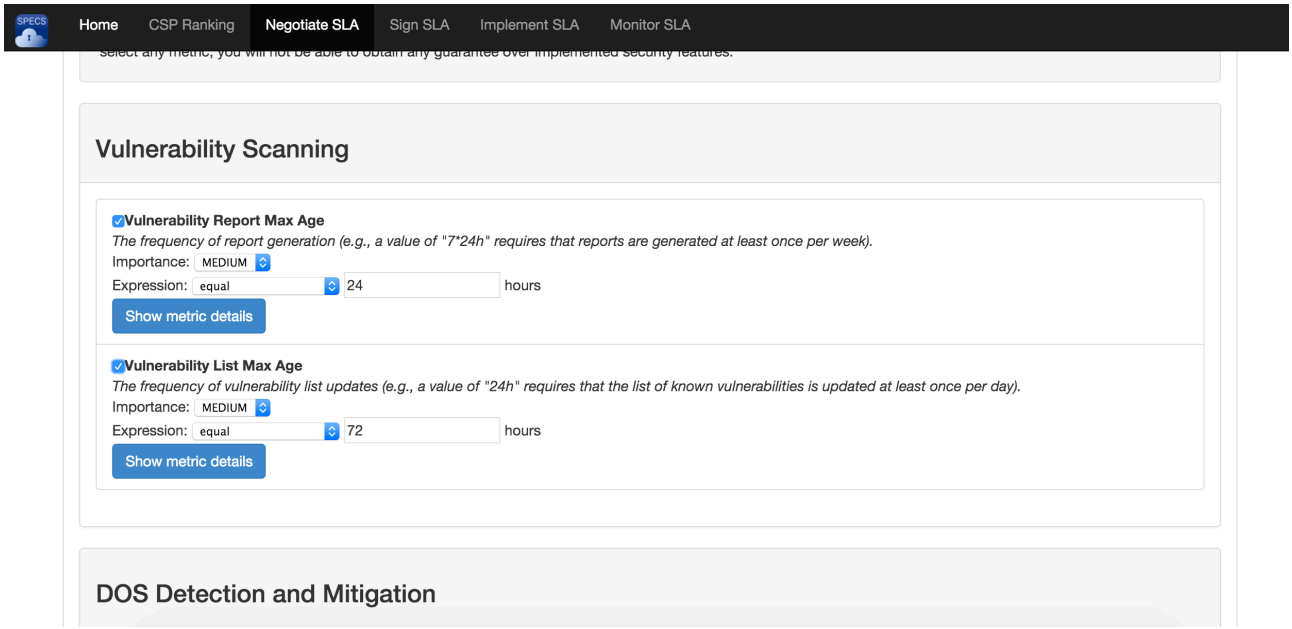

Figure 5. Definition of metrics values

5.6. Test

A testing activity has been conducted having no failed tests. Unit tests allow to cover the 66.9% of the branches of the entire *webcontainer-app* software:

- *eu.specsproject.app.sws.backend* reached the 68.1%
- *eu.specsproject.app.sws.frontend.rest* reached 69.9%
  - *eu.specsproject.app.sws.frontend.rest.entities* reached 100%
  - *eu.specsproject.app.sws.frontend.struts* reached 56.2%
- *eu.specsproject.app.sws*.utility reached 67.7%

Qualitative and quantitative details of these tests are in Appendix A.

6. The Metric Catalogue

The Catalogue is proposed as an application to help the users to manage easily a catalogue of Security Metrics defined according to a specification. The creation or the display of a Security Metric through an XML that is compliant with a specific XSD is not simple (obviously also the update is not a simple operation) for the mean computer user not expert in programming. The Metric Catalogue consists of one or more SQLite [9] databases containing the Security Metrics managed by the users.

6.1. Development Final Results

The development of the application is completed, and a preliminary testing campaign has been successfully accomplished. It is now possible to use the "alpha" release of the application, deploying it from the repository website.

According to D5.1.2, the metric catalogue application covers the Validation Scenarios reported in Table 1 with the indication of the percentage of coverage.

	% of coverage
Metric_Definition	100%

Table 4. Mapping Metric Catalogue on VSs (excerpt from D5.1.2)

The application does not solicit any SPECS components with the exception of the SLA Platform.

Differently from the other SPECS Validation Applications, the Metric Catalogue is not a refinement of any User Story neither has a relation to the solution portfolio. This notwithstanding, the Metric Catalogue is an application that has a strong impact on all the other SPECS VAs which use Metric Catalogue to define and implement new security metrics. Metric Catalogue can be seen as a sort of utility application for SPECS.

Here, a brief discussion of the impact of this Validation Application is reported. The objective is to evaluate how much of the validation-related KPIs (defined in D5.1.2) is covered by the Metric Catalogue. To accomplish to this task, Table 5 reports on the row the KPIs that are related to the execution of VAs. The table has two columns: the first reports which should be the value of the KPIs if Metric Catalogue was the only available application; the second reports these values considering all the VAs.

