Technical Test Report 1

SEPTEMBER 2015

DELIVERABLE

Project Acronym:	**SDI4Apps**
Grant Agreement number:	**621129**
Project Full Title:	**Uptake of Open Geographic Information Through Innovative Services Based on Linked Data**

D3.6.1 TECHNICAL TEST REPORT 1

Revision no. 05

Authors: Daniele Tarini (Hyperborea Srl)
All Technical Partners

Project co-funded by the European Commission within the ICT Policy Support Programme		
Dissemination Level		
P	Public	X
C	Confidential, only for members of the consortium and the Commission Services	

REVISION HISTORY

Revision	Date	Author	Organisation	Description
01	29/09/2015	Daniele Tarini	Hyper	Initial draft
02	29/09/2015	Karel Charvat	HSRS	Provision of comments for final submission
03	29/09/2015	Martin Kuba	MU	Provision of comments for final submission
04	29/09/2015	Runar Bergheim	Avinet	Provision of comments for final submission
05	30/09/2015	Daniele Tarini	Hyper	Consolidated final version prepared according to gathered comments and feedback from Peer Reviewers

TABLE OF CONTENTS

LIST OF ACRONYMS

SaaS	Software as a Service
SDI	Spatial Data Infrastructure
SME	Small and Medium Enterprises
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
IoT	Internet of the Things

EXECUTIVE SUMMARY

The SDI4Apps platform consists of two service levels: basic and advanced. The testing of the services in both levels is guided by a common technical test methodology which was defined in D3.5 “Technical Test Methodology” report. This document D.3.6.1 “Technical Test Report 1” focuses on the testing of basic services, while the advanced services will be taken care of in another document, namely D.3.6.2 " Technical Test Report 2"., In the light of the above, this document has the following two main goals.

- The first goal is to refer, recap, expand and complete the technical test methodology, providing the definition and all the details for testing the platform basic services (therefore to finalised the former D3.5 "Technical Test Methodology" as better explained in the next section "Introduction").
- The second goal is to report the results of the first batch of tests run on the basic services offered by the platform.

1 INTRODUCTION

According to the DoW, this document aims at providing "an overview of the tests performed and their results from the first phase of the project implementation", i.e. the first batch of technical tests run on the project platform. As such, it be mainly focused on testing the platform services and provide the appropriate reports on the outcomes.

However, according to the comments provided by the Reviewers after the review technical meeting held in Luxembourg the 6th of May 2015, the current report also includes actions undertaken by project partners to complete the overall technical test methodology (D3.5) which has been already submitted at M5.

, In particular, by the time of the definition of the test methodology, and in the writing of the Technical Test Methodology document (project deliverable D3.5, as already mentioned above), many of the decisions about the implementation of the services were in progress. In fact, the technology of choice was still to be chosen among a few good candidates, and even more so specific details about the implementation of the services were still to be defined. As a result, the testing methodology provided in the document D3.5 defined a high level, solid and widely accepted, approach for testing to be applied to the project's platform services, and provided common guidelines on tools to be used for carrying out the tests.

After D3.5 submission at M5, further tasks were accomplished and the Dow goals to define the technology to support the platform services have been achieved. It is worth reminding that, according to the work plan defined in DoW, the platform open API is undergoing detailed definition, especially for the basic services, therefore, related tests and final results will be duly carried out and documented in the next report D3.6.2, whose deadline is M36.

, In the light of all the above assumptions and clarifications, the **first goal** of this document is to refer, recap, expand and complete the contents Technical Test Methodology (as it was in the Consortium's intentions and as agreed during the First Technical Review Meeting).With the details of the Technical Test Methodology shaped out, the **second goal** of the document is to effectively provide the results of the execution of the first batch of tests on the basic services. This is detailed in the second part of this document.

The final part of this document resumes the conclusions of all the contents described above.

