



D 6.6 SW QUALITY ASSESSMENT REPORT

Project title	Collaborative Recommendations and Adaptive Control for Personalised Energy Saving
Project acronym	enCOMPASS
Project call	EE-07-2016-2017 Behavioural change toward energy efficiency through ICT
Work Package	WP6
Lead Partner	PDX
Contributing Partner(s)	PMI, CERTH, GRA, SMOB, SUPSI
Security classification	Public (PU)
Contractual delivery date	31/10/2019
Actual delivery date	29/10/2019
Version	1.0
Reviewers	SMOB, PMI

History of changes

Version	Date	Comments	Main Authors
0.1	10/09/2019	First draft: DDP (Deliverable Development Plan) – definition of the document structure and the contributions expected from each partner	Laura Scalzo (PDX)
0.2	11/10/2019	PMI contribution: Code quality and performance testing of the AA and Funergy	Sergio Herrera (PMI)
0.3	15/10/2019	CERTH contribution: code quality testing	Stelios Krinidis (CERTH) and Valia Dimaridou (CERTH)
0.4	16/10/2019	PDX adds its contribution to Testing section and Appendix	Laura Scalzo (PDX)
0.5	16/10/2019	RE contribution: code quality and testing information	Gabor Vadasz, Janos Matyas, Krisztina Tarjanyi (GRA)
0.6	16/10/2019	DE contribution: code quality and testing information	Marco Derboni (SUPSI)
0.7	17/10/2019	Changed Figure 1	Laura Scalzo (PDX)
0.8	18/09/2019	SMOB contribution: code quality testing	Luigi Caldararu (SMOB)
0.90	15/10/2019	PMI contribution	S Herrera (PMI)
0.91	21/10/2019	SMOB contribution merging	L Scalzo (PDX)
0.92	25/10/2019	Usability test reference addition	K Koroleva (EIPCM)
0.93	28/10/2019	Test added and final Integration	L. Scalzo (PDX)
1.0	28/10/2019	PMI: final quality check	Piero Fraternali, Andrea Vallan (PMI/FP)

Disclaimer

This document contains confidential information in the form of the enCOMPASS project findings, work and products and its use is strictly regulated by the enCOMPASS Consortium Agreement and by Contract no. 723059.

Neither the enCOMPASS Consortium nor any of its officers, employees or agents shall be responsible or liable in negligence or otherwise howsoever in respect of any inaccuracy or omission herein.

The contents of this document are the sole responsibility of the enCOMPASS consortium and can in no way be taken to reflect the views of the European Union.



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 723059.

TABLE OF CONTENTS

Executive Summary	6
1 Introduction	7
2 Quality and Testing Domain	9
2.1 Code quality Testing	9
2.2 Individual component Testing	11
2.3 Integrated Testing	12
2.4 Functional Testing	13
2.5 Performance and Scalability Testing	13
2.6 Deployment Testing	13
3 Quality Assessment Metrics	14
3.1 Code size metrics	14
3.2 Code Quality metrics	14
3.3 Performance metrics	15
3.4 Reliability metrics	17
4 Deployment and Quality Assurance Tools	19
4.1 Deployment Environment (IDE)	19
4.2 Code Quality Checking Tools	21
4.3 Version Control Tools	24
4.4 Issue Tracker	24
4.5 Performance Testing Tools	25
5 Performed Tests	25
5.1 The Service Integration and Orchestration Component	25
5.2 The Awareness Application for Web and Mobile Access	26
5.3 The Gamification Engine	27
5.4 The Energy Efficiency Console for Utility and Buildings	28
5.5 The Disaggregation Engine	28
5.6 The Notification Engine	29
5.7 The Recommendation Engine	29
5.8 The Inference Engine	30
5.9 Funergy – Digital Game Extension of the Board Game	30
6 Appendix A	31
6.1 Service Integration and Orchestration	31
6.1.1 Service Integration and Orchestration Performance Tests	31
6.1.2 Service Integration and Orchestration Functional Tests	32

6.2	Gamification engine and Awareness Application (AA)	33
6.2.1	Gamification Engine Performance Tests	33
6.2.2	Awareness Application Functional Tests	35
6.3	Disaggregation Engine	39
6.3.1	Disaggregation Engine Performance Tests	39
6.3.2	Disaggregation Engine Functional Tests	39
6.4	Notification Engine	41
6.4.1	Notification Engine Performance Tests	41
6.4.2	Notification Engine Functional Tests	41
6.5	Recommendation Engine	42
6.5.1	Recommendation Engine Functional Tests	43
6.6	Inference Engine	47
6.6.1	Inference Engine Performance Tests	47
6.7	Funergy	49
6.7.1	Funergy Performance Tests	49
6.7.2	Funergy Functional Tests	51
7	References	53

EXECUTIVE SUMMARY

This deliverable summarizes the quality aspects of the developed software. As per its description in the DOW D6.6 SW (Quality assessment report):

SW Quality assessment report: This deliverable is a summary of the findings of the code verification and application testing procedures which have been used during the development of the platform. It provides metrics to assess the overall quality, usability¹ and reliability of the platform.

The enCOMPASS platform (the infrastructure to collect and organize the energy consumption and sensor data from end-consumers and public buildings, with the first integration of the user interfaces) includes component releases that have been rolled out on the Pilot sites of the case studies SES (Locarno, Switzerland), SHF (Hassfurt, Germany) and WVT (Athens, Greece):

- The Service Integration and Orchestration Component;
- The Awareness Application for web and mobile access (Google and iOS versions);
- The Gamification Engine and the Energy Efficiency Console for Buildings;
- The Disaggregation Engine component;
- The Inference Engine component;
- The Recommendation Engine component;
- The Energy Efficiency game (FUNERGY).

The dependencies of this final deliverable on preceding ones are as follows:

- The architecture and components of the EnCompass platform are described in **D6.2 (Platform Architecture and Design)**, especially in the section 2 where the EnCompass architecture specification is detailed. This deliverable reports the quality assurance procedures applied to the components designed in D6.2. The reader is referred to D6.2 for the terminology and the explanation of the modules subjected to testing and quality assurance.
- **D6.5 (Platform – Final prototype)** is the software deliverable containing the final versions of all the components of the EnCompass platform. The software package is accompanied by a reporting document, which describes the updates and additions from release R1 to the final release of the EnCompass platform (R3, due at the end of the project).

¹ The results of initial usability tests in the development cycle have been reported in **D5.3 First visualization and feedback interfaces and behavioural game concept**, while the final results of the usability evaluation in the pilots are reported in **D7.4 Final overall validation and impact report**.

1 INTRODUCTION

From a Software System perspective, the EnCompass Platform should be a scalable integrated Platform, made of heterogenous components being able to process large sets of raw data and being able to serve a large base of End Users of Energy Utility Companies. Also, the development of EnCompass lasted for a considerable period with different members of the Project Consortium being involved in the process. Adapted to these specifics, the **Software Quality Assurance (QA) Strategy** was naturally built around the lifecycle stages of the Platform.

The main phases of Platform development life cycle and their respective QA focus, are:

1. **Individual Software Asset Phase.** According to Project deliverable “D8.1 Early Exploitation Plan”, in this incipient phase some individual software components were identified: Funergy digital game, Awareness Application, Smart Metered Management Data Component (SMMDC). QA during this phase focused on functional individual component testing and integration testing using mock-up endpoints;
2. **EnCompass Platform Prototype.** In this phase, the components were integrated together through a service integration layer. During this phase, the QA focused on End-to-End integrated tests among all Platform components;
3. **SES, SHF and WVT Demo Cases.** The EnCompass Platform has been deployed on three production software environments hosted by SES in Locarno (Switzerland), SHF in Hassfurt (Germany) and WVT in Athens (Greece). In this stage, User Experience (UX) based on direct user feedback were agreed to be central subjects to be monitored. Also, consumption data processing was validated against End Users;

The Figure 1 presents at a high-level stages of development lifecycle. Even if each of the lifecycle phases had a central role at strategical level, at the detailed level all other activities related to QA such as unit testing, functional and non-functional testing, were also performed and monitored.

Due to the live deployments at SES, SHF and WVT Utility Companies, a forking strategy was also applied. The following structure of the main source code repository was envisioned at Platform Level:

- Core repository;
- Prototype repository;
- SES repository;
- SHF repository;
- WVT repository.

During the development of the Demo Cases, each change request or bug notification raised at any of the 3 demo sites was classified and inserted appropriately into the Platform code structure: core, prototype or demo case level.

Production environments of SES, SHF and WVT also raised the challenge of maintaining a test environment for each Demo Case Production Environment and of releasing packages deployment from the Test Environment to Production Environment.

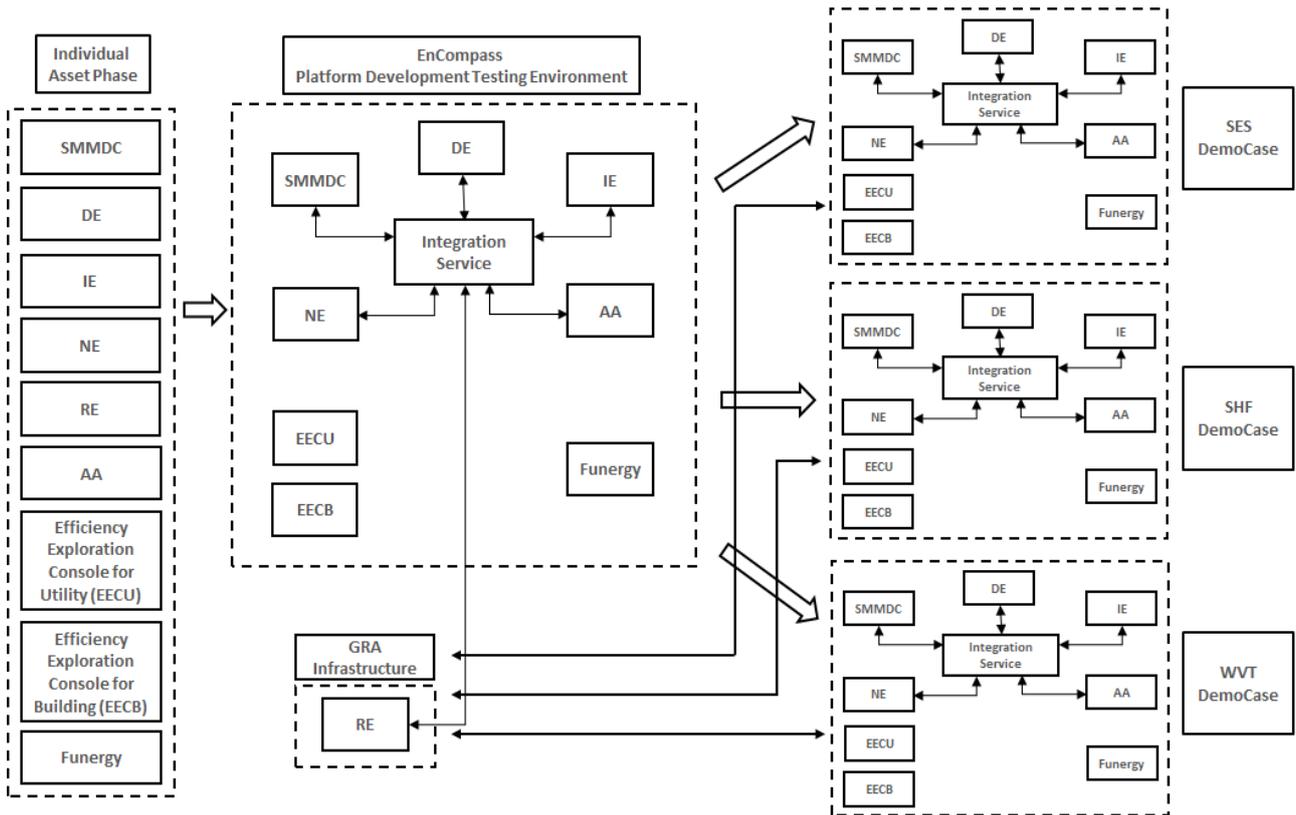


Figure 1 - Phases of the EnCompass development lifecycle.

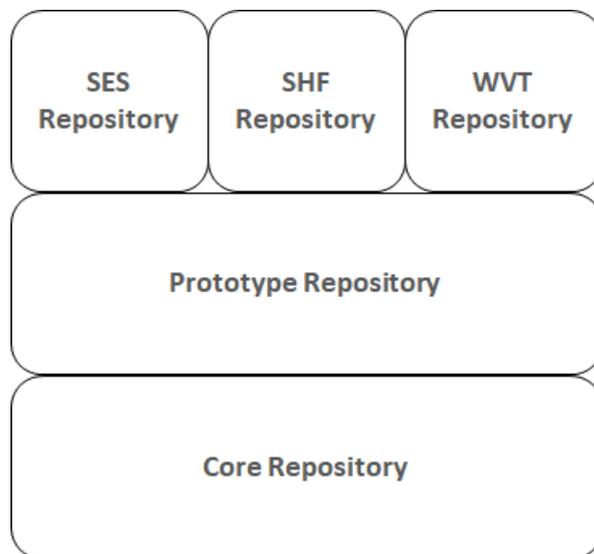


Figure 2- Structure of the EnCompass software repository.

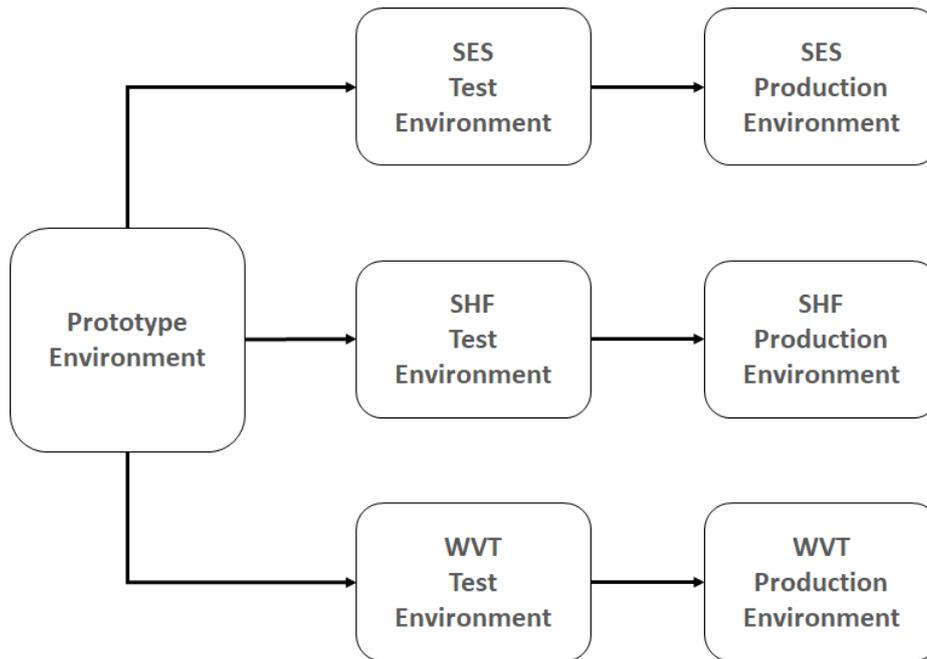


Figure 3– Release deployment flow in the EnCompass use cases.