	Only Metric Catalogue	**All VAs**
EC_U	16.67%	66.67%
EC_{TS}	0%	100%
EC_{IC}	0%	100%
EC_{SLA}	0%	78.9%
EC_{SS}	23%[1]	100%

Table 5. Impact of Metric Catalogue on the Validation KPIs

[1] This percentage takes into account SPECS platform requirements over all requirements

6.2. Application Description

Through a web interface each user can manage a database that represents the catalogue. The user is guided in the creation of a new Security Metric through a form that explains each field to add to create the Security Metric; the application will transform the input information in the form into an XML file.

The display of a Security Metric is not simple through an XML, so the Security Metric Catalogue allows displaying a Security Metric in a structured way to get all the information about it easily. Moreover each Security Metric can be updated using the same structured form to create it. Finally the user can retrieve a backup of the metrics or can restore it very easily. Once the EU runs the Security Metric Catalogue application, he/she is provided a simple user interface through which he/she can choose the section of interest.

To develop this application, the phases of the process described in section 4 have been followed with a strong simplification due to the nature of the application. As stated before, Metric Catalogue is a utility application whose aim is not to offer SLA based – security enforced services. It simply allows CRUD (Create, Read, Update ad Delete) operations on security metrics. Hence only the Cloud Service Definition phase has been accomplished:

1. **Cloud Service Definition**, the functionalities of the Metric Catalogue are depicted in the UML Use Case Diagram in Figure 6. Each functionality is described by a separate UML Activity Diagram: Store Metric, allowing to store a new Security Metric from its XML representation (Figure 7); Store Metric Form: allowing to create a new Security Metric compiling a form (Figure 8); Retrieve Metric (Figure 9); Get Metric XML (Figure 10); Remove Metric (Figure 11); Update Metric (Figure 12); Backup Metrics (Figure 13) and Restore Metrics (Figure 14).