2 TECHNICAL TEST METHODOLOGY

2.1 Introduction

D3.5 gave the guidelines about the test methodology as it was meant to be applied to the platform basic services. At the time of its writing the services were still to be defined in detail and the methodology itself was elaborated accordingly. Later on, D4.4 further contributed to refine the methodology and provided a solid base on which the platform advanced services could be tested. Below, main technical test methodology aspects (derived by former work) are reported:

1. identification of functional and non-functional requirements to test
2. definition of the test matrix, i.e. a list of tests addressing the requirements, where each test states:
    a. the success condition addressing one of the requirements
    b. the indicator(s) that can be measured to verify the condition
    c. the metric of the indicator
    d. the thresholds of success to be used for each metric

About the first main point, the identification of functional and non functional requirements, this document focuses mainly non functional ones, since the basic services are provided by the platform exploiting well known technologies. As D3.3.1 "Open API Design & First Release" states in the description of the basic services:

> *These services are all based on well-tested, well-proven protocols and standards. The emphasis is therefore not on the technical sophistication of the services but rather the performance, scalability and robustness under heavy load.*

As a consequence, the non-functional requirement in the spotlight, in this document, is performance in the provision of basic services in a scalable architecture, and robustness under heavy load.

About the second point, this document will provide the list of tests concerning the validation of the requirements of each service by including all main related aspects, e.g. conditions, indicators, metrics, thresholds, etc.

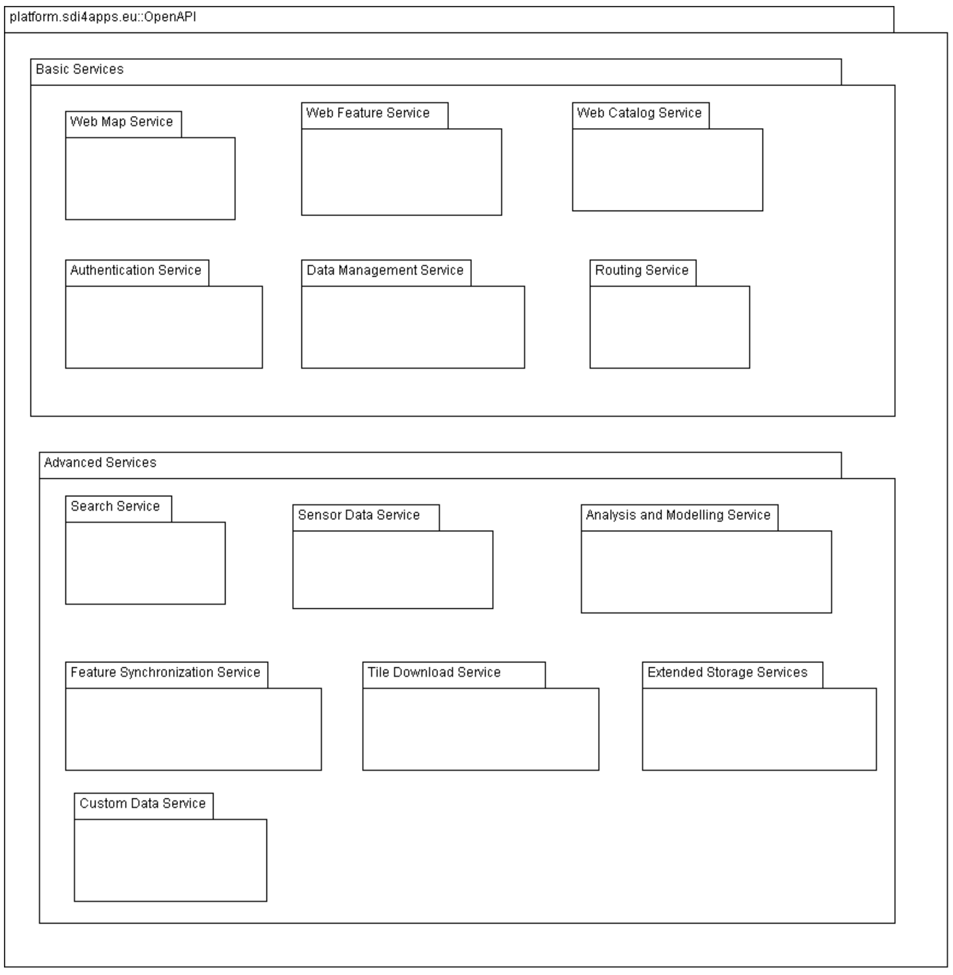
Furthermore, core technologies exploited by the services were assessed as well and the results achieved are documented in this report either. Tests for this assessment are included in this document as well.