A change at Core or Prototype level should also be deployed at Demo Case test environment.

A change at Demo Case level should only be tested at Demo Case Test Environment.

Tested Release Packages will be deployed from Demo Case Test Environment to Demo Case Production Environment.

2 QUALITY AND TESTING DOMAIN

2.1 CODE QUALITY TESTING

A good software system must ensure quality starting from of its building blocks: the quality of source code. Quality of source code was pursued by adopting best practice coding procedures and by using tools such as Code Analysers and Code Optimisers (such as JIndent² by Newforms Technologies and YourKit³) and tools for automated code generation (such as the WebRatio⁴ platform).

Coding procedures were exploited to achieve:

- consistent naming conventions;
- consistent code commenting and documentation;
- clear structure of project files;
- clear structure of code layers:
 - presentation layer;
 - business logic layer;

² <http://www.newforms-tech.com/products/jindent/about>

³ <https://www.yourkit.com/>

⁴ <http://www.webratio.com/>

- data access layer.

Code Analysers and Code Optimiser tools helped to:

- avoid usage of deprecated classes or methods;
- avoid usage of repeated code;
- improve memory management.

Smart Meter and Sensor Data Manager (SMSDM) component was developed using Java based frameworks as Spring and Camel. The development process was performed using the Integrated Development Environment (IDE) “Eclipse”, provided by Eclipse Foundation.

The **Awareness application and the Gamification Engine** were developed using WebRatio, a model-driven development tool. WebRatio automatically generates the code based on the model designed using the OMG Flow Modelling language IFML standard⁵ which automatically ensures the same code quality across the entire application.

The **Funergy mobile application** was developed using IFMLEdit⁶, an online model-driven development tool for mobile and web applications. IFMLEdit automatically generates NodeJS code based on the model designed, it is also based on the OMG standard IFML. The model-driven approach ensures the code quality across the entire application and across several platforms, as in this case the generated code is used on iOS and Android devices.

The **Inference Engine** was developed using Python. However, the developed code has been checked utilizing Pylint⁷, a tool for testing coding standard (e.g. variable names and well forming, line-code’s length, imported modules, etc.), error detection (e.g. if modules are imported, if interfaces are implemented. Hardcoded names and paths, etc.). Thus, the final code has been improved leading to qualified code able to use it either at windows and/or ubuntu environment.

The **Recommendation Engine** is a SaaS component, integrated using loose coupling principle. It receives the data from the encompass platform core in tab separated files transmitted through sftp protocol. The data transmission is triggered using the http REST API of the platform. The recommendations are returned to enCOMPASS platform through the REST API in json format.

The RE is implemented in python programming language, and was developed using IntelliJ PyCharm, an integrated development environment for python. The development of the production code was aided by the Python code insight tools of PyCharm.

Jupyter Notebook, a web-based interactive environment for statistical modelling, data visualization, and machine learning, was used for algorithm prototyping.

The **Disaggregation Engine** was developed using the Integrated Development Environments (IDEs) “Eclipse”, provided by Eclipse Foundation⁸, and “PyCharm” (provided by JetBrains⁹).

Eclipse has been used to develop the integration of the DE module into the orchestration component, while PyCharm has been used to develop the Disaggregation Engine backend.

⁵ <http://www.ifml.org/>

⁶ <https://ifmledit.org/>

⁷ <https://www.pylint.org/>

⁸ <https://www.eclipse.org/ide/>

⁹ <https://www.jetbrains.com/pycharm/>

Both IDEs automate many tasks when writing code: optimize import action, clean unused imports, auto-generate code, inspect code that detect and correct anomalous code in the project, find and highlight various problems, locate dead code, find probable bugs, find spelling problems and improve the overall code structure. All these features improve the quality of the code to an adequate level.

Given that the Disaggregation Engine processes very large amount of data at each call, it is independently called on an external server while its outcome is joined to the outcome of the other components. Therefore, the orchestration component needs to connect to the DE backend. This is done through a simple REST API provided by an Apache web Server¹⁰.

The **Notification Engine** (NE) component was developed using Java based frameworks Spring and Firebase on the server side and Firebase on the client side. The development process was performed using the Integrated Development Environment (IDE) "Eclipse", provided by Eclipse Foundation.

2.2 INDIVIDUAL COMPONENT TESTING

According to the QA Strategy described in the Introduction, each component was initially tested independently of the other components. Nevertheless, each individual component was developed with the respect to the agreed interfaces for the integration scenario.

Smart Meter and Sensor Data Manager (SMSDM) component was tested with camel-test modules with energy consumption and sensor data files (temperature, humidity, luminance, occupancy) provided by the Utility Companies that are partners in the project. Next, tests related to the functionality of the component were performed:

- retrieval of the raw files with consumption and sensor data from the Utility;
- processing of consumption files using threads for parallel processing;
- saving of processed data into the database.

Gamification Engine was tested using gamification data from a local database. A large range of functional tests was performed on the local environment to ensure the coverage of all the services required by the use cases specified in D2.3 (Functional System Specifications). The performance of the exposed service API's was tested using Apache Bench to ensure response time when the system receives request from several users concurrently.

Awareness Application was tested using consumption and gamification data from a local database. A large range of functional tests was performed against the local database, covering the use cases specified in D2.3 (Functional System Specifications).

Integrated tests were performed in the second phase of the development lifecycle:

- User registration
- Integration with the Service integration layer
- Access to the platform consumption database

Funergy Application was tested using a local database with 300 questions in 4 languages (English, Italian, German and Greek) and with the help of a Content Management System. A large range of functional tests was performed against the local database, covering the game specification defined on D 5.5 (Final

¹⁰ https://encompass.idsia.ch/webscript/cgi-bin/disaggregation_engine

Behavioral Game Concept). The performance of the game backend was tested using Apache Bench to ensure response time when the system receives request from several users concurrently.

Inference Engine was tested using energy consumption and environmental data coming from the Smart House located at CERTH (which also is one of the European Digital Innovation Hubs). The CERTH's smart house is equipped with energy consumption meters at both central panels, as well as at individual devices. Furthermore, each room is equipped with all kind of indoor environmental sensors (i.e. temperature, humidity, CO2, luminance, occupancy, etc.). In general, CERTH's smart house is equipped with more than 300 smart sensors metering almost everything in it. Finally, a large range of tests was performed on this data ensuring the accuracy of the output, as well as the quality of the outputs (deliverables D3.2, D3.4, D3.5, D4.1 and D4.3).

Recommendation Engine was tested using test data from local database. Manual tests aided by several tools were performed to validate the functionality of the component.

Disaggregation Engine was separately tested in order to assess functionality, reliability, security and performances of the module. The Disaggregation Engine producer is provided by a REST API, so it was decided to test the module by sending requests and analysing responses through a collaborator platform for API development. The tool used is called Postman (provided by Postman Inc.¹¹). The algorithm was tested by using the UK-DALE dataset (UK Domestic Appliance Level Electricity).

Notification Engine was tested internally with Junit framework using regression tests and manual tests. For allowing individual testing by other components of the enCOMPASS platform (such as Inference Engine and Recommendation Engine) a set of testing web services has been published.

2.3 INTEGRATED TESTING

During the Platform Prototype phase of the QA Strategy, end-to-end integration tests were performed. The integration engine of the Platform was ensured by the Enterprise Service Bus (ESB) component. As stated in the Description of Work, the Platform architecture was designed as a Service Oriented Architecture. Based on the Service Architecture, Platform components were designed to interact with each other using WebServices. Each component exposes and/or consumes WebServices.

The integration tests covered the following domains:

- basic, physical connectivity testing;
- authentication testing;
- end-to-end testing.

Connectivity testing. These tests had the role to ensure that Platform components can connect to the integration component ESB. Meaning that the IP address and ports of ESB End Points were accessible to each of Platform's component. URL names of WebServices exposed by the ESB were tested to be according to technical specifications.

Authentication testing. These tests had the role to ensure that Platform components can authenticate in the Platform to perform its designated actions. For example, the User Portal component is authenticated by the Platform to access energy consumption data.

¹¹ <https://www.getpostman.com/>

These tests also ensured that a non-authenticated access of the Platform was denied. For example, that a non-authorized application that has network access to the Consumption Service End Point does not have access to consumption data.

End-to-End testing. These tests overlap with the next testing domain: Functional Testing. In the context of Integration testing domain, these tests focused on interface testing. Meaning that messages exchanged respected the format and structure as stated in the technical specifications by all the components involved in the tested scenario.

2.4 FUNCTIONAL TESTING

Functional tests are defined starting from the functional requirements. Each requirement of the Platform was translated into one or more Test Cases, which were formalized with the following structure:

Table 1: Test Case structure.

<TC Number>	TC Name <TC Name>
Description	<Description of the test case>
Pre-Conditions	<State of involved components or modules>
Actions	<User or System ordered list of actions to be performed during the test>
Expected Result	<Description of the expected result after executing the ordered action list>
Actual Result	<Description of the actual result if is different from the Expected result>
Test Result	<Passed if Expected Result = Actual Result / Failed if Expected Result != Actual Result >

2.5 PERFORMANCE AND SCALABILITY TESTING

For each component of the Platform specific performance and scalability tests were designed and performed:

Awareness Application. Performance tests targeted:

- Service availability and response time with an increased number of concurrent users;
- Service availability and response time with a large volume of data: Subscribers and Consumption;
- Response time with an increased number of concurrent users and large data volume.

Service Integration layer. Performance tests concerned the availability and response time of exposed Web Services:

- Stress testing of WebServices.

SMMDC. The performance and scalability tests for this component was related processing of consumption data files:

- Processing time of large consumption data files;
- Adding a processing node or a storage node in Apache Hadoop architecture.

2.6 DEPLOYMENT TESTING

This testing domain is not related to the functionality or performance of the Platform but to ensure a correct deployment in the shortest possible time. The deployment plan contains the required steps to be performed to have an up and running Platform. The deployment plan must also embed environment specific parameters: machines, IPs, URLs, ports, file system, file structure.

Tests that covered this domain ensured that each deployment type is done with expected results:

- Deployment of Prototype Platform;
- Deployment of SES Demo Case:
 - Test Environment;
 - Production Environment;
- Deployment of SHF Demo Case:
 - Test Environment;
 - Production Environment;
- Deployment of WVT Demo Case:
 - Test Environment;
 - Production Environment;
- What is to be deployed:
 - Core repository;
 - Deployment specific.

3 QUALITY ASSESSMENT METRICS

After defining the testing domain (what to test), the most adequate **QA Metrics** were selected to set the quality standards of software development process.

Following **QA Metrics** were decided to be monitored during the entire development lifecycle of the Platform.

3.1 CODE SIZE METRICS

Metrics name	Meaning
NCSS (Non-commenting source statements)	Used to: - estimate the order of magnitude of the application: 10k of lines of code, 10M lines of code; - estimate the required maintenance effort; - serve as the basis for various Code Quality metrics.
Number of classes and interfaces	Used to: - estimate the order of magnitude of the application: 1k of classes and interfaces, 10k of classes and interfaces; - estimate the required maintenance effort; - serve as the basis for various Code Quality metrics

3.2 CODE QUALITY METRICS

Metrics name	Unit	Meaning
Code Coverage	ratio	Percent of code that is run when an automated test suite is performed
Afferent Couplings	scalar value	The number of other packages that depend upon classes within the package is an indicator of the package's

		responsibility (Apache Commons ¹²)
Efferent Couplings	scalar value	The number of other packages that the classes in the package depend upon is an indicator of the package's independence (Apache Commons).
Abstractness	ratio	The ratio of the number of abstract classes (and interfaces) in the analyzed package to the total number of classes in the analyzed package. The range for this metric is 0 to 1, with A=0 indicating a completely concrete package and A=1 indicating a completely abstract package (Apache Commons).
Instability	ratio	The ratio of efferent coupling (Ce) to total coupling (Ce / (Ce + Ca)). This metric is an indicator of the package's resilience to change. The range for this metric is 0 to 1, with I=0 indicating a completely stable package and I=1 indicating a completely unstable package (Apache Commons).
Cycles	percent	Packages participating in a package dependency cycle are in a deadly embrace with respect to reusability and their release cycle. Package dependency cycles can be easily identified by reviewing the textual reports of dependency cycles. Once these dependency cycles have been identified with JDepend, they can be broken by employing various object-oriented techniques (Apache Commons).

3.3 PERFORMANCE METRICS

Metrics name	Unit	Meaning
Time taken for tests	seconds	the observation interval
Concurrency Level	ratio	total execution time of all queries during the observation interval
Complete requests	scalar value	number of completed requests during the observation interval
Failed requests	scalar value	number of failed requests during the observation interval
Total transferred	bytes	number of bytes transferred between client and server during the observation interval
HTML transferred	bytes	number of HTML bytes transferred between client and server during the observation interval
Requests per second	scalar value	number of client requests per second during the observation interval
Time per request	seconds	mean time to process a request during the observation interval
Transfer rate	scalar value	Total transferred / Time taken for tests

Smart Meter and Sensor Data Manager (SMSDM) test:

¹² <https://commons.apache.org/proper/commons-daemon/jdepend-report.html>

The performance and scalability tests for this component were related to optimal processing of energy consumption and sensor data files. The performance testing regarded:

- processing time of large consumption and sensor data files;
- adjusting the amount of processing memory for an optimal processing time.

Gamification Engine Performance test:

The main exposed services of gamification engine were tested, the full detail of the individual tests is included in the Appendix A.

	Time taken for tests (Seconds)	Concurrency Level	Complete requests	Failed requests	Total transferred (bytes)	HTML transferred (bytes)	Requests per second	Time per request (ms)	Transfer rate (kb/s)
AddUsageLog	1.654	10	100	0	45000	18500	60.47	16.546	26.58
getAction	2.467	10	100	0	45200	18500	40.54	24.665	17.90
getActions	2.434	10	100	0	44800	18500	41.08	24.342	17.97
assignActionsToUsers	1.632	10	100	0	46800	18500	61.27	16.32	28
assignExternalAction	1.633	10	100	0	47500	18500	61.23	16.332	28.40
getAreas	1.649	10	100	0	44400	18500	60.64	16.491	26.29
getBadge	1.659	10	100	0	44900	18500	60.27	16.591	26.43
getBadges	1.635	10	100	0	44600	18500	61.17	16.348	26.64
getGoal	1.661	10	100	0	44900	18500	60.19	16.614	26.39
getLeaderboard	1.642	10	100	0	45600	18500	60.89	16.424	27.11
getReward	1.604	10	100	0	45100	18500	62.34	16.040	27.47
getRewards	1.666	10	100	0	44800	18500	60.03	16.658	26.26
getUser	1.663	10	100	0	44900	18500	60.15	16.626	26.37
getUserActions	1.615	10	100	0	48300	18500	61.94	16.94	29.21
getUserBadges	1.657	10	100	0	47000	18500	60.35	16.570	27.70
getUserCredits	1.670	10	100	0	46800	18500	59.90	16.696	27.37
getUserGoals	1.652	10	100	0	46400	18500	60.53	16.520	27.43
getUserRewards	1.670	10	100	0	46400	18500	59.88	16.699	27.13
setGoal	1.653	10	100	0	44200	18500	60.48	16.533	26.11

Funergy Application Backend Performance Test:

The backend services of the Funergy Application were tested, the full detail of the individual tests is included in the Appendix A.