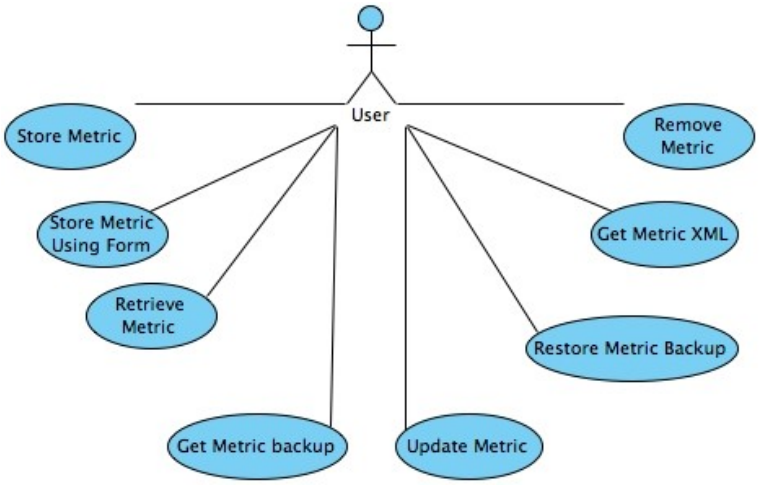

Figure 6. Overall UML Use Case Diagram of the Metric Catalogue

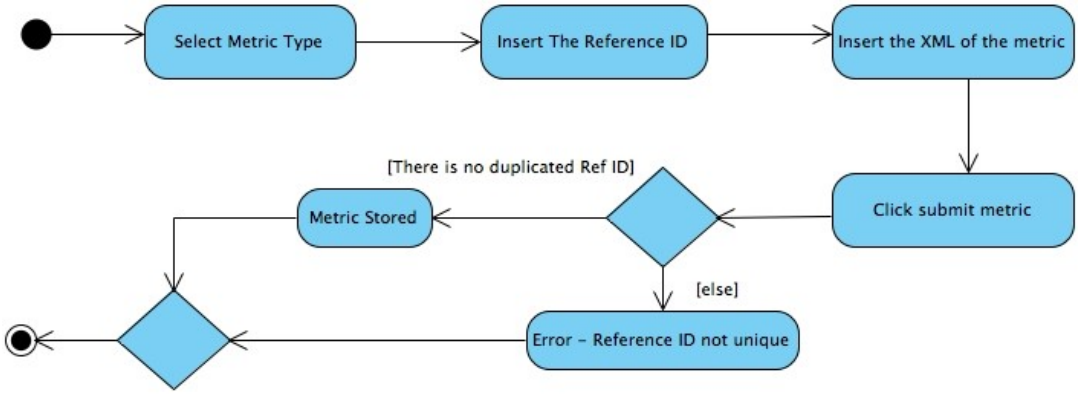

Figure 7. Store Metric UML Activity Diagram

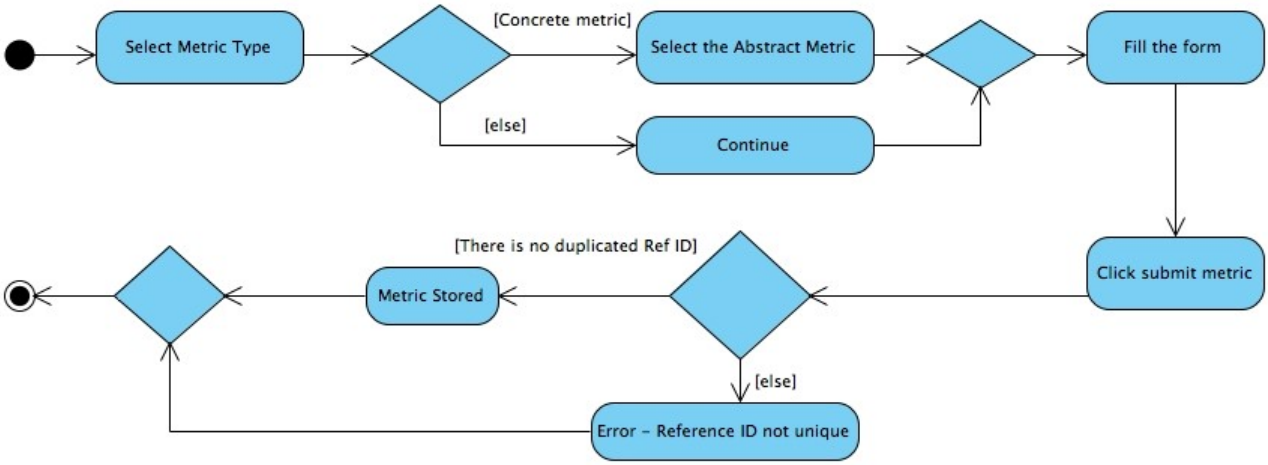

Figure 8. Store Metric Using Form UML Activity Diagram

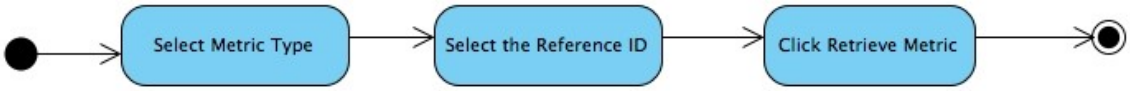

Figure 9. Retrieve Metric UML Activity Diagram

Figure 10. Get Metric XML UML Activity Diagram

Figure 11. Remove Metric UML Activity Diagram

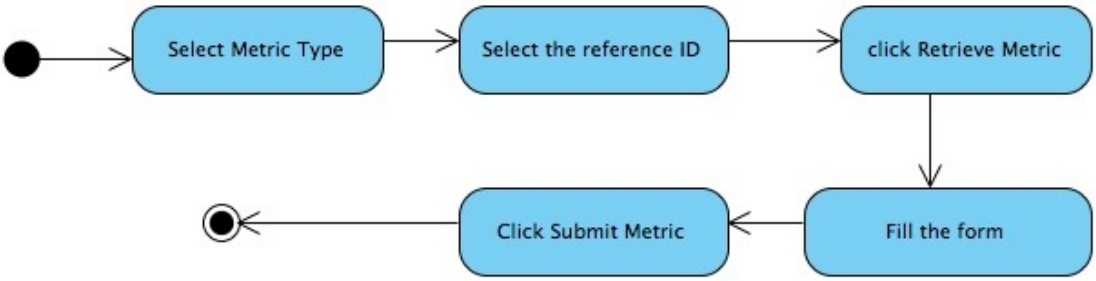

Figure 12. Update Metric UML Activity Diagram

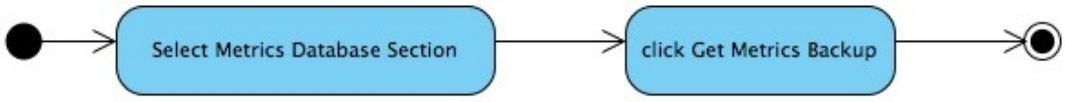

Figure 13. Backup Metrics UML Activity Diagram

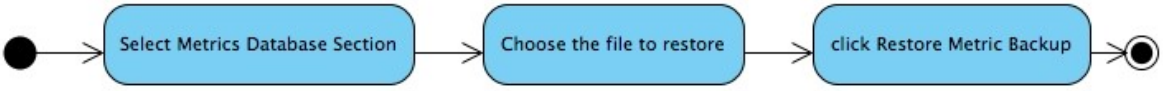

Figure 14. Restore Metrics UML Activity Diagram

A full detailed architectural description of the application is out of the scope of this deliverable. Technical documentation is available in the SPECS repository.