2.2 Platform services and technologies under test

Document D3.3.1 defines the six basic services that the project's platform should provide. Here is a list as a summary of all of them:

- Web Map Service
- Web Feature Service
- Catalogue Service
- Authentication Service
- Data Management Service
- Routing Service

Full details about the basic service is provided in D3.3.1, from which the following picture is extracted:



D3.3.1 also includes details about possible technologies to support each service, even though there is still room for change. On this matter, many of the platform basic services envision to rely on the PostgreSQL-XL technology, which addresses the need for scalability within PostgreSQL. The platform testing should include assessing the read and write performance of PostgreSQL-XL clustered database compared to the classic PostgreSQL only installation, comparing performances in terms of time consumption. So the platform tests should include one more item:

- Assessment testing of PostgreSQL-XL versus PostgreSQL

Other similar testing might be necessary, in the future, to make assessments on technologies and tools to introduce in the platform. In that case assessment testing will be included in the next release of this document.

2.3 Test Details for all the items

This section details what should be tested for each item under test defined in the previous section. Each of the following section defines all the parameters needed to set up the tests.

For the Web Map Service and Web Feature Service there is a section including both, since the tests on these two services are conducted at the same time.

For the Authentication Server SDI4Apps platform is using CAS service as authentication service. User management is done through Liferay UI using its control panel. Liferay is also used as authorization service for connected applications to portal. CAS service is standard version which is used broadly in different applications and projects. Therefore, this component will be tested in the next project phases, in a broader context and along with the overall components exploited in each pilot site.

All other services have one section each.

One final section is dedicated to the definition of the test for the PostgreSQL-XL versus PostgreSQL technology assessment.

Many of the tests defined here use Jmeter to simulate scenarios and test performances under heavy load, which Jmeter is specifically designed for. Other tests make use of specific scripts that provide the necessary test fixture (set-up), run the test and provide the results.

2.3.1 Web Map Service and Web Feature Service

Web Map Service (WMS) and Web Feature Service (WFS) will be tested using two data sources provided by PostgreSQL and PostgreSQL-XL. This should be sufficient because most resource consuming operation is often in the selection of correct data from data sources.

OpenStreetMap (OSM) data for Europe will be used as dataset. Test will be done using Jmeter which will simulate three scenarios

- 5 concurrent users, each user making 4 requests
- 10 concurrent users, each user making 4 requests
- 15 concurrent users, each user making 4 requests

Test will be done for GetMap, GetFeature requests. WMS will be tested for large, medium and small scales. WFS will be tested for medium and small scales.

2.3.2 Catalogue Service

CSW will be tested using MIcKA metadata catalogue using two instances with PostgreSQL and PostgreSQL-XL backend. This database will have 1500 records. Test will be done using Jmeter which will simulate three scenarios

- 5 concurrent users, each user making 4 requests
- 10 concurrent users, each user making 4 requests
- 15 concurrent users, each user making 4 requests

2.3.3 Data Management Service

Data Management service will be tested through layman REST interface namely list of datasets in database and list of published layers. List of datasets in database will be tested using PostgreSQL and PostgreSQL-XL. List Layers service is not tested against PostgreSQL-XL because it is read from Geoserver.
Test will be done using Jmeter which will simulate three scenarios