	Time taken for tests (Seconds)	Concurrency Level	Complete requests	Failed requests	Total transferred (bytes)	HTML transferred (bytes)	Requests per second	Time per request (ms)	Transfer rate (kb/s)
getLocalizedQuestion	0.791	10	100	0	30400	4200	126.49	7.906	37.55
getNextQuestion	1.140	10	100	0	34900	12200	87.72	11.400	29.90
getNextQuestionForTag	1.881	10	100	0	34900	12200	53.17	18.806	18.12
getAvailableTags	1.103	10	100	0	268700	243000	90.65	11.031	237.88

Inference Engine Performance Test:

The Inference Engine services were tested:

	Time taken for tests (seconds)	Failed response per user	Defects and failures	Integration testing	Effectiveness	System testing	Operational acceptance testing	Completion rate
Thermal Comfort inference	1.100	0	0	Successful	100%	Successful	Successful	1

Visual Comfort inference	0.700	0	0	Successful	100%	Successful	Successful	1
Occupancy inference	0.300	0	0	Successful	100%	Successful	Successful	1
Activity Inference	0.500	0	0	Successful	100%	Successful	Successful	1

Recommendation Engine Performance Test:

From the end-user point of view the Recommendation Engine is an out of process service which operates outside of the request cycle. It is triggered as a batch process, and it does not provide a HTTP interface. The above-mentioned performance metrics are therefore not applicable for this component.

Disaggregation Engine Performance test:

Given that the Disaggregation Engine processes very large amount of data at each cycle of execution, it is independently called on an external server. Some internal tests have been executed in order to assess functionality, reliability, security and performances of the modules, but since the DE processing is not a blocking point for the orchestration chain, the mentioned performance metrics are therefore not applicable for this component.

Notification Engine Performance test:

Delivering push notifications is a critical responsibility of Notification Engine component on behalf of enCOMPASS platform. In the context of the specific business logic of the components making the platform, the following parameters have been observed:

- internal messages generated by the platform components
- internal messages transformed into user notifications
- user notifications sent
- user notification delivered
- user notifications open

3.4 RELIABILITY METRICS

Metrics name	Unit	Meaning
MTBF	Time Average	Mean Time Between Failure
MTTF	Time Average	Mean Time to Failure
MTTR	Time Average	Mean Time to Repair
Availability	Percentage	Steady state availability.

Software reliability is the probability that software component will work properly in a specified environment and for a given amount of time. Using the following formula, the probability of failure is calculated by testing a sample of all available input states. Mean Time Between Failure (MTBF)=Mean Time To Failure (MTTF)+ Mean Time To Repair (MTTR). Steady state availability represents the percentage the software is operational, and can be calculate as MTTF/MTBF.

In order to calculate the MTTF, tools are needed to keep track of the errors and crashes of the system. There are several tools that enable the automatic detection of errors like Firebase Crashlytics, mobile frameworks provide their own set tools for error detection and issue tracking. In the context of an Android Application the Google Play console provide the tools to analyse errors over a period of time.

Figure 4 shows an example of an android crash report of the enCOMPASS application, it displays the occurrences of errors that impacted the application over a period of time. In this example, the observed period is 30 days and the number of errors is 3, such information can be used to calculate the MTTF.

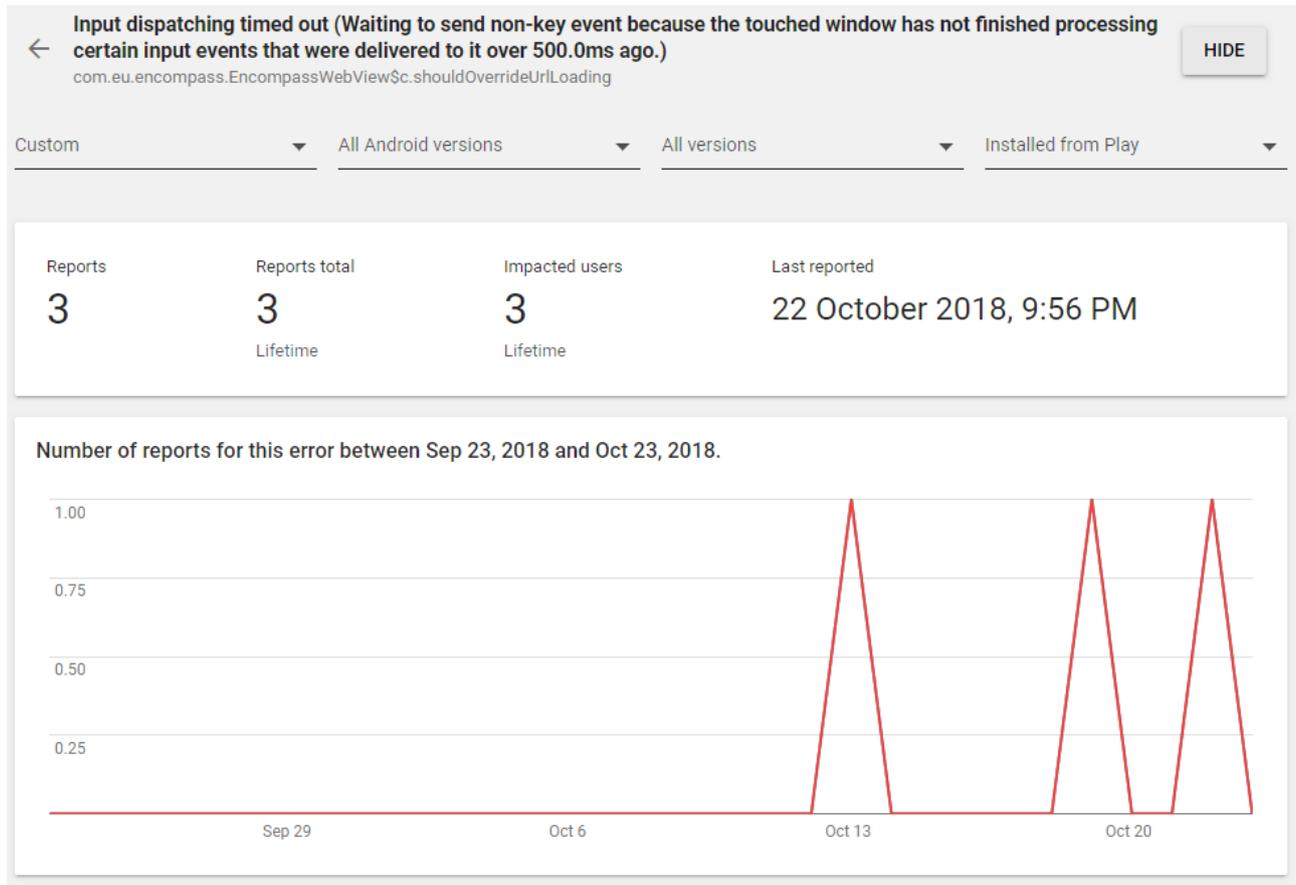


Figure 4– Android Crash Report

To calculate the MTTR it is necessary to keep track of the code history to identify the time between code fixes. Modern code versioning systems like GIT and BitBucket provide tools for issue and history tracking (Figure 5). On GIT web interface, the “commits” view provide the list of code changes including the date the change was submitted and the description of the fixed error. On projects implementing Agile methodologies the code changes are submitted to the “Master” branch only after they have been tested and approved, therefore the list of changes on the master branch should be used as the reference time track to calculate the MTTR.

During the first six months of the operation of the platform, the Crash Analytics reported 9 errors, MTTF=480h. The process of identifying, fixing and deploying the solution took on average 24h (MTTR). The availability of the AA of the system during this period would be approximately of the 95.2%.

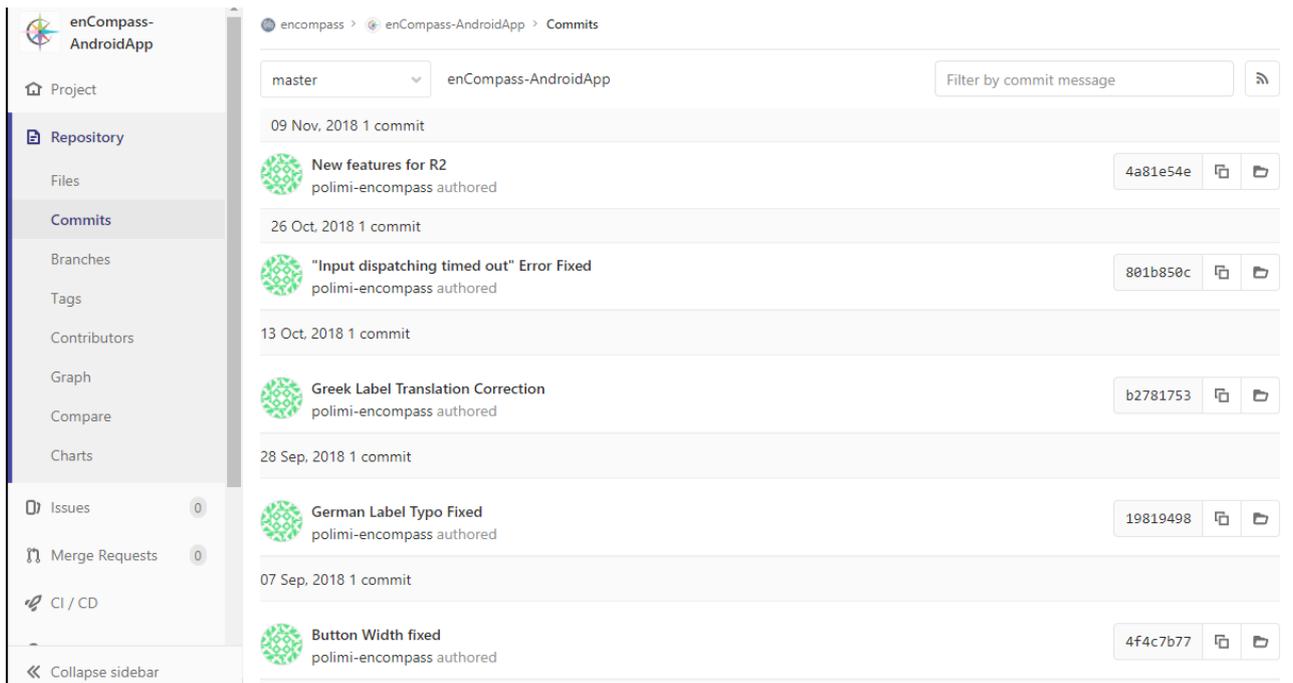


Figure 5- GIT Commit History

4 DEPLOYMENT AND QUALITY ASSURANCE TOOLS

In this section, we describe all the Tools used during the development of the EnCompass architecture that allows the Quality Assurance.

4.1 DEPLOYMENT ENVIRONMENT (IDE)

There are some obvious QA benefits from usage of specialized tools for development:

- Clear code organization and editing leads to a clearer code, solid code architecture, reduced number of bugs and reduced correction time;
- Automation of tasks such as: code analysis, compiling, packaging and deployment eliminates or drastically reduces human error that could occur during such tasks;
- They are extendible applications which can add various components that can improve code quality: Code Analysers and Code Optimisers.

The EnCompass architecture has been developed using 3 IDEs: Eclipse IDE (SMMDC modules) and WebRatio (Consumer Portal, Gamification Engine, Admin Portal etc), Spyder (Machine learning components).

Eclipse IDE

- The most used Java Enterprise Applications (J2EE) development tool, that has a base workspace and an extendible architecture that allows integration of various plugins.
- Allowed installation of various plugins that contributed to an integrated development infrastructure, ensuring quality by providing:
 - o Version control integration plugin;
 - o Package building and deployment plugin;
 - o Code optimisation plugin.

Checkstyle violation type	Marker count
Wrong lexicographical order for 'X' import. Should be before 'X'.	1
'X' construct must use '{}'.s.	1
'X' is followed by whitespace.	1
'X' is preceded with whitespace.	1
Abbreviation in name 'X' must contain no more than 'X' consecutive ca...	1
'X' should be separated from previous import group by one line.	1
Line is longer than X characters (found X).	12
Name 'X' must match pattern 'X'.	15
'X' should be separated from previous statement.	15
First sentence of Javadoc is incomplete (period is missing) or not present.	15
'X' is not preceded with whitespace.	23
'X' is not followed by whitespace.	25
'X' child have incorrect indentation level X, expected level should be X.	58
'X' have incorrect indentation level X, expected level should be X.	72
Line contains a tab character.	577

Figure 6 - CheckStyle error report

WebRatio

- An Eclipse based IDE, that shares the same workspace as all the tools and extensions already available in Eclipse for developing J2EE Applications;
- Uses IFML modelling standard to define the interaction flow between the User and the Application;
- Standard based (IFML) automatically generated code;
- Automatic model checking and quality assurance tool that verifies errors and warning in the IFML models.

Description	Resource
ArrayList is a raw type. References to generic type ArrayList<E> should be parameterized	FacebookUser.java
ArrayList is a raw type. References to generic type ArrayList<E> should be parameterized	GooglePlusUser.java
ArrayList is a raw type. References to generic type ArrayList<E> should be parameterized	LinkedInUser.java
Iterator is a raw type. References to generic type Iterator<E> should be parameterized	LinkedInUser.java
List is a raw type. References to generic type List<E> should be parameterized	FacebookUser.java
List is a raw type. References to generic type List<E> should be parameterized	GooglePlusUser.java
List is a raw type. References to generic type List<E> should be parameterized	LinkedInUser.java
List is a raw type. References to generic type List<E> should be parameterized	LinkedInUser.java
List is a raw type. References to generic type List<E> should be parameterized	ResultFragment.java
List is a raw type. References to generic type List<E> should be parameterized	ResultFragment.java
List is a raw type. References to generic type List<E> should be parameterized	ResultFragment.java

Figure 7 – WebRation Problem Report View.

Spyder

- Python based IDE;
- Offers advanced editing, analysis, debugging, and profiling functionality;

- Offers data exploration, interactive execution and deep inspection;
- Beyond its many built-in features, its abilities can be extended even further via its plugin system and API.