6.3. Repository

The repository containing all the data and artefacts necessary to understand, use and improve the web container application is:

https://bitbucket.org/specs-team/specs-app-security_metric_catalogue

6.4. Installation

The installation guide covers two scenarios:

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

6.4.1. Install using precompiled binaries

The precompiled binaries are available under the SPECS Artefact Repository [13].

Prerequisites:

- Oracle Java JDK 7;
- SQLite 3.9.x [9];
- Java Servlet/Web Container (recommended: Apache Tomcat 7.0.x [10]);
- a running SPECS SLA Platform.

Installation steps:

- download the web application archive (war) file from the artefact repository: http://ftp.specs-project.eu/public/artifacts/applications/metric-catalogue-app/metric_catalogue-app-0.0.1-SNAPSHOT.war
- 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.

6.4.2. Compile and install from source

A developer should have the following prerequisites:

- a Git client [12];
- Apache Maven 3.3.x [11];
- SQLite 3.9.x [9];
- a Java 7 Development Kit;
- Java web container (recommended Apache Tomcat 7.0.x [10]);
- a running SPECS SLA Platform.

Installation steps:

- clone the Bitbucket repository:
  - *git clone git@bitbucket.org:specs-team/ specs-app-security_metric_catalogue.git*
- under *specs-app-security_metric_catalogue*
  - run: *mvn install*
  - run*: mvn package*

Maven generates a .war file and automatically stores it in the "/target" folder. By following the steps at Section 6.4.1 the application could be deployed and used.

6.5. Usage

Once the EU runs the Metric Catalogue application, he/she is provided a simple user interface through which he/she can choose the section of interest. There is a section for each of the functionalities previously described. Given the simplicity of the structure of the application, some screenshots are reported in the following figures.