- 5 concurrent users, each user making 4 requests
- 10 concurrent users, each user making 4 requests
- 15 concurrent users, each user making 4 requests

2.3.4 Routing Service

The routing service is a part of the SDI4Apps basic services and the first revision of the service is implemented using the pgRouting module of PostgreSQL and PostGIS. This implements a variety of different routing algorithms that can be invoked on dynamic data that are stored in database tables.
Storing the data to be routed in a database is slower than using a routing engine that is based on a static network, i.e. a road network that is updated 2-3 times per year. Such a routing engine can pre-generate an exhaustive cache of source-destination combinations that will make route searches almost instantaneous. However, the benefit of the dynamic routing mechanism that SDI4Apps has opted for is that it permits the network to be modified in real time. A link in the network can be blocked, different types of networks can be loaded, a new link can be added; these and many other dynamic factors have a great impact on the shortest/fastest/best route between two or more points and are thus important.

Another area of performance where dynamic routing does not measure up to static routing is scalability. The number of nodes that can be handled once they have to be traversed exhaustively for each request is limited.
Thus, dynamic routing is most useful when calculating routes within a finite graph, i.e. the transportation network of a city, the public transit system of a metropolitan area, the utility grid of a power company or similar territorially defined networks.
The routing algorithm that has been implemented in the first version is bi-directional A* shortest path.
The test-dataset is the OpenStreetMap road network for Berlin. This dataset includes 116 440 links within an area of roughly 890 km^2 and illustrates a typical large European city. The rationale for the selection of the test dataset is that it illustrates a relatively large area where the dynamic routing service may be used successfully. This performance scale enables a wide range of potential use cases for both local governments (large scale networks) and regional administrations (intermediate scale networks).
Two different tests should be conducted on the network. The first one is to verify performance during concurrent use of the service. The second is to verify performance for single requests, given complex routing requests.

Concurrent use

The relevant test parameters to verify the performance of the routing service during concurrent use has been determined as follows: the service will have 10 concurrent active user sessions, each issuing 10 routing requests per minute for a period of 5 minutes, resulting in 500 route requests in 300 seconds.
The start and stop nodes are picked at random, resulting in some complicated calculations and some less complicated calculations, as would be the case in a regular concurrent use scenario.
The threshold that the test must pass is less than 600 ms processing time per request in order not to create a queue on the server. However, in order to permit for 300% peak in concurrent use, the target is to achieve a processing time of 200 ms or less.

Per request performance

The second test scenario selects a set of complex routes that are calculated 100 times repeatedly at random intervals, each time measuring the performance in milliseconds.
The test includes three routes, each of which needs to traverse more than 100 links from input start and end points. Route A to C increases in complexity with C being the most complex and A being the simplest.

Tools

The testing is conducted using a web script written in PHP to invoke SQL queries to a PostgreSQL + PostGIS + pgRouting back-end. Data are loaded to the PostgreSQL database from OpenStreetMap data downloaded from geofabrik. The tool osm2pgrouting is used to convert *.osm files to compliant RDBMS tables in PostgreSQL. The repetitions simulating concurrent use are implemented as another PHP script that repeatedly invokes the routing class. The

test is run on a single server that also runs the pgRouting database, thus the tests do not take into account network latency.
However, it is reasonable to assume that the impact of network latency will be individual to each user, thus limiting the frequency of requests from that specific user.

2.3.5 Technology assessment: PostgreSQL-XL vs PostgreSQL

Performance of clustered database solution can be measured on a 2-5 nodes Postgres-XL cluster comparing the results to a normal PostgreSQL solution.
The read performance can be compared trying out 5-10 different views with MapServer on map of Europe. Time taken to provide an answer is to be measured and compared, assessing the opportunity of using a clustered database solution against a classical non-clustered database solution.
Write performance can be measured monitoring the write operations on some 100.000+ write operations on three use cases: data import, geometry and index building. As in the previous case, comparison of time taken to carry out the write operations is to be measured and compared, assessing the opportunity of using a clustered database solution against a classical non-clustered database solution.