4.2 CODE QUALITY CHECKING TOOLS

Code quality was improved by the usage of several tools:

- **CheckStyle** – Source Code Formatter for Java;
- **JDepend** – Source Code Analyzer;
- **YourKit** – Profiling Tool;
- **Java NCSS** – Source Code Analyzer;
- **FindBugs** – Bug detector tool;
- **Emma** – Source Code Analyzer;
- **WebRatio** – Model Validator;
- **Pylint** – Source code analyzer.

CheckStyle

- Intelligent line wrapping;
- Scope related indentation;
- Brace style transformation;
- Insertion of parentheses and braces;
- Blank line and white space formatting;
- Semantic source code separation;
- Sorting of source code elements;
- Insertion and substitution of header and footer;
- Conversion between character and end-of-line encodings;
- Javadoc validation, formatting and template-driven generation.

JDepend:

- Traverses Java class file directories and generates design quality metrics for each Java package;
- Allows you to automatically measure the quality of a design in terms of its extensibility, reusability, and maintainability to effectively manage and control package dependencies.

YourKit

- CPU usage graph showing total, kernel and garbage collector times is always available;
- CPU views present results as call trees, hot spots, method lists, back traces, merged calls and calls lists;
- Memory usage graphs show heap and non-heap memory pools, GC activity and, if recorded, object creation rate per-second;
- Comprehensive heap inspection and analysis;
- Object allocation recording to solve garbage collection and memory allocation issues. Available settings allow to balance between result fullness and profiling overhead. In particular, consider the unique object counting mode with almost zero overhead;
- Object explorer to inspect individual objects;

- Garbage collection profiling to estimate garbage collector load, and, if garbage collection takes a significant amount of time, pin-point the problematic code;
- Thread profiling: monitor thread states and stacks, estimate CPU usage in a time range;
- Deadlock detector;
- Exception profiling: where thrown, of what class and how many;
- Event recording: in addition to low level profiling results such as method calls, the profiler can record higher level events with their essential properties such as database queries, web requests and I/O calls. Use built-in probes to recognize typical problems or write your own to inspect specifics of your application.

Java NCSS

- Metrics can be applied to global-, class-, or function-level;
- Non-Commenting Source Statements (NCSS);
- Cyclomatic Complexity Number (McCabe metric);
- Packages, classes, functions and inner classes are counted;
- Number of formal Javadoc comments per class and method.

FindBugs

- Makes static analysis to look for bugs in Java code;
- It detects bugs like:
 - o “correctnes bug”: Probable bug, an apparent coding mistake resulting in code that was probably not what the developer intended;
 - o “bad practice”: Violations of recommended and essential coding practice. Examples include hash code and equals problems, cloneable idiom, dropped exceptions, serializable problems, and misuse of finalize;
 - o “dodgy”: Code that is confusing, anomalous, or written in a way that leads itself to errors. Examples include dead local stores, switch fall through, unconfirmed casts, and redundant null check of value known to be null.

Emma

- Open source toolkit for measuring Java Code Coverage;
- Can instrument classes for coverage offline (before they are loaded) or “on the fly”;
- Can support coverage for: class, method, line, basic block.

WebRatio

The WebRatio platform provides a built-in validation tool for the IFML models, which objective is to ensure the model consistency and correctness; executing the validation process before the automatic code generation prevents both code creation and compilation errors and ensure code optimization. The validation process can be executed at any step of the development process, it provides a list of the problems found along with the type of problem, the related component, and offers a quick solution for each problem.

The validation tools classify problems as errors and warnings; “Errors” are inconsistencies in the model that prevents the code generation, caused by of missing elements or references, duplication of component IDs, or incomplete cycles in interaction flows.

Some of the most common types of error are:

- The entity for the component is unspecified: This kind of error indicates that a component of a form, page or service was not associated with an entity of the database.
- Missing database table for entity: Indicates that the model is referencing an entity that does not exist in the database, the presence of this error often indicates that the domain model is not synchronized with the database.
- Missing Home element for site view: This error is generated when a site view is created without a default page, this page is defined to be displayed with the view is access.
- The custom URL name is duplicated: When a specific URL is created for an element, the validation process verifies that there no duplicated URL that may create conflicts on the application.

The parameter binding references an input parameter not provided by the flow's target: The validation process ensures that data binding between elements is consistent, ensuring that the parameter passing between elements match perfectly and there is no reference to non-existing elements.

Fixing this kind of problems ensures that the generated code will be consistent, and will be optimized by the generation process since it will not create variable or references that will never be used.

Problems classify by the validation tool as “Warnings”, on the other hand, represent identified situations on the model or in external sources, such as plugins or coded components, that may result on exception or unexpected errors during the execution of the code, but do not prevent the code generation.

Most common warnings found are:

- Java Deprecation Error: This warning is common when external components are added to the platform, the tool validates that the code used in java or groovy sources is consistent with the version of the platform. Although deprecation does not present a problem for the operation of a component, it may represent a problem in future versions.
- Reference to Generic Types in Java Sources: As in deprecation, the tool validates correct use of the components according to the Java version, and encourage the use of generics in Java collections, although it does not represent an issue for the operation of the component.
- No attributes to display are specified for the component: The tool validates that elements intended to display information such as list or details, effectively have selected field to be shown on the interface.
- The component is never used in module: In order to ensure code optimization, the tool warns about any element that has no assigned behaviour and no interactions with other elements, these elements should be removed from the project.
- A hidden field should not be modifiable: The tool verifies consistency in the behaviour of form components, hidden elements should not be displayed in the user interface and should not be modifiable. Removing the modifiable property from these elements ensure the generated code is syntactically and conceptually correct.

Warnings do not offer a quick fix option as errors do, but an indication of the element with problem is provided by the tool.

Once all the errors are resolved, the automatic code generation warranties that the generated code is correct, efficient and optimal, as it lacks unused code and meaningless variable or relationship.

Pylint

Pylint is a tool searching for errors in Python code. It tries to enforce a coding standard. It can look for certain type of errors, while it can make recommendations/ suggestions about how particular blocks can be refactored. Finally, it provides you information about the code's complexity.

Some indicative errors that it can detect are listed below, while a full list can be found at Pylint¹³:

- “ignore”: Hardcoded names and paths;
- “jobs”: Use multiple processes to speed up;
- “unsafe-load-any-extension”: Allow loading of arbitrary C extensions. Extensions are imported into the active Python interpreter and may run arbitrary code.

4.3 VERSION CONTROL TOOLS

As Version Control Tool it has been decided to use **Git** as distributed Version Control System (VCS) for tracking changes of the software components making the enCOMPASS platform.

Git main roles were:

- code projects and repositories organization;
- manage version control of source code;
- issue tracking.

Project	Key	Description
 COMMON Repository		Common code repositories
 EnCompass 1st Prototype		First prototype of the EnCompass platform code repositories
 SHF Repository	SHFR	SHF Demo Case code repositories
 SES Repository	SESR	SES Demo Case code repositories
 WVT Repository	WVTR	WVT Demo Case code repositories

4.4 ISSUE TRACKER

Issue tracking was performed using **Git**. Issue tracking system covered the entire quality assurance process related to code development:

issue reporting;

- issue responsible assignment;
- priority assignment:
 - o minor
 - o major
- issue workflow from identification to resolution through various issue statuses:
 - o open
 - o on-hold
 - o duplicated

¹³ https://pylint.readthedocs.io/en/latest/technical_reference/features.html

- invalid
- closed
- commenting and collaboration;
- file attachment.

Issues are recorded at Repository level.

4.5 PERFORMANCE TESTING TOOLS

Performance Tests have been performed using **Apache Benchmark** (by Apache Software Foundation). This utility is a free open-source software that allows to measure the performance of HTTP web servers. It was used to test the performance of the WebServices API expose by the several components.

The tool allows developers to simulate any number of concurrent users sending any number of requests.

5 PERFORMED TESTS

In this Section we describe how we used the Testing Tools above mentioned to test EnCompass components and present a few examples of the quality assurance reports generated with the tools and methods described in the preceding sections. These reports have been used continually during the whole development lifecycle and have guaranteed the successful management of the EnCompass platform the deployment of the releases in 3 distinct real-life use cases.

Following the list of EnCompass components described in “D6.5 - PLATFORM IMPLEMENTATION AND INTEGRATION – FINAL PROTOTYPE” and their testing protocols.

See Appendix A for the complete list of tests and results both to test the Performance and the Functionalities of each Component.

5.1 THE SERVICE INTEGRATION AND ORCHESTRATION COMPONENT

Code quality: This component has been created using Eclipse. The Code Quality is ensured by the IDEs features.

Performance and Functional Testing:

The following procedure describes the steps to set up the testing environment for the enCompass backend services, instantiated on each one of three pilot servers.

This Functional Test Plan consists of three different parts: *Service Calculations*, *Component Orchestration* and *Component Messages*. For these tasks, the tester will have to access the database and the backend services’ APIs of each pilot server.

The backend services’ APIs are exposed by a Swagger interface that is accessible on the development and test server and on each pilot server with specific parameters: <http://{server-ip}:port/swagger-ui.html>

For each server instance, the tester executes the tasks described below.

1. Service Calculations

1. Extract into a spreadsheet the corresponding DB values from the tables *meter_consumption*, *indoor_conditions_humidity*, *indoor_conditions_temperature* and *baseline*
 2. Calculate the averages, baselines, savings according to specific logic
 3. Confront the results against the results provided by the following services exposed by Swagger:
 1. GET /si/users/{id}/consumption
 2. GET /si/users/{id}/consumption/summary
 3. GET /si/users/{id}/humidity/average
 4. GET /si/users/{id}/temperature/average
 5. GET /si/users/{id}/baseline
 6. GET /si/users/{id}/savings
 4. Repeat the operation for all each pilot
2. Component Orchestration
 1. Search for the content of the *semaphore_log* table and sort it in descending order by the timestamp.
 2. Identify the component orchestration sessions.
 3. Check that the component results save a return code that is not -99 (exception).
 3. Component Messages
 1. Search for the content of the *message_instance* table and sort it in descending order by the *timestamp_creation*.
 2. Identify the messages that have been produced and stored by IE and RE components after a component orchestration session.

5.2 THE AWARENESS APPLICATION FOR WEB AND MOBILE ACCESS

Code quality: This component has been created using WebRatio, the Code Quality is ensured by WebRatio automatic code generation mechanism. See section 2.1 Code Quality Testing.

Performance and Functional testing: to perform performance testing on the Awareness Application, Selenium scripts have been implemented, Functional testing was executed on test cases based on the main use cases including:

Section	Test Case
Login	The user successfully login to the application
Tips	The user enters the tip section and navigates the available tips
	The user reads a tip and provides feedback about it
	The user reads a tip and watches the video it contains
Profile	The user enters the profile page
	The user edits the household information on the profile page
Achievements	The user visualizes his achievements and scrolls the previously performed actions
	The user visualizes the leader board and scrolls

	to see his position
Reward	The user enters the reward section and navigates the available rewards
Consumption	The user enters the consumption section and visualizes its daily consumption
	The user enters the consumption page and navigates the consumption bar changing the period
	The user enters the consumption page and changes the granularity (daily, weekly, monthly) and visualizes the corresponding average.
Impact	The user enters the goal page and visualize the progress towards its goal, the user visualizes the disaggregate consumption information.
	The user enters the goal page, selects a new goal and saves it
	The user enters the impact page to visualize its saving impact, the user selects a different visualization from the available option, his impact is displayed with the selected visualization type
	The user enters the comfort page, visualize the comfort levels of the current month, the user navigates the comfort levels of the previous months

5.3 THE GAMIFICATION ENGINE

Code quality: This component has been created using Webratio, the Code Quality is ensured by Webratio automatic code generation mechanism. See section 2.1 Code quality Testing.

Performance and Functional Testing: Apache Bench tests have been executed on all the exposed application services.

UserActivityCreditWebServiceREST/AddUsageLog/AddUsageLog
UserActivityCreditWebServiceREST/GetAction/getAction
UserActivityCreditWebServiceREST/GetActions/getActions
UserActivityCreditWebServiceREST/AssignActionsToUsers/assignActionsToUsers
UserActivityCreditWebServiceREST/AssignExternalActionToUsers/assignActionsToUsers
UserActivityCreditWebServiceREST/GetAreas/getAreas
UserActivityCreditWebServiceREST/GetBadge/getBadge
UserActivityCreditWebServiceREST/GetBadges/getBadges
UserActivityCreditWebServiceREST/GetGoal/getGoal
UserActivityCreditWebServiceREST/GetLeaderboard/getLeaderboard
UserActivityCreditWebServiceREST/GetReward/getReward
UserActivityCreditWebServiceREST/GetRewards/getRewards
UserActivityCreditWebServiceREST/GetUser/getUser
UserActivityCreditWebServiceREST/GetUserActions/getActions
UserActivityCreditWebServiceREST/GetUserBadges/getBadges
UserActivityCreditWebServiceREST/GetUserConsumptionGoals/getGoals
UserActivityCreditWebServiceREST/GetUserCredits/getUserCredits

UserActivityCreditWebServiceREST/GetUserGoals/getUserGoals
 UserActivityCreditWebServiceREST/GetUserRewards/getRewards
 UserActivityCreditWebServiceREST/SetGoal/setGoal

5.4 THE ENERGY EFFICIENCY CONSOLE FOR UTILITY AND BUILDINGS

Code quality: This component has been created using AngularJS, while the Code Quality is ensured by Jenkins, which has been used during the developed of the platform.

Performance and Functional Testing: Functional testing on the Energy Efficiency Console for Utility and Buildings can be performed by various use cases including:

Section	Test Case
Login	The user successfully login to the platform
Dashboards	The user is able to create his/her own dashboards
	The user is able to add at the dashboards his/her own widgets
	The user is able to visualize the consumption data using different widgets
Comparisons	The use is able to perform consumption comparisons for different periods
Reports	The user can create his/her own reports
	The user can have the reports periodically through email
	The user can modify existing reports

5.5 THE DISAGGREGATION ENGINE

Code quality: This component has been created using Eclipse and PyCharm, the Code Quality is ensured by the IDEs features. See section 2.1 Code quality Testing.

Performance and Functional Testing: The Disaggregation Engine (DE) component processes aggregated consumption data and returns the estimated end-uses of the single devices in a household. The services provided by this component are the producer and the consumer. Functional tests implemented were aimed at verifying the functional requirements and specifications, in order to ensure that they are properly satisfied by the component. For this purpose, a test protocol was designed both for the producer and the consumer services and a sample of five users was tested (see Appendix A).

The test protocol implemented in order to ensure that the DE producer service is able to process all the data retrieved from the database consisted in verifying the presence of one entry for each user in the encompass_model.disaggregation_data table, downstream the algorithm processing.