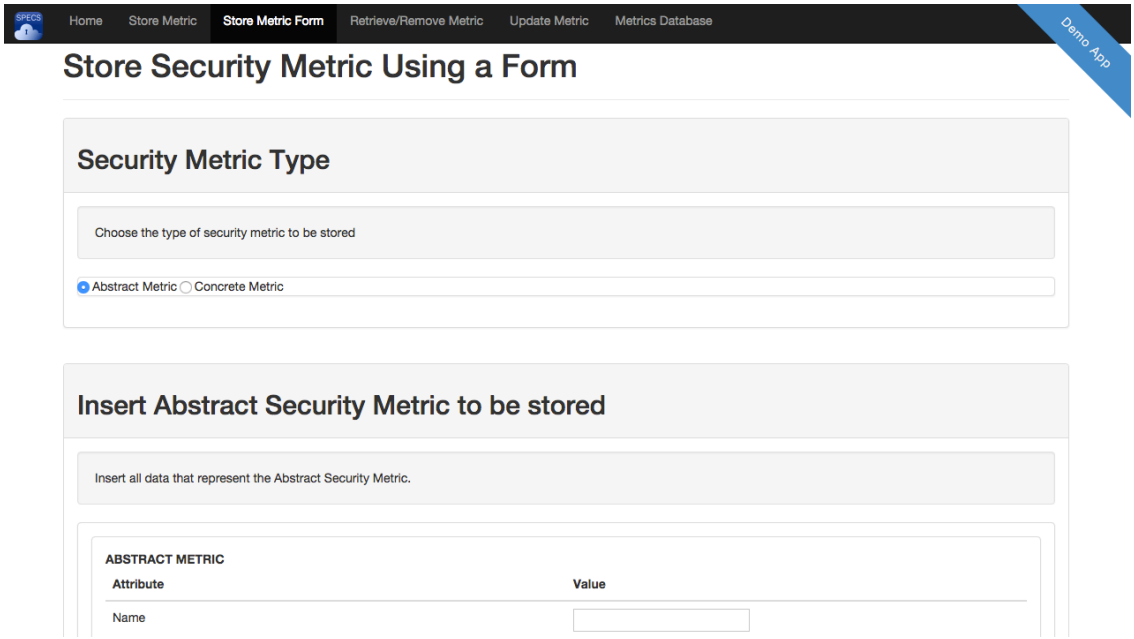

Figure 15. Store a Metric Using a Form

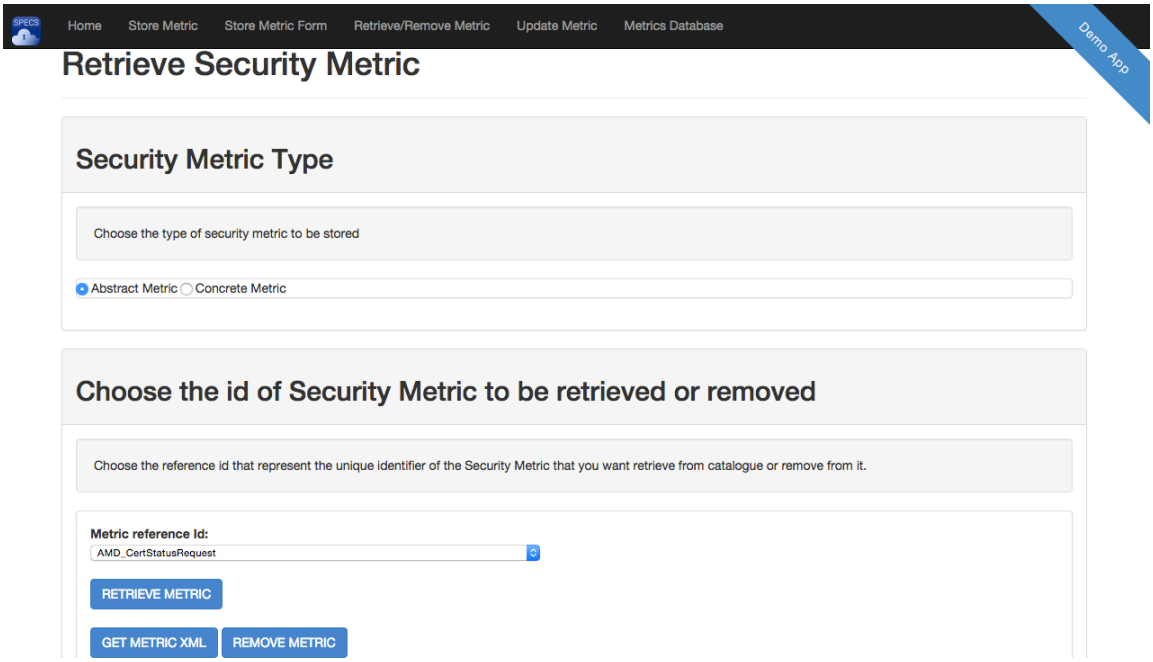

Figure 16. Retrieve a Security Metric

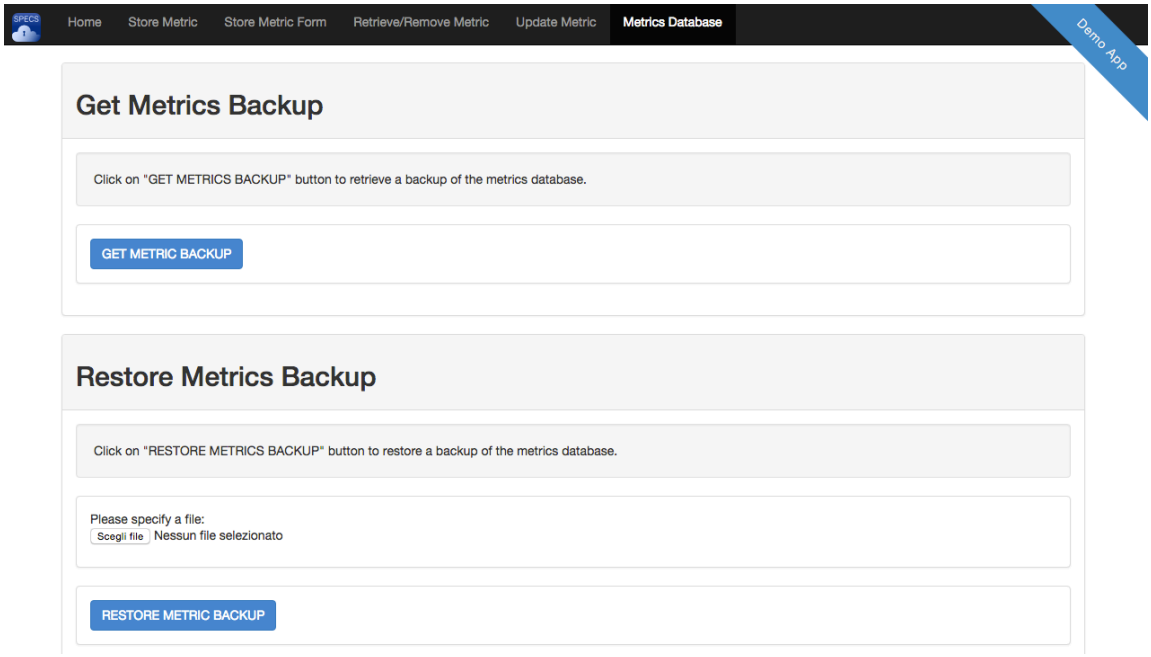

Figure 17. Manage the Security Metrics backup

6.6. Test

A testing activity has been conducted having no failed tests. Unit tests allow to cover the 71.8% of the branches of the entire *specs-app-security_metric_catalogue* software:

- *eu.specsproject.app.metriccatalogue.backend* reached the 83.8%
- *eu.specsproject.app.metriccatalogue.frontend.rest* reached 63.3%
- *eu.specsproject.app.metriccatalogue.utility* reached 46.4%
- *eu.specsproject.app.metriccatalogue.rest.entities* reached 100%

Qualitative and quantitative details of these tests are in Appendix B.