3 TECHNICAL TEST REPORT

This section provides the results of testing as defined in the previous chapter. Each of the following sections refers to a section in the previous chapter, where test details are provided.

3.1 Web Map Service and Web Feature Service

The following sections report the testing done on the Web Map and Web Feature services, each section reporting the test results.

3.1.1 Web Map Service

5 users * 4 requests per user

Service	Average (msec)	Min (msec)	Max (msec)
large scale	7758	6351	11289
large scale (XL)	4724	3959	5578
medium scale	3021	1289	7546
medium scale (XL)	1199	1043	1350
small scale	539	350	796
small scale (XL)	472	288	824

Test for 10 users * 4 requests per user

Service	Average (msec)	Min (msec)	Max (msec)
large scale	12954	10496	14380
large scale (XL)	9146	7677	10494
medium scale	2336	1478	2862
medium scale (XL)	2282	1191	3124
small scale	815	459	1215
small scale (XL)	682	357	1007

Test for 15 users * 4 requests per user

Service	Average (msec)	Min (msec)	Max (msec)
large scale	19652	16819	21553
large scale (XL)	14077	12831	15351
medium scale	3622	1185	4830
medium scale (XL)	3379	1831	4473
small scale	1178	742	1655
small scale (XL)	1054	288	1641

3.1.2Web Feature Service

5 users * 4 requests per user

Service	Average (msec)	Min (msec)	Max (msec)	Requests/sec
medium scale	141	101	225	21.3
medium scale (XL)	120	91	220	23.3
small scale	139	100	261	21.6
small scale (XL)	121	90	226	23.3

Test for 10 users * 4 requests per user

Service	Average (msec)	Min (msec)	Max (msec)	Requests/sec
medium scale	117	98	165	30.0
medium scale (XL)	111	91	154	30.1
small scale	11	100	144	28.7
small scale (XL)	108	89	136	29.9

Test for 15 users * 4 requests per user

Service	Average (msec)	Min (msec)	Max (msec)	Requests/sec
medium scale	256	110	468	31.0

medium scale (XL)	182	97	396	36.6
small scale	232	100	425	32.7
small scale (XL)	179	95	292	38.5

3.1.3 Catalogue Service

The following tables report the test results on the Catalogue Service.

Test for 5 users * 4 requests per user

Service	Average (msec)	Min (msec)	Max (msec)	Requests/sec
get all records	648	562	785	6.0
get all records (XL)	561	275	735	6.7
get filtered records	566	494	642	6.5
get filtered records (XL)	2353	2180	2486	2.0

Test for 10 users * 4 requests per user

Service	Average (msec)	Min (msec)	Max (msec)	Requests/sec
get all records	1207	596	1553	7.0
get all records (XL)	1202	358	1842	7.0
get filtered records	1045	501	1349	7.8
get filtered records (XL)	3581	3105	3883	2.7

Test for 15 users * 4 requests per user

Service	Average (msec)	Min (msec)	Max (msec)	Requests/sec
get all records	1867	684	2242	7.1
get all records (XL)	1897	578	2819	7.0
get filtered records	1597	657	2002	8.2
get filtered records (XL)	5034	3324	6256	2.8

3.1.4 Data Management Service

Test for 5 users * 4 requests per user

Service	Average (msec)	Min (msec)	Max (msec)	Requests/sec
list datasets	216	181	271	12.1
list datasets (XL)	192	165	274	13.0
list layers	269	219	322	10.8
list layers (XL)	N/A	N/A	N/A	N/A