The functional test performed in order to ensure that the consumer service works as expected consists of three basic steps:

1. Verify that at least 10 days of disaggregated data have been computed by the producer service. In detail this means to check that at least 10 entries were recorded in the database;

2. Ensure that the sum of the disaggregated consumption of the appliances (fridge, washing machine, tumble dryer, dishwasher, AC, electric car, electric oven, heat pump, other) for each user over the 30 days is equal to the total consumption over the same period;
3. Verify that the values provided by the consumer service coincide with the sums mentioned above.

5.6 THE NOTIFICATION ENGINE

Code quality: Code quality was implemented using FindBugs for static analysis of the Java code and for bug detection, Emma for offline coverage of the code and CheckStyle for standardizing and style alignment the source code. Additionally, JDepend was used for source code analyses such as checking Java class file directories and generating design quality metrics for each Java package.

Performance and Functional Testing:

Then the following actions have been performed according to the test plan.

1. Creating a notification and posting it to one of the message queues

Open SWAGGER-UI in a browser and call `postMessageInstanceservice` (@SI Controller - http://server-IP:port/swagger-ui.html#!/SI_Endpoint/postMessageInstanceUsingPOST_1)

using the following JSON:

```
{
  "generic_message_oid": 30, //replace 30 with the oid of the notification from the generic_message
  table
  "hidden": true,
  "is_static": true,
  "userId": 1 //replace 1 with the userId from the user table
}
```

Each notification is created based on the `generic_message_oid` and it has a priority specified by `notification_type.default_priority`. The currently assigned priorities are set for testing purpose (Eg. `consumption_keepontrack` notification type has a high priority assigned). It is possible to change the notification type priorities according to the real scenario by going to the `notification_type` table and modifying the values in the column 'default_priority'. The only accepted values are low and high according to the queue's types.

2. Check the content of the queues
The specific service URL is open in the browser <http://server-IP:port/ne/amq/browse>
3. Force processing the queues to send the notifications
<http://server-IP:port/ne/amq/process>
4. Empty the queues and delete the delivered notifications (from the `notification_delivery` table)

5.7 THE RECOMMENDATION ENGINE

Code quality: this component is implemented in python programming language, and was developed using IntelliJ PyCharm, an integrated development environment for python. The development of the production code was aided by the Python code insight tools of PyCharm. The Code Quality is ensured by the IDEs features.

Jupyter Notebook, a web-based interactive environment for statistical modelling, data visualization, and machine learning, was used for algorithm prototyping.

Performance and functional testing: For the functional testing of the RE a separate testing environment was created on SES pilot test instance. One important requirement was set up for RE testing by the partners: GRA needed to show that the result of the test (ie. the same user receives the same recommendation based the given dataset) is the same whenever the test is run. As this logic contradicts with the normal logic of RE, a special testing environment was built (as it is described in 6.5.1) to show the results on static dataset. However, it worth to note that in practice as the background data is changing the calculated best recommendation for a given user can be different from the previously calculated one. This the intended functioning of the RE.

5.8 THE INFERENCE ENGINE

Code quality: This component has been developed at Python, while the Code Quality is ensured by Pylint tool. See section 2.1 Code quality Testing.

Performance and Functional Testing: The Inference Engine is comprised by four main components, i.e. visual comfort inference, thermal comfort inference, occupancy inference and activity inference. It works autonomous, retrieving data from the enCOMPASS DB, while the output is stored again at the DB in order to be used by other components of the enCOMPASS framework.

5.9 FUNERGY – DIGITAL GAME EXTENTION OF THE BOARD GAME

Code quality: This component has been created using IFMLEdit.org, the Code Quality is ensured by IFML automatic code generation mechanism.

Performance and Functional Testing: The backend of the application has been tested with Apache Bench, tests have been executed on all the exposed application services.

- Services/ffv/getLocalizedQuestion
- Services/ffv/getNextQuestion
- Services/ffv/getNextQuestionForTag
- Services/ffv/getAvailableTags

6 APPENDIX A

6.1 SERVICE INTEGRATION AND ORCHESTRATION

6.1.1 Service Integration and Orchestration Performance Tests

Stress tests of the Platform’s WebServices were performed using Apache Benchmark testing tool. For the list of the services of enCOMPASS see “D6.2 – Platform Architecture and Design”.

In this section we show the results of the Performance and Scalability Tests performed on the service `getConsumptionSummary` which returns the summary of consumption for a given user in a specific interval of time. In this test (see Table 2, Table 3 and Table 4) we performed 3615 requests with a 100 Concurrency Level (100 requests in parallel).

Table 2: Scalability test results for the `getConsumptionSummary` WebService.

Metric	Result
Document Length	1385 bytes
Concurrency Level	100
Time taken for tests	85.467 seconds
Complete requests	3615
Failed requests:	0
Total transferred	6040665 bytes
HTML transferred	5006775 bytes
Requests per second	42.30 [#/sec] (mean)
Time per request	2364.234 [ms] (mean)
Time per request	23.642 [ms] (mean, across all concurrent requests)
Transfer rate	69.02 [Kbytes/sec] received

Table 3: Connection Times (ms) results for the `getConsumptionSummary` service performance test.

	<i>min</i>	<i>max</i>	<i>mean[+/-sd]</i>	<i>median</i>	<i>SD</i>
<i>Connect</i>	4	31110	59	11	932.5
<i>Processing</i>	24	1690	591	510	291.6
<i>Waiting</i>	24	1687	585	504	290.3
<i>Total</i>	32	31182	650	536	955.8

Table 4: Percentage of the requests served within a certain time (ms) results for the `getConsumptionSummary` service performance test.

Percentage	Time (ms)
50%	536
66%	650
75%	765
80%	818
90%	1004

95%	1265
98%	1462
99%	1584
100%	31182 (longest request)

6.1.2 Service Integration and Orchestration Functional Tests

Examples of Functional tests executed for these components are follow:

Service Calculations

Test	Service Calculation
Description	The user verify that dates are correct
Pre-Conditions	User has access to the database and the backend services' APIs of each pilot server.
Actions	<ol style="list-style-type: none"> 1. The user extract in an Excel sheet the corresponding DB values from the tables meter_consumption, indoor_conditions_humidity, indor_conditions_temperature and baseline 2. Calculate the averages, baselines, saving according to specific logic 3. Confront the results against the results provided by the following services exposed by Swagger: <ul style="list-style-type: none"> • GET /si/users/{id}/consumption • GET /si/users/{id}/consumption/summary • GET /si/users/{id}/humidity/average • GET /si/users/{id}/temperature/average • GET /si/users/{id}/baseline • GET /si/users/{id}/savings 4. Repeat the operation for all the Pilot DBs
Expected Result	The values are the same
Actual Result	The values are the same
Test Result	Passed

Component Orchestration

Test	Component Orchestration
Description	The user verify that Orchestration works properly
Pre-Conditions	User has access to the database and the backend services' APIs of each pilot server.
Actions	<ol style="list-style-type: none"> 1. Search for the content of the <i>semaphore_log</i> table and sort it in descending order by the timestamp. 2. Identify the component orchestration sessions. 3. Check that the component results save a return code that is not -99 (exception). 4. Ignore <i>import_co2</i> process outcome that is not ran - it is logged formally only.
Expected Result	The return value is valid
Actual Result	The return value is valid
Test Result	Passed

Component Messages

Test	Component Messages
Description	The user verify that Messages works properly
Pre-Conditions	User has access to the database and the backend services' APIs of each pilot server.

Actions	<ol style="list-style-type: none"> 1. Search for the content of the <i>message_instance</i> table and sort it on descending order by the <i>timestamp_creation</i>. 2. Identify the messages that have been produced and stored by IE and RE components after a component orchestration session.
Expected Result	The messages are stored by IE and RE
Actual Result	The messages are stored by IE and RE
Test Result	Passed

6.2 GAMIFICATION ENGINE AND AWARENESS APPLICATION (AA)

6.2.1 Gamification Engine Performance Tests

The following are examples of the test performed with Apache Bench on some of the services exposed by the gamification engine:

Service addUsageLog:

```

This is ApacheBench, Version 2.3 (<$Revision: 1843412 $>)
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking myencompass.supsi.ch (be patient).....done

Server Software:      nginx/1.12.2
Server Hostname:     myencompass.supsi.ch
Server Port:         80

Document Path:       /community/UserActivityCreditWebServiceREST/AddUsageLog/AddUsageLog
Document Length:     185 bytes

Concurrency Level:   10
Time taken for tests: 1.654 seconds
Complete requests:   100
Failed requests:     0
Non-2xx responses:   100
Total transferred:   45000 bytes
Total body sent:     29500
HTML transferred:    18500 bytes
Requests per second: 60.47 [#/sec] (mean)
Time per request:    165.358 [ms] (mean)
Time per request:    16.536 [ms] (mean, across all concurrent requests)
Transfer rate:       26.58 [Kbytes/sec] received
                    17.42 kb/s sent
                    44.00 kb/s total

Connection Times (ms)
   min  mean[+/-sd] median   max
Connect:    15    16   0.4    16    16
Processing:  16   139  25.7   146   159
Waiting:    16   101  37.1   111   147
Total:      32   155  25.7   162   175

Percentage of the requests served within a certain time (ms)
 50%    162
 66%    163
 75%    163
 80%    163
 90%    163
 95%    163
 98%    164
 99%    175
100%    175 (longest request)

```

Service getAction:

```
This is ApacheBench, Version 2.3 <$Revision: 1843412 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking myencompass.supsi.ch (be patient).....done

Server Software:      nginx/1.12.2
Server Hostname:     myencompass.supsi.ch
Server Port:         80

Document Path:       /community/UserActivityCreditWebServiceREST/GetAction/getAction?id=10
Document Length:    185 bytes

Concurrency Level:   10
Time taken for tests: 2.467 seconds
Complete requests:   100
Failed requests:     0
Non-2xx responses:   100
Total transferred:   45200 bytes
HTML transferred:    10500 bytes
Requests per second: 40.54 [#/sec] (mean)
Time per request:    246.651 [ms] (mean)
Time per request:    24.665 [ms] (mean, across all concurrent requests)
Transfer rate:       17.90 [Kbytes/sec] received

Connection Times (ms)
  min  mean[+/-sdl] median  max
Connect:    18   24   8.9   21   76
Processing:  38  208  45.7  199  316
Waiting:    19  150  57.5  158  278
Total:      66  232  45.3  222  344

Percentage of the requests served within a certain time (ms)
 50%    222
 66%    235
 75%    244
 80%    267
 90%    300
 95%    319
 98%    342
 99%    344
100%    344 (longest request)
```

Service assingActionsToUsers:

```
This is ApacheBench, Version 2.3 <$Revision: 1843412 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking myencompass.supsi.ch (be patient).....done

Server Software:      nginx/1.12.2
Server Hostname:     myencompass.supsi.ch
Server Port:         80

Document Path:       /community/UserActivityCreditWebServiceREST/AssignActionsToUsers/assignActionsToUsers
Document Length:    185 bytes

Concurrency Level:   10
Time taken for tests: 1.632 seconds
Complete requests:   100
Failed requests:     0
Non-2xx responses:   100
Total transferred:   46800 bytes
Total body sent:     33500 bytes
HTML transferred:    18500 bytes
Requests per second: 61.27 [#/sec] (mean)
Time per request:    163.203 [ms] (mean)
Time per request:    16.320 [ms] (mean, across all concurrent requests)
Transfer rate:       28.00 [Kbytes/sec] received
                   20.05 kb/s sent
                   48.05 kb/s total

Connection Times (ms)
  min  mean[+/-sdl] median  max
Connect:    15   16   0.5   16   17
Processing:  17  137  25.1  143  159
Waiting:    16   96  37.5   96  149
Total:      33  153  25.0  158  176

Percentage of the requests served within a certain time (ms)
 50%    158
 66%    159
 75%    159
 80%    161
 90%    164
 95%    165
 98%    165
 99%    176
100%    176 (longest request)
```

6.2.2 Awareness Application Functional Tests

Examples of Functional tests executed on AA Application are described step by step and the output of each step is shown.

Test Procedure

TEST 1

Test 1	User logs in to the application
Description	The user types its credentials, and login to the app that will display the home page.
Pre-Conditions	User has received the username and password and has the app installed in the phone.
Actions	<ol style="list-style-type: none"> 1. The user types its credential in the corresponding fields and click the sign in button 2. The application validates the credentials, <ul style="list-style-type: none"> • if they are correct, the session is created, and the application display the home page • if they are incorrect, the application notifies the user that there is a problem with the inserted credentials and remains in the login page.
Expected Result	The user is able to login to the application by providing the correct credential and the home page is displayed.
Actual Result	The user is able to login to the application by providing the correct credential and the home page is displayed.
Test Result	Passed

Test 1 output

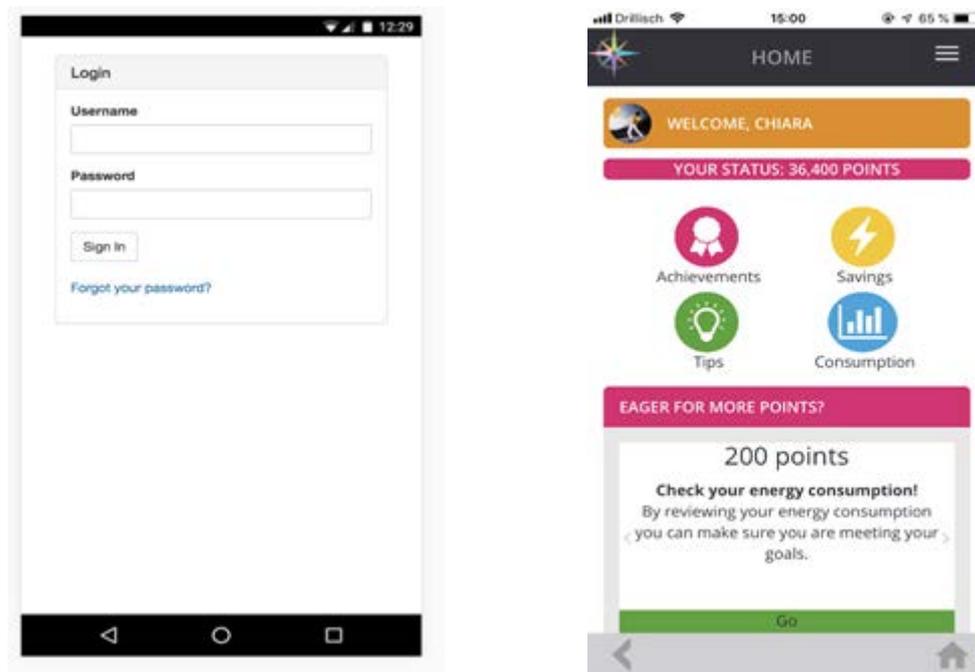


Figure 8 – On the left, the login page of the encompass application; on the right, the home page of the encompass application is display after successful login authentication.