7. Conclusions

This deliverable provided installation and usage procedures of two Validation Applications of SPECS: Secure Web Container and Security Metric Catalogue.

The testing activity has not addressed any meaningful issue and, hence, the applications can be downloaded and tested also by external users as the source code is available in the reported repositories.

The coverage levels reached in the unit testing activities are 66.9% for Web Container and 71.8% for the Metric Catalogue.

The importance of the Web Container application is evidenced by the crucial role it plays in the coverage of the Key Concerns, as well as the relationship between this application and the SPECS portfolio scenario. The later feature here highlights the possibility to exploit this application for building real world industrial products.

On the other hand, the Metric Catalogue application has a prime role among all the other SPECS applications since many SPECS application usage scenarios need the services provided by Metric Catalogue.

8. Bibliography

[1] "Unified Modeling Language version 2.5". Object Management Group. 2015.
[2] http://www.specs-project.eu/solutions-portofolio/secure-web-container/
[3] http://docs.chef.io/
[4] G. J. Myers, C. Sandler. "The Art of Software Testing". John Wiley & Sons. 2004
[5] K. Naik, P. Tripathy. "Software Testing and Quality Assurance: Theory and Practice." John Wiley & Sons. 2008
[6] http://junit.org
[7] http://wiremock.org
[8] http://www.sonarqube.org
[9] http://www.sqlite.org
[10] http://tomcat.apache.org
[11] http://maven.apache.org
[12] http://git-scm.com
[13] SPECS Artifact Repository, [Online], http://ftp.specs-project.eu/public/artifacts/
[14] http://sonar.services.ieat.ro

Appendix A – Web Container Unit Tests

This appendix details the artefacts produced to test the Web Container application as well as reports the testing results in term of branch coverage.

Fourteen tests have been defined and reported in Table 6, Table 7, Table 8, Table 9, Table 10, Table 11, Table 12, Table 13, Table 14, Table 15, Table 16, Table 17, Table 18 and Table 19.

Test ID	getSlaTemplateTest
Test objective	The goal is to verify that the API listen on the path "/slaTemplate" returns the template of the SLA returned by the call to the api "/slo-manager-api/sla-negotiation/sla-templates/{template_id}" (GET request)
Inputs	Get request to "/slaTemplate" path
Expected results	A 200 OK response with the value of the sla template in the body
Outputs	200 OK
Comments	All operations executed successfully

Table 6. Details of getSlaTemplateTest

Test ID	signTest
Test objective	The goal is to verify that the API listen on the path "/sign" put the state of the sla template sent in the body in sign using the api "/sla-manager/cloud-sla/slas/ {sla_id}/sign" (POST request)
Inputs	Post request to "/sign" path
Expected results	A 204 No Content response that represents the correct execution of the request
Outputs	204 No Content
Comments	All operations executed successfully

Table 7. Details of signTest

Test ID	submitTest
Test objective	The goal is to verify that the API listen on the path "/submit" put the sla template sent in the body using the api "/sla-manager/cloud-sla/slas/ {sla_id}" (POST request)
Inputs	Post request to "/submit" path
Expected results	A 204 No Content response that represents the correct execution of the request
Outputs	204 No Content
Comments	All operations executed successfully

Table 8. Details of submitTest

Test ID	implementTest
Test objective	The goal is to verify that the API listen on the path "/implement" put the state of the sla identified by the id sent in the body in observe using the api "/sla-manager/cloud-sla/slas/{sla_id}/observe" and so it verifies that the sla is returned correctly to implement it (POST request)

Inputs	Post request to "/implement" path
Expected results	A 204 No Content response that represents the correct execution of the request
Outputs	204 No Content
Comments	All operations executed successfully

Table 9. Details of implementTest

Test ID	implementedInfoTest
Test objective	The goal is to verify that the API listen on the path "/implementedInfo" works properly
Inputs	Post request to "/implementedInfo" path
Expected results	A 200 OK response with the implemented info in the body
Outputs	200 OK
Comments	All operations executed successfully

Table 10. Details of implementedInfoTest

Test ID	implementedMetricValuesTest
Test objective	The goal is to verify that the API listen on the path "/metricValues" works properly
Inputs	Post request to "/metricValues" path
Expected results	A 200 OK response with the implemented metrics info in the body
Outputs	200 OK
Comments	All operations executed successfully

Table 11. Details of implementedMetricValuesTest

Test ID	getSlaOffersTest
Test objective	The goal is to verify that the API listen on the path "/slaoffers" returns the list of the SLA offers related to the SLA sent in the body. The sla offers are returned by the call to the api "/slo-manager-api/sla-negotiation/sla-templates/{template_id}/slaoffers" (POST request)
Inputs	Post request to "/slaoffers" path
Expected results	A 200 OK response with the list of the sla offers in the body
Outputs	200 OK
Comments	All operations executed successfully