Test for 10 users * 4 requests per user

Service	Average (msec)	Min (msec)	Max (msec)	Requests/sec
list datasets	331	191	489	18.2
list datasets (XL)	306	177	489	19.5
list layers	486	249	749	14.2
list layers (XL)	N/A	N/A	N/A	N/A

Test for 15 users * 4 requests per user

Service	Average (msec)	Min (msec)	Max (msec)	Requests/sec
list datasets	553	240	775	19.3
list datasets (XL)	466	180	1295	20.2
list layers	832	249	1291	14.1
list layers (XL)	N/A	N/A	N/A	N/A

3.1.5 Routing Service

The following sections report the test results on the Routing Service on both concurrent use and per request performance.

Concurrent use

	First run	Second run	Third run	Fourth run

Time	215 ms	220 ms	250 ms	214 ms

As is evident from the test-results, additional optimization is required if the performance targets for handling peak traffic are to be met. However, all four iterations demonstrate that the service is capable of handling the envisaged volume of use as defined by the test.

Per request performance

	Route A	Route B	Route C
Average performance (ms)	344 ms	572 ms	1 291 ms
Slowest performance (ms)	532 ms	807 ms	2 441 ms

It is evident from the testing that complex routing operations are CPU intensive and that some level of short-term caching may be required to prevent repeated queries for common routes to consume too much system resources.
Except for route C, the other routes performed within the performance target of 600 ms - but it must be assumed that the heterogeneous requests received from mixed use will be a mix of requests with varying complexity.

3.1.6 Technology assessment: PostgreSQL-XL vs PostgreSQL

Performance of clustered solution was measured on a 3-node Postgres-XL cluster versus a classical non-clustered PostgreSQL installation.
Hardware configuration of clustered database:

Node	CPU	RAM (GB)
Node1	4	12
Node2	4	2
Node3	4	2

Hardware configuration of classical PostgreSQL database:

Node	CPU	RAM (GB)
PostgreSQL	4	12

Read performance was measured by creating eight different views with MapServer on map of Europe. In Postgres-XL each database table was distributed among nodes by hashing the content of its id column. All measured values are summarized in the following table (all values are in seconds).

	View 1 (sec)	View 2 (sec)	View 3 (sec)	View 4 (sec)	View 5 (sec)	View 6 (sec)	View 7 (sec)	View 8 (sec)
Postgres-XL	19.949	12.054	15.895	7.798	18.227	11.770	8.231	5.341
PostgreSQL	5.248	11.817	12.387	1.421	10.980	7.697	4.048	2.268

The results show that classical PostgreSQL was faster than clustered database in all the test runs. For a single read query a single PostgreSQL instance is a better solution than a clustered Postgres-XL database.
Write performance was measured on three use cases – data import, geometry and index building. Test data was a file containing 200724 items of US Populated Places gazetteer. Results are expressed in seconds in the following table:

	Postgres-XL (sec)	PostgreSQL (sec)
Data import	0.633	1.353
Geometry building	1.003	2.982
Index building	0.550	2.560

Writing into clustered database showed significant improvement, which could be even better with a database distributed among higher amount of nodes.

4 CONCLUSION

Four basic services of SDI4Apps platform (WMS, WFS, Catalogue Service and Data Management Service) were tested against classical PostgreSQL and clustered Postgres-XL database, which should be able to take full advantage of cloud infrastructure. All services were tested with multiple users and multiple requests per user. Additionally Postgres-XL was tested for complex single user read queries and sample write queries.

Benchmark demonstrated, that clustered database, specifically Postgres-XL, isn't a general solution for improving performance of spatial queries. Especially complex single user read queries were all slower than classical PostgreSQL. WMS, WFS and Data Management Service showed in some cases significant improvement, which can be further enhanced by adding more nodes, or by adjusting data partitioning for particular queries.

Write performance is a strong feature of clustered database – benchmark showed scale factor better than ½ for 3-node setup. These results make it a good solution for applications with heavy data output like data collecting from Internet of Things.