TEST 2

Test 2	User access the saving section
Description	The user opens the Menu and selects the “Savings” section to see the current status.
Pre-Conditions	User has already login to the application and its consumption data are stored.
Actions	<ol style="list-style-type: none"> 1. The user opens the Menu 2. Select “Savings” menu and see the Goal section: <ol style="list-style-type: none"> a. The user is able to see its current consumption status, and change its consumption
Expected Result	The user is able to see the current consumption status
Actual Result	The user is able to see the current consumption status
Test Result	Passed

TEST 2 Output

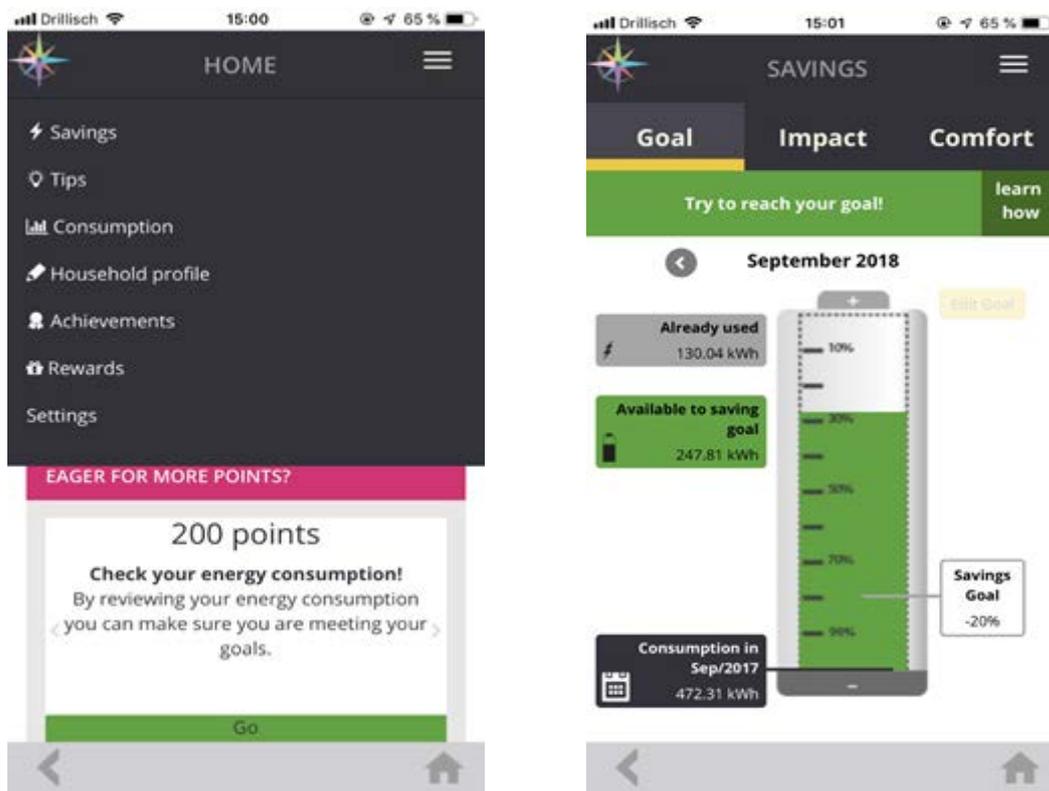


Figure 9 - On the left, the encompass menu; on the right, the impact section displaying the Goal subsection where users can see the consumption status.

TEST 3

Test 3	Visualize Inferred Comfort and provide feedback
Description	The user opens the Menu and selects the “Savings” section, and goes Comfort subsection so the inferred comfort level and provide feedback.
Pre-Conditions	User has already login to the app and its comfort levels have been estimated by the Inference Engine.
Actions	<ol style="list-style-type: none"> 1. The user opens the Menu 2. Selects the “Savings” menu, on the Savings section the user selects the “Comfort” tab. 3. The application displays the savings percentage, the average temperature and

	<p>the humidity, and the inferred comfort level.</p> <ol style="list-style-type: none"> User can provide feedback about the comfort level by clicking on the “+” button, and selecting one of the 7 comfort levels available. Then clicks “send” button to save the feedback.
Expected Result	The user is able to see the average temperature, humidity and the inferred comfort level. The user is able to provide feedback about the comfort level.
Actual Result	The user is able to see the average temperature, humidity and the inferred comfort level. The user is able to provide feedback about the comfort level.
Test Result	Passed

TEST 3 Output

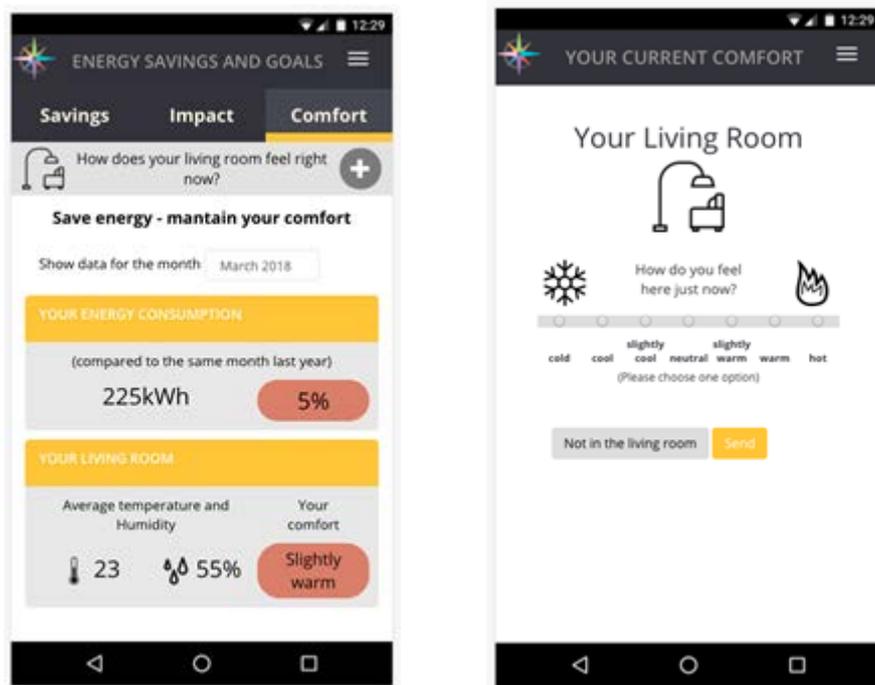


Figure 10 – On the left, the Comfort section displaying the savings percentage, the average temperature, the average humidity, and the inferred comfort level. On the right, the comfort feedback section.

TEST 4

Test 4	Read tips
Description	The user opens the Menu and selects “Tips” section to see the tips and personal recommendations.
Pre-Conditions	User has already login to the app and tips have been assign to his profile.
Actions	<ol style="list-style-type: none"> The user opens the Menu Selects “Tips” menu and see a list of tips and recommendation to save energy User can click on the next arrow or dot menu to see a different tip. User can provide feedback for each tip, by clicking on the available feedback options. The user reads a recommendation (by spending 5 seconds on the tips) and the application assigns 200 points for this action. By providing feedback , the user get 400 points.
Expected Result	The user is able to see the list of tips and recommendations and provides feedback to a tip, the application assigns points for reading the tip and providing feedback
Actual Result	The user is able to see the list of tips and recommendations and provides feedback to a

	tip, the application assigns points for reading the tip and providing feedback
Test Result	Passed

TEST 4 Output

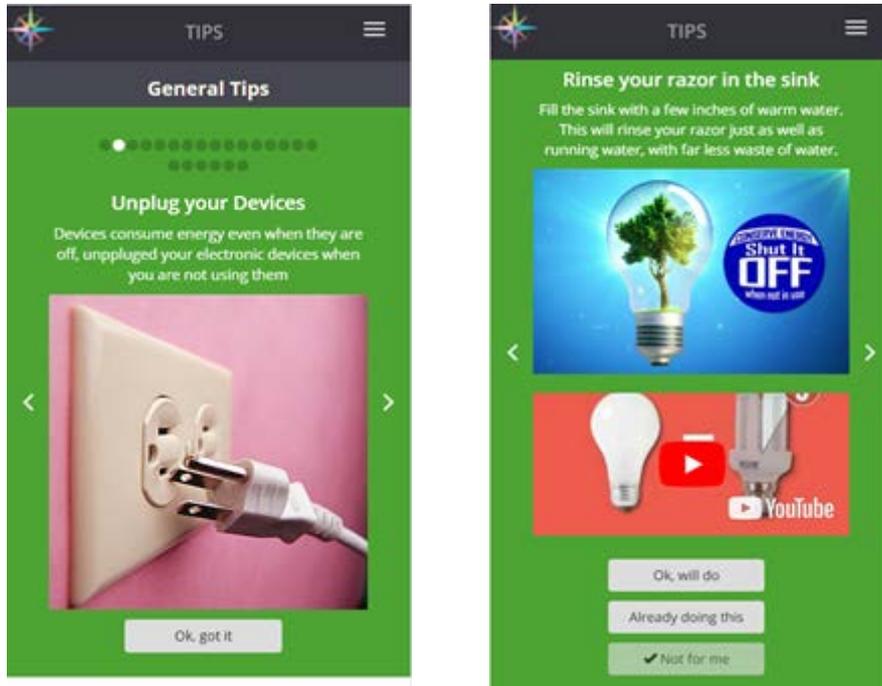


Figure 11 – On The left, the Tip section showing a tip, the previous and next buttons, and the navigation list. On the right, a tip showing the tip text, an image, a video, and the feedback options.

TEST 5

Test 5	View the detailed consumption
Description	The user opens the Menu and selects the “Consumption” section to see a detailed consumption information.
Pre-Conditions	User has already login to the application and its consumption data have been store on the db.
Actions	<ol style="list-style-type: none"> 1. The user opens the Menu 2. Select the “Consumption” menu 3. The application display a bar graph with the daily consumption detail. 4. The user can navigate the consumption using the bar at the bottom of the graph. 5. The user can change the granularity (daily, weekly monthly) by selecting an option from the dropdown menu.
Expected Result	The user is able to see and explore his energy consumption at different levels of granularity
Actual Result	The user is able to see and explore his energy consumption at different levels of granularity
Test Result	Passed

TEST 5 Output

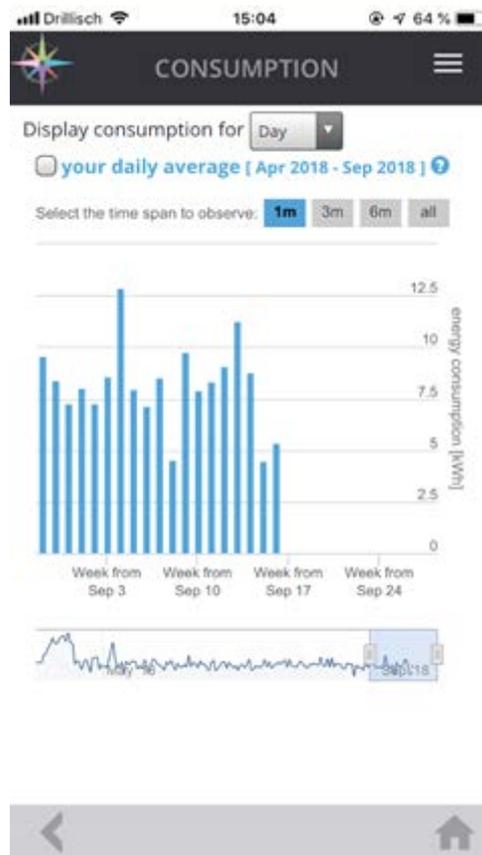


Figure 12 – The detail consumption section displays the daily consumption information using a bar graph, the user can explore it consumption by interacting with the graph controls.

6.3 DISAGGREGATION ENGINE

6.3.1 Disaggregation Engine Performance Tests

Internal tests have been executed in order to assess functionality, reliability, security and performances of the modules, but since the DE processing is not blocking for the orchestration chain, no performance test results are available for this component.

6.3.2 Disaggregation Engine Functional Tests

The following procedure describes the steps to set up the testing environment for the enCompass backend services, instantiated on each one of three pilot servers.

This Functional Test Plan consists of Disaggregation Engine tests. For these tasks, the tester will have to access the database and the backend services' APIs of each pilot server.

The backend services' APIs are exposed by the following Swagger interface:

<http://{server-ip}:8081/swagger-ui.html>

Examples of Functional tests executed are described in the following section:

Processing

Test	Processing
Description	The user accesses to the DB and acquires the data
Pre-Conditions	The tester will have to access the database and the backend services' APIs of each pilot server
Actions	Run this query in order to get the list of the daily disaggregated data: <pre>SELECT encompass_model_ses.disaggregation_data.* FROM encompass_model_ses.disaggregation_data WHERE datetime_received IN (select max(datetime_received) as maxdt from encompass_model_ses.disaggregation_data where date >= DATE_SUB(now(), INTERVAL 3 DAY) AND date <= DATE_SUB(now(), INTERVAL 2 DAY) GROUP BY user_oid, date) AND date >= DATE_SUB(now(), INTERVAL 3 DAY) AND date <= DATE_SUB(now(), INTERVAL 2 DAY) order by date desc</pre>
Expected Result	The user is able to see the entries
Actual Result	The user is able to see the entries
Test Result	Passed

User Service

Test	User Service
Description	The user accesses to the DB and verify that data is present for the chosen time period
Pre-Conditions	The tester will have to access the database and the backend services' APIs of each pilot server
Actions	<ol style="list-style-type: none"> Extract in an Excel sheet the output of this query: <pre>SELECT disaggregation_data.* FROM encompass_model_ses.disaggregation_data WHERE datetime_received IN (select max(datetime_received) as maxdt from encompass_model_ses.disaggregation_data where user_oid = <user_oid> and date >= DATE_SUB(now(), INTERVAL 33 DAY) AND date <= DATE_SUB(now(), INTERVAL 3 DAY) GROUP BY date) AND date >= DATE_SUB(now(), INTERVAL 33 DAY) AND date <= DATE_SUB(now(), INTERVAL 3 DAY) AND user_oid = <user_oid> ORDER BY date DESC</pre> <p><u>Best case</u>: the query returns 30 entries, this means that the DE has correctly generated all data in the last 30 days. <u>Worst case</u>: 0 results, no data generated by the DE. PASS: at least 10 entries are present (the DE service needs at least 10 days of disaggregated data in order to provide useful information to the user)</p> Calculate the sum of the columns: fridge, washing_machine, tumble_dryer, dishwasher, AC, electric_car, electric_oven, heat_pump, other, total_consumption

	3. Compare the results against the results provided by the following service exposed by Swagger: GET /de/getDisaggregatedDataMean (only set the user_oid) PASS: the results match
Expected Result	All steps results are passed
Actual Result	All steps results are passed
Test Result	Passed