Table 12. Details of getSlaOffersTest

Test ID	getSlaOfferTest
Test objective	The goal is to verify that the API listen on the path "/slaOffer" returns the value of the SLA offer identified by the id sent in the body of the request. To retrieve the sla offer is used the call to the api "/slo-manager-api/sla-negotiation/sla-templates/{template_id}/slaoffers/{offer_id}" (POST request)
Inputs	Post request to "/slaOffer" path

Expected results	A 200 OK response with the value of the sla offer in the body
Outputs	200 OK
Comments	All operations executed successfully

Table 13. Details of getSlaOfferTest

Test ID	implementationInfoTest
Test objective	The goal is to verify that all methods of the ImplementationInfo 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
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 ImplementationInfo class that has all the attributes set with the required values
Comments	All operations executed successfully

Table 14. Details of implementedInfoTest

Test ID	metricInfoTest
Test objective	The goal is to verify that all methods of the MetricInfo 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
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 MetricInfo class that has all the attributes set with the required values
Comments	All operations executed successfully

Table 15. Details of implementedInfoTest

Test ID	sqliteHelperTest
Test objective	The goal is to verify that all methods of the SQLiteHelper 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
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 SQLiteHelper class that has all the attributes set with the required values
Comments	All operations executed successfully

Table 16. Details of sqliteHelperTest

Test ID	implementActionTest

Test objective	The goal is to verify that the ImplementAction Struts works fine and that it returns a not null ActionForward Object.
Inputs	Call to execute method
Expected results	An ActionForward Object
Outputs	Not Null ActionFarward Object
Comments	All operations executed successfully

Table 17. Details of implementedActionTest

Test ID	observeActionTest
Test objective	The goal is to verify that the ObserveAction Struts works fine and that it returns a not null ActionForward Object.
Inputs	Call to execute method
Expected results	An ActionForward Object
Outputs	Not Null ActionFarward Object
Comments	All operations executed successfully

Table 18. Details of observeActionTest

Test ID	signActionTest
Test objective	The goal is to verify that the SignAction Struts works fine and that it returns a not null ActionForward Object.
Inputs	Call to execute method
Expected results	An ActionForward Object
Outputs	Not Null ActionFarward Object
Comments	All operations executed successfully

Table 19. Details of signActionTest

Once the tests have been executed, SonarQube elaborates the results and produce a graphical representation of the code coverage reported in Figure 18: the colors go from red (0% coverage) to bright green (100% coverage).

Figure 18. Coverage of the Web Container application (by SonarQube)

Appendix B – Metric Catalogue Unit Tests

This appendix details the artefacts produced to test the Web Container application as well as reports the testing results in term of branch coverage.

Seven tests have been executed: they are synthetically described in Table 20, Table 21, Table 22, Table 23, Table 24, Table 25, Table 26 and Table 27.

Test ID	SQLiteHelperTest
Test objective	The goal is to verify that the constructor of the class SQLiteHelper returns a correct instance of the class
Inputs	n.a.
Expected results	An instance of the class SQLiteHelper that allows the connection to database
Outputs	The instance of the SQLiteHelper class
Comments	All operations executed successfully

Table 20. Details of SQLiteHelperTest

Test ID	retrieveMetricsTest
Test objective	The goal is to verify that the API listen on the path "/retrieve" returns the list of the security metrics returned by the call to the api "/metric-catalogue/cloud-sla/security-metrics" (GET request)
Inputs	Get request to "/retrieve" path
Expected results	A 200 OK response with the list of security metrics in the body
Outputs	200 OK
Comments	All operations executed successfully

Table 21. Details of retrieveMetricTest

Test ID	storeMetricTest
Test objective	The goal is to verify that the API listen on the path "/store" stores the value of the security metric sent in the body using the api "/metric-catalogue/cloud-sla/security-metrics" (POST request)
Inputs	Post request to "/store" path
Expected results	A 200 OK response with the id of the stored security metric in the body
Outputs	200 OK
Comments	All operations executed successfully

Table 22. Details of storeMetricTest

Test ID	updateMetricTest
Test objective	The goal is to verify that the API listen on the path "/update" updates the value of the security metric identified by the id param with the value of the metric sent in the body using the api "/metric-catalogue/cloud-sla/security-metrics/{id}" (PUT request)
Inputs	Put request to "/update" path

Expected results	A 200 OK response with the id of the updated security metric in the body
Outputs	200 OK
Comments	All operations executed successfully

Table 23. Details of updateMetricTest

Test ID	retrieveMetricByIdTest
Test objective	The goal is to verify that the API listen on the path "/retrieve/{id}" returns the security metric identified by the id param calling the api "/metric-catalogue/cloud-sla/security-metrics/{id}" (GET request)
Inputs	Get request to "/retrieve/{id}" path
Expected results	A 200 OK response with the value of the security metric identified by the id param in the body
Outputs	200 OK
Comments	All operations executed successfully

Table 24. Details of retrieveMetricTest

Test ID	removeMetricTest
Test objective	The goal is to verify that the API listen on the path "/remove/{id}" removes the security metric identified by the id param calling the api "/metric-catalogue/cloud-sla/security-metrics/{id}" (DELETE request)
Inputs	Delete request to "/remove/{id}" path
Expected results	A 200 OK response with the value of the id of the security metric removed in the body
Outputs	200 OK
Comments	All operations executed successfully

Table 25. Details of removeMetricTest

Test ID	getMetricBackupTest
Test objective	The goal is to verify that the API listen on the path "/backupMetrics/{dbName}" returns the file that represents the backup of the database of the security metrics calling the api "/metric-catalogue/cloud-sla/security-metrics/backup/{dbname}" (GET request)
Inputs	Get request to "/backupMetrics/{dbName}" path
Expected results	A 200 OK response with the backup file in the body
Outputs	200 OK
Comments	All operations executed successfully

Table 26. Details of getMetricBackupTest

Test ID	getMetricXMLTest
Test objective	The goal is to verify that the API listen on the path "/retrieve/{metric_id}.xml" returns the xml file that contains the

	value of the security metric identified by the metric_id param calling the api "/metric-catalogue/cloud-sla/security-metrics/{metric_id}.xml" (GET request)
Inputs	Get request to "/retrieve/{metric_id}.xml" path
Expected results	A 200 OK response with the xml file in the body
Outputs	200 OK
Comments	All operations executed successfully

Table 27. Details of getMetricXMLTest

Once the tests have been executed, SonarQube elaborates the results and produce a graphical representation of the code coverage reported in Figure 19: the colors go from red (0% coverage) to bright green (100% coverage).

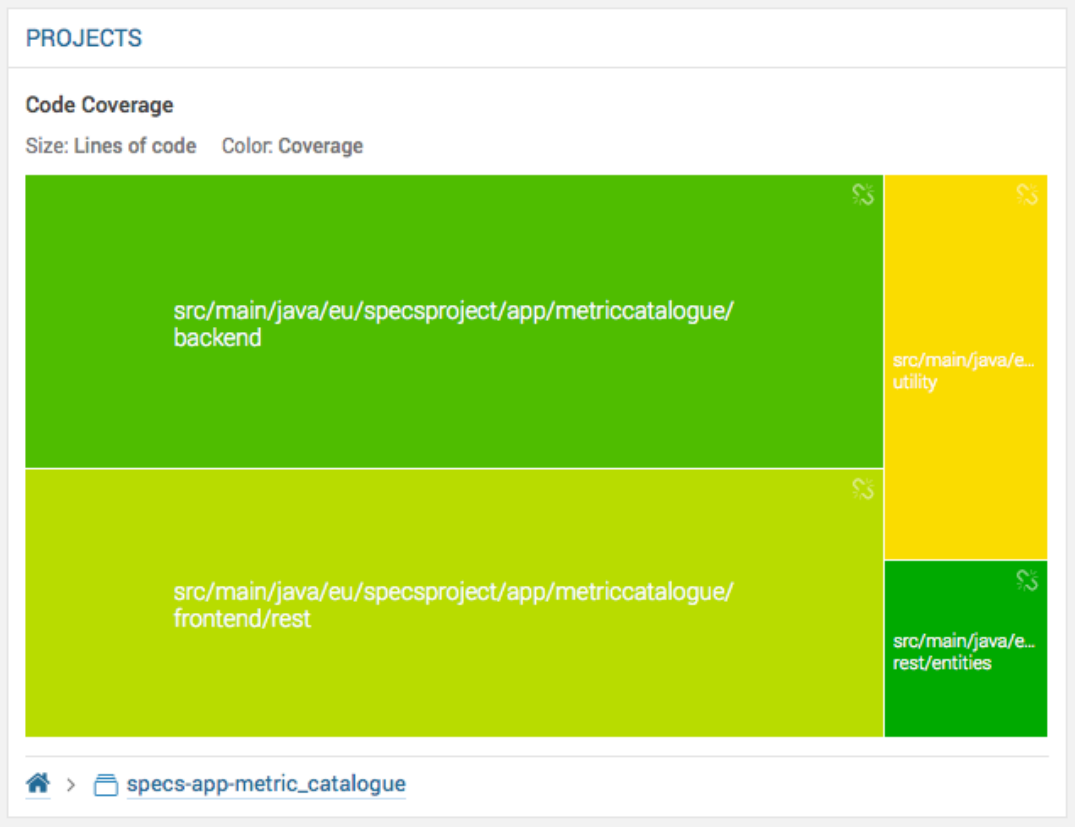

Figure 19. Coverage of the Metric Catalogue application (by SonarQube)