6.4 NOTIFICATION ENGINE

6.4.1 Notification Engine Performance Tests

Post message instance service:

```
[ec2-user@ip-10-3-3-138 ~]$ ab -n 1 http://
This is ApacheBench, Version 2.3 <Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking                               (be patient).....done

Server Software:      Apache/2.2.21
Server Hostname:
Server Port:         80

Document Path:
Document Length:     8572 bytes

Concurrency Level:    1
Time taken for tests: 0.029 seconds
Complete requests:   1
Failed requests:     0
Write errors:        0
Total transferred:   8995 bytes
HTML transferred:    8572 bytes
Requests per second: 34.05 [#/sec] (mean)
Time per request:    29.368 [ms] (mean)
Time per request:    29.368 [ms] (mean, across all concurrent requests)
Transfer rate:       299.11 [Kbytes/sec] received

Connection Times (ms)
      min  mean[+/-sd] median  max
Connect:    3    3  0.0    3    3
Processing: 26   26  0.0   26   26
Waiting:    16   16  0.0   16   16
Total:      29   29  0.0   29   29
```

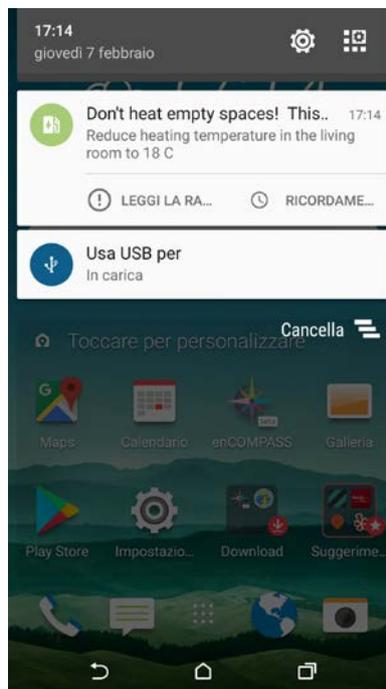
6.4.2 Notification Engine Functional Tests

Examples of Functional tests executed are described in the following:

Test	Notification Engine
Description	The user can see a notification for incentive recommendation is sent, as well as how it looks on the app
Pre-Conditions	User has received the username and password and has the app installed in the phone.
Actions	1. Login to the AA APP

	<ol style="list-style-type: none"> 2. Enable notifications with high frequency in the App. 3. Ensure the app is allowed to receive notifications from Android. 4. Create a notification following this: <ol style="list-style-type: none"> a. Create a notification and post it to one of the message queues Open SWAGGER-UI into a browser and find postMessageInstance. Call the service using the following JSON: <pre> { "generic_message_oid": 271, // Example of Message OID "hidden": true, "is_static": false, "userId": 1 } </pre> <p>If you send is_static as true the motivation will be appended to the recommendation's title, otherwise it won't. It may worth executing this message 3 times, to enqueue 3 notifications.</p> b. Check the content of the queues c. Force processing the notifications in the queues d. Notification output is shown on your phone
Expected Result	The Notification is received on phone.
Actual Result	The Notification is received on phone.
Test Result	Passed

An example of how notifications look on the phone is shown in follow screenshot:



6.5 RECOMMENDATION ENGINE

The Recommendation Engine was tested using test data from local database. Manual tests aided by several tools were performed to validate the functionality of the component.

6.5.1 Recommendation Engine Functional Tests

Preconditions

The following preconditions must be met/done before the testing session by relevant partners.

- Clean enCOMPASS platform installation (no other data is in the relevant tables, but the ones provided as “Data Set”).
- The test encompass platform installation is configured to connect to the test Recommendation Engine.
- The test Recommendation Engine is empty, no old data, or calculated data exists in its database.
- To make the Recommendation Engine algorithm deterministic so it will not be dependent on the time when test will be run, we fixed the date, and the random number generator in the test deployment (this is the only difference between the test and production version).

Data Set

The input starting database state will be provided in separate document as sql file (this file will contain the db reset mechanisms, and the data needed in the first step of the Test Procedure)

Expected Results

Because the data set that was generated for testing purposes is the same for all use cases, each use case can be run individually from start-to-end, or the whole process once, and only the results can be evaluated for all the use cases.

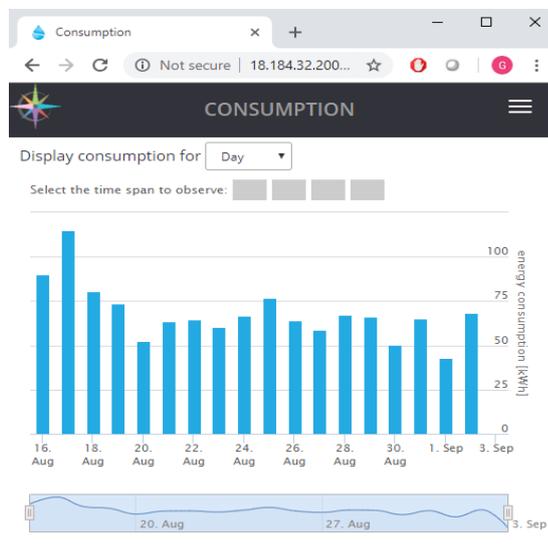
We chose the following use cases for the test.

Case	Case 1	Case 2	Case 3	Case 4	Case 5
Simulated Condition	The peak consumption (consumption presumably related to activity) is high and the household has a TV set	Humidity is above 50% for at least 2 hours	Base consumption (consumption presumably not related to an activity) is high and the household has a freezer	During the day the luminance is above 60 for at least an hour when nobody is at home	Testing fixed executable recommendation
User id (username)	3 (test3)	4 (test4)	5 (test5)	96 (test8)	1 (test1)
Expected Recommendation in AA's Just for You tab and Notifications	title: “Not there? Then switch it off”, description: “Is the TV running even though you are doing something different? Turn it off!” (id:106)	title: “Let the wind blow”, description: “Shock ventilate more often.” (id: 81)	title: “Fight the ice!”, description: “Defrost freezer (e.g. before vacation) to melt ice layer.” (id: 6)	title: “Let's enjoy a bit of shade”, description: “To keep your living room cool” (id: 77)	title: “Don't heat empty spaces!” description: “Reduce heating temperature in the living room to 18 C” (id: 271)
Expected Time of the Recommendation Notification	2018-08-30 9:00	2018-08-30 9:00	2018-08-30 9:00	2018-08-30 8:00	2018-08-30 at 8:00
User Motivation	3 (This will help you to collect more energy)	1 (This will help you to save money)	2 (This will help to protect the environment)	2 (This will help to protect the environment)	1 (This will help you to save money)

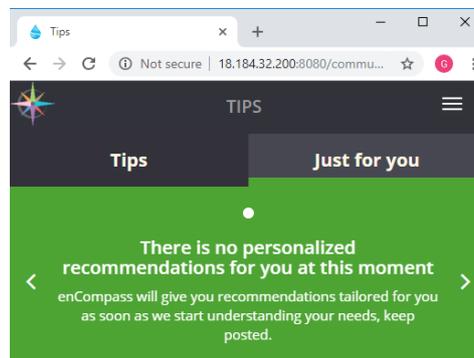
As it is visible, we tested different test users with different background conditions (ie. dataset) and we also tested whether the so called semi-executable recommendations can be applied.

Test Procedure

1. Data Setup (set the user/smart_meter/etc data to a fixed state provided by us)
 - a) Data set should be imported by resetting the db state to the default starting state by running the provided sql scripts (or the all-in-one version).
2. Check on AA that users are there with all the data, but they have no recommendations at the moment. (for simplicity sake we use the web version of the AA available on the relative url: /community)
 - a) User data is loaded into the encompass platform:
(Sensor data cannot be shown on the AA)



- a) User tips should be empty for each user on the AA:



3. (Run the export for old data - as the user pre-set data is from 2018 august)
 - a) Call the API which exports all previous old data from enCompass platform to RE backend. The relative URL for starting the old export data is the same as starting the orchestration step:

/re/component/start

But you have to also send the following JSON HTTP POST payload:

```
{
    "exportDayCount": 300
}
```

(the exportDayCount parameters should be set to include 2018 august)

4. Manually start the orchestration on enCompass platform, which will run all the previous components leading up to Recommendation Engine. (Or just manually start the orchestration step for RE, as the functionality of the Orchestration as a whole is not the scope of this test)

The relative URL that starts the orchestration step for RE is: /re/component/start
Which must be called as HTTP POST.

5. Recommendation Engine (automatic)
 - a) Recommendation engine export, and calculation will run as part of the Component Orchestration. (automatic)
 - b) You can check semaphore_log table in the main encompass platform main database to see status codes and check its status code.
 - c) You can also check the SFTP access provided by GRA for the deployed instance, and check import status of the export:
 1. checking the filename for the date
 2. and extension for status:
 - .go: not yet processed
 - .processing: under process
 - .finished: the data is imported to the RE
 - .failed: there were some problems during the import
6. Check results in EnCompass Platform
 - a) After Recommendation Engine process is finished you can check results the results for the different Use Cases in encompass Platform (in DB, or in AA). (See Test Results part of this document)
 - b) Notifications are in the past, so it's expected result can only be checked in DB
This is available in the encompass main database's notification_instance table

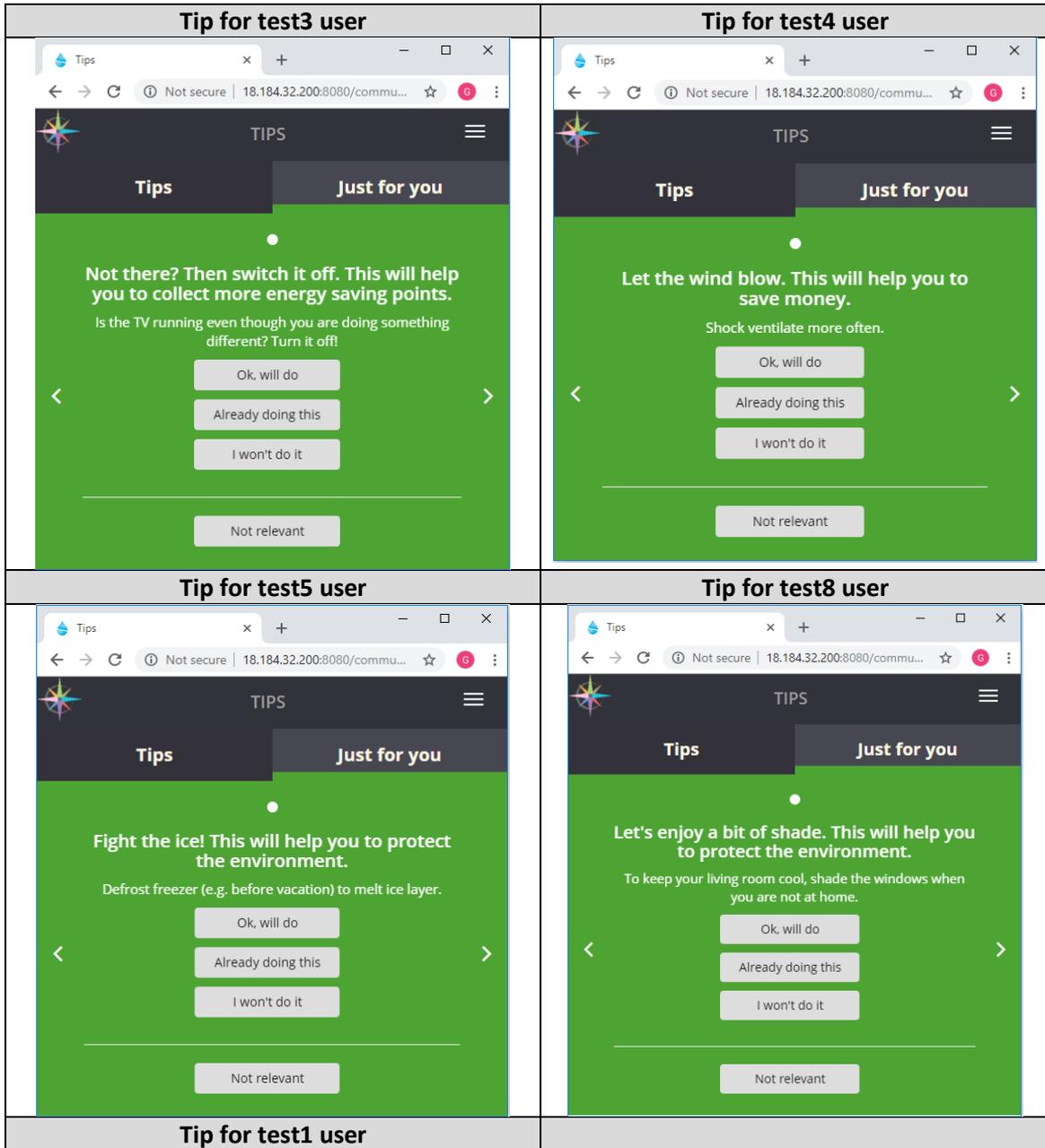
Use Cases

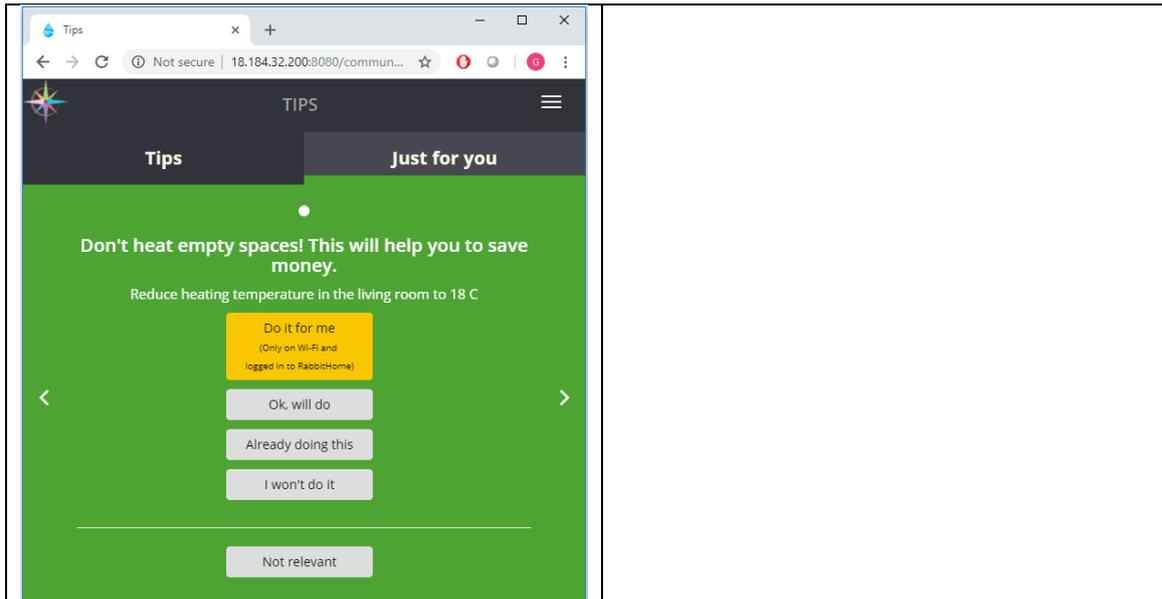
Case	Case 1	Case 2	Case 3	Case 4	Case 5
Simulated Condition	The peak consumption (consumption presumably related to activity) is high and the household has a TV set	Humidity is above 50% for at least 2 hours	Base consumption (consumption presumably not related to an activity) is high and the household has a freezer	During the day the luminance is above 60 for at least an hour when nobody is at home	Testing fixed executable recommendation
User id (username)	3 (test3)	4 (test4)	5 (test5)	96 (test8)	1 (test1)
Expected Recommendation in AA's Just for You tab and Notifications	title: "Not there? Then switch it off", description: "Is the TV running even though you are doing something different? Turn it off!" (id:106)	title: "Let the wind blow", description: "Shock ventilate more often." (id: 81)	title: "Fight the ice!", description: "Defrost freezer (e.g. before vacation) to melt ice layer." (id: 6)	title: "Let's enjoy a bit of shade", description: "To keep your living room cool" (id: 77)	title: "Don't heat empty spaces!" description: "Reduce heating temperature in the living room to 18 C" (id: 271)
Expected Time of the Recommendation	2018-08-30 9:00	2018-08-30 9:00	2018-08-30 9:00	2018-08-30 8:00	2018-08-30 at 8:00

Notification					
User Motivation	x	x	x	x	x

Test Result

Results of the test was exactly what we were expected. This means that we were able to run the test without any problems and in the AA the following messages appeared, for the given test users:





You can clearly see that all the required parts related to the recommendations (i.e. motivational incentive, feedback buttons and the Semi-automated “Do it for me button in test1 user’s account appeared) in the AA.

Test of the RE component was successfully done. The test was made and documented with PDX.

6.6 INFERENCE ENGINE

6.6.1 Inference Engine Performance Tests

Thermal comfort inference service:

```

===== CPU information =====
Platform information : AMD64
Platform version and information : 10.0.18362 Windows-10-10.0.18362-SP0
Processor : Intel64 Family 6 Model 94 Stepping 3, GenuineIntel

Running tests for Thermal comfort Inference
Integration testing: ... System started at 16-10-2016 and finished succesfully.
System testing: ... Connected to DB. System tested succesfully.
Effectiveness: ... 100%
System testing: ... Successfull
Operational acceptance testing: ... Successfull
Completion rate: ... Run for 1 user estimated comfort for 1 user, rate= 1
Operational acceptance testing: ... Component ready.Run succesfully

Connected to WVT Database
Estimating thermal comfort for dwelling 106...
Estimation of thermal comfort finished, execution time 1.100 sec

```

Visual comfort inference service:

```
===== CPU information =====
Platform information : AMD64
Platform version and information : 10.0.18362 Windows-10-10.0.18362-SP0
Processor : Intel64 Family 6 Model 94 Stepping 3, GenuineIntel

Running tests for visual comfort Inference
Integration testing: ... System started at 16-10-2016 and finished succesfully.
System testing: ... Connected to DB. System tested succesfully.
Effectiveness: ... 100%
System testing: ... Successfull
Operational acceptance testing: ... Successfull
Completion rate: ... Run for 1 user estimated comfort for 1 user, rate= 1
Operational acceptance testing: ... Component ready.Run succesfully

Connected to SES Database
Estimating visual comfort for dwelling 88...
Estimation of visual comfort finished, execution time 0.700 sec
```

Occupancy inference service:

```
===== CPU information =====
Platform information : AMD64
Platform version and information : 10.0.18362 Windows-10-10.0.18362-SP0
Processor : Intel64 Family 6 Model 94 Stepping 3, GenuineIntel

Running tests for occupancy Inference
Integration testing: ... System started at 16-10-2016 and finished succesfully.
System testing: ... Connected to DB. System tested succesfully.
Effectiveness: ... 100%
System testing: ... Successfull
Operational acceptance testing: ... Successfull
Completion rate: ... Run for 1 user estimated occupancy for 1 user, rate= 1
Operational acceptance testing: ... Component ready.Run succesfully

Connected to WVT Database
Estimating occupancy for dwelling room 116...
Occupancy inference finished, execution time 0.300 sec
```

Activity inference service:

```
===== CPU information =====
Platform information : AMD64
Platform version and information : 10.0.18362 Windows-10-10.0.18362-SP0
Processor : Intel64 Family 6 Model 94 Stepping 3, GenuineIntel

Running tests for activity Inference
Integration testing: ... System started at 16-10-2016 and finished succesfully.
System testing: ... Connected to DB. System tested succesfully.
Effectiveness: ... 100%
System testing: ... Successfull
Operational acceptance testing: ... Successfull
Completion rate: ... Run for 1 user estimated activity for 1 user, rate= 1
Operational acceptance testing: ... Component ready.Run succesfully

Connected to SES Database
Estimating activity for dwelling room 100...
Activity inference finished, execution time 0.500 sec
```

6.7 FUNERGY

6.7.1 Funergy Performance Tests

Service getLocalizedQuestion:

```
This is ApacheBench, Version 2.3 <$Revision: 1843412 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking funergy.ifmledit.org (be patient)....done

Server Software:
Server Hostname:      funergy.ifmledit.org
Server Port:         80

Document Path:       /funergy/Services/ffv/getLocalizedQuestion?language=en&level=1
Document Length:     42 bytes

Concurrency Level:   10
Time taken for tests: 0.791 seconds
Complete requests:   100
Failed requests:     0
Non-2xx responses:   100
Total transferred:   30400 bytes
HTML transferred:    4200 bytes
Requests per second: 126.49 [#/sec] (mean)
Time per request:    79.059 [ms] (mean)
Time per request:    7.906 [ms] (mean, across all concurrent requests)
Transfer rate:       37.55 [Kbytes/sec] received

Connection Times (ms)
  min   mean[+/-sd] median   max
Connect:    3     6   3.3      5     31
Processing: 14    68  21.3     67    126
Waiting:    14    63  22.1     63    125
Total:      21    74  21.7     74    130

Percentage of the requests served within a certain time (ms)
 50%    74
 66%    83
 75%    88
 80%    91
 90%   106
 95%   115
 98%   117
 99%   130
100%   130 (longest request)
```

Service getNextQuestion:

```
This is ApacheBench, Version 2.3 <$Revision: 1843412 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking funergy.ifmledit.org (be patient)....done

Server Software:
Server Hostname:      funergy.ifmledit.org
Server Port:         80

Document Path:       /funergy/Services/ffv/getNextQuestion2?language=en&level=1&oid=1
Document Length:     122 bytes

Concurrency Level:   10
Time taken for tests: 1.140 seconds
Complete requests:   100
Failed requests:     0
Total transferred:   34900 bytes
HTML transferred:    12200 bytes
Requests per second: 87.72 [#/sec] (mean)
Time per request:    113.998 [ms] (mean)
Time per request:    11.400 [ms] (mean, across all concurrent requests)
Transfer rate:       29.90 [Kbytes/sec] received

Connection Times (ms)
  min   mean[+/-sd] median   max
Connect:    2     6   6.5      5     67
Processing: 36    98  29.7     98    181
Waiting:    33    86  29.0     86    171
Total:      43   104  30.0    106    185

Percentage of the requests served within a certain time (ms)
 50%   106
 66%   115
 75%   127
 80%   132
 90%   144
 95%   149
 98%   177
 99%   185
100%   185 (longest request)
```

Service getNextQuestionForTag:

```
This is ApacheBench, Version 2.3 <$Revision: 1843412 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking funergy.ifmledit.org (be patient).....done

Server Software:
Server Hostname:      funergy.ifmledit.org
Server Port:          80

Document Path:        /funergy/Services/ffv/getNextQuestionForTag?language=en&level=1&oid=1&tag=television
Document Length:      122 bytes

Concurrency Level:    10
Time taken for tests:  1.881 seconds
Complete requests:    100
Failed requests:      0
Total transferred:    34900 bytes
HTML transferred:     12200 bytes
Requests per second:  53.17 [#/sec] (mean)
Time per request:     180.862 [ms] (mean)
Time per request:     18.806 [ms] (mean, across all concurrent requests)
Transfer rate:        18.12 [Kbytes/sec] received

Connection Times (ms)
  min   mean[+/-sd] median   max
Connect:    3    6   3.4    5    25
Processing: 76   172  29.6   177   251
Waiting:    23   162  31.6   165   239
Total:      89   179  30.0   184   258

Percentage of the requests served within a certain time (ms)
 50%    184
 66%    190
 75%    197
 80%    204
 90%    209
 95%    229
 98%    247
 99%    258
100%    258 (longest request)
```

Service getAvailableTags:

```
This is ApacheBench, Version 2.3 <$Revision: 1843412 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking funergy.ifmledit.org (be patient).....done

Server Software:
Server Hostname:      funergy.ifmledit.org
Server Port:          80

Document Path:        /funergy/Services/ffv/getAvailableTags
Document Length:      2430 bytes

Concurrency Level:    10
Time taken for tests:  1.103 seconds
Complete requests:    100
Failed requests:      0
Total transferred:    268700 bytes
HTML transferred:     243000 bytes
Requests per second:  90.65 [#/sec] (mean)
Time per request:     110.310 [ms] (mean)
Time per request:     11.031 [ms] (mean, across all concurrent requests)
Transfer rate:        237.88 [Kbytes/sec] received

Connection Times (ms)
  min   mean[+/-sd] median   max
Connect:    3    8   6.5    6    60
Processing: 43   92  37.2   78   224
Waiting:    30   83  32.4   73   191
Total:      52  100  36.8   84   233

Percentage of the requests served within a certain time (ms)
 50%    84
 66%   109
 75%   123
 80%   127
 90%   153
 95%   177
 98%   205
 99%   233
100%   233 (longest request)
```

6.7.2 Funergy Functional Tests

Examples of Functional tests executed on Funergy Application are described step by step and the output of each step is shown.

Test Procedure

TEST 1

Test 1	User Plays Single Player mode
Description	The user starts the Funergy digital game extension to play on single mode.
Pre-Conditions	The user has installed the funergy app on a mobile device.
Actions	<ol style="list-style-type: none"> 1. The user starts the Funergy app. 2. On the home menu of the app he selects the single player mode. 3. The app presents a question corresponding to the current user's level. 4. The user selects one of the possible answers. 5. The application provides feedback whether the answer was correct or wrong and presents the user with the options of reading the explanation to the question or continue playing. 6. If the user selects the explanation option, the application shows a brief explanation about the topic of the question and show the option to continue playing. 7. If the user selects the continue option, the application will display a new question, and the cycle will continue.
Expected Result	The user is able to play on single mode by receiving a continues flow of energy related questions.
Actual Result	The user is able to play on single mode by receiving a continues flow of energy related questions.
Test Result	Passed

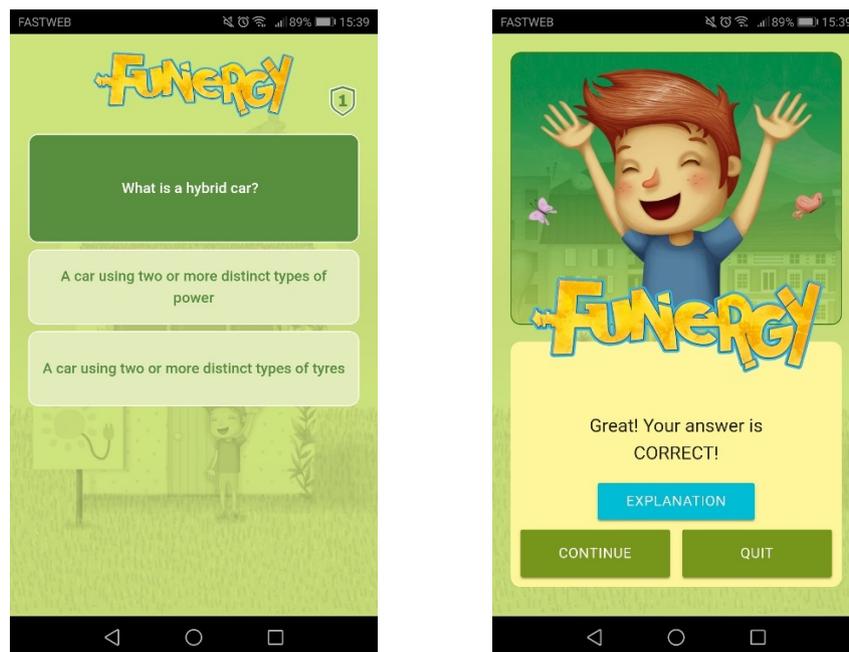


Figure 13 – On the left, the Funergy app displaying a Question with 2 possible answers. On the right, the app display positive feedback after a correct answer, options to see the explanation and to continue playing are available.

TEST 2

Test 2	User Decodes a Card
Description	While playing the Funergy board game, the user needs to scan a card and answer a question to complete the game round.
Pre-Conditions	The user has installed the funergy app on a mobile device.
Actions	<ol style="list-style-type: none"> 1. The starts the Funergy app. 2. On the home menu he selects the decode a card mode. 3. The app starts the camera to scan for a QR Code. 4. The user places the QR code in front of the camera. 5. The application recognizes the QR code and presents a question corresponding to the current user's level. 6. The user selects one of the possible answers. 7. The application provides feedback whether the answer was correct or wrong and presents the user with the option of reading the explanation to the question. 8. If the user selects the explanation option, the application shows a brief explanation about the topic of the question and show the option to go back to the home menu.
Expected Result	The user is able to play the game by decoding a card.
Actual Result	The user is able to play the game by decoding a card.
Test Result	Passed



Figure 14 – On the left, the home menu of the funergy app showing the Decode a card and single player options. On the right, the decode a card option opens the camera and enable the QR code decoding.

7 REFERENCES

- JIndent - <http://www.newforms-tech.com/products/jindent/about>
- YourKit - <https://www.yourkit.com>
- WebRatio - <http://www.webratio.com/>
- IFML standard - <http://www.ifml.org/>
- IFMLEdit - <https://ifmledit.org/>
- Pylint - <https://www.pylint.org/>
- Eclipse Foundation - <https://www.eclipse.org/ide/>
- JetBrains - <https://www.jetbrains.com/pycharm/>
- Apache web Server - https://encompass.idisia.ch/webscript/cgi-bin/disaggregation_engine
- Postman Inc. - <https://www.getpostman.com/>
- Apache Commons - <https://commons.apache.org/proper/commons-daemon/jdepend-report.html>
- Pylint - https://pylint.readthedocs.io/en/latest/technical_reference/features.html