Technical Test Report 2

MARCH 2017

SDI4Apps

DELIVERABLE

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

D3.6.2 TECHNICAL TEST REPORT 2

Revision no. 03

Authors: D. Tarini, A.Iembo, N. Zanetti (Hyperborea)
S.R. Bergheim (AVINET)
M. Tuchyna (SAZP)
M. Kuba (MU)
T. Sapak (MU)
K. Charvat (CCSS)
D. Kozhukh (CCSS)

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	16/03/2017	Alfredo Iembo	Hyper	Initial draft based on previous report
02	29/03/2017	All authors	All contributors	Contributions for each module/component
03	30/03/2017	Alfredo Iembo	Hyper	Final release

TABLE OF CONTENTS

Revision History 3
Table of Contents 4
Executive Summary 6
1 Introduction 7
2 technical Test Methodology 8
2.1 Introduction 8
2.2 Platform Services and Technologies Under Test 9
2.3 Test Details for All the Items 10
2.3.1 Web Map Service and Web Feature Service 11
2.3.2 Catalogue Service 11
2.3.3 Data Management Service 11
2.3.4 Routing Service 11
2.3.5 Technology Assessment: PostgreSQL-XL vs PostgreSQL 14
3 Technical Test Report 15
3.1 Web Map Service and Web Feature Service 15
3.1.1 Web Map Service 15
3.1.2 Web Feature Service 16
3.1.3 Catalogue Service 17
3.1.4 Data Management Service 18
3.1.5 Routing Service 18
3.1.6 Technology Assessment: PostgreSQL-XL vs PostgreSQL 19
4 Conclusion 21

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.3.6.2 “Technical Test Report 2” focuses on the testing of advanced services, while the basic services have be taken care of in previous document D.3.6.1 " Technical Test Report 1"., In the light of the above, this document focuses on tests performed and their results from final phase of project implementation and from utilisation of the platform from the side of the users and developers.

The test methodology was outlined and established in previous versions of test report so it has also been applied to the final version of the platform, after upgrading and completion.

So, the final version of test report include description and results of tests defined for modules/components added in last year, tests results for modules/components modified in last year and confirmation of test for modules/components not modified in last year.

In this way, this final report will provide a comprehensive and sustainable report of platform tests and results.

The outcomes have also been improved through appropriate tuning activities implemented by the use of the platform by both external stakeholders and the Pilots, in particular of optimizing the performance of the various components.

It is worth pointing out that the final testing were performed for each module after the related finalisation achieved also through the valuable contributions/feedback etc. provided by third parties/external stakeholders that effectively contributed to the Validation Phase of the project. In fact, all the Code Camps, Hackathons, national events/meetings organised with external developers allowed the SDI4APPS partners, responsible of each platform module, to gather suggestions and comments that were leveraged to tune the modules themselves, resulting in better performances as documented in the current version of the final project technical test phase

1 INTRODUCTION

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

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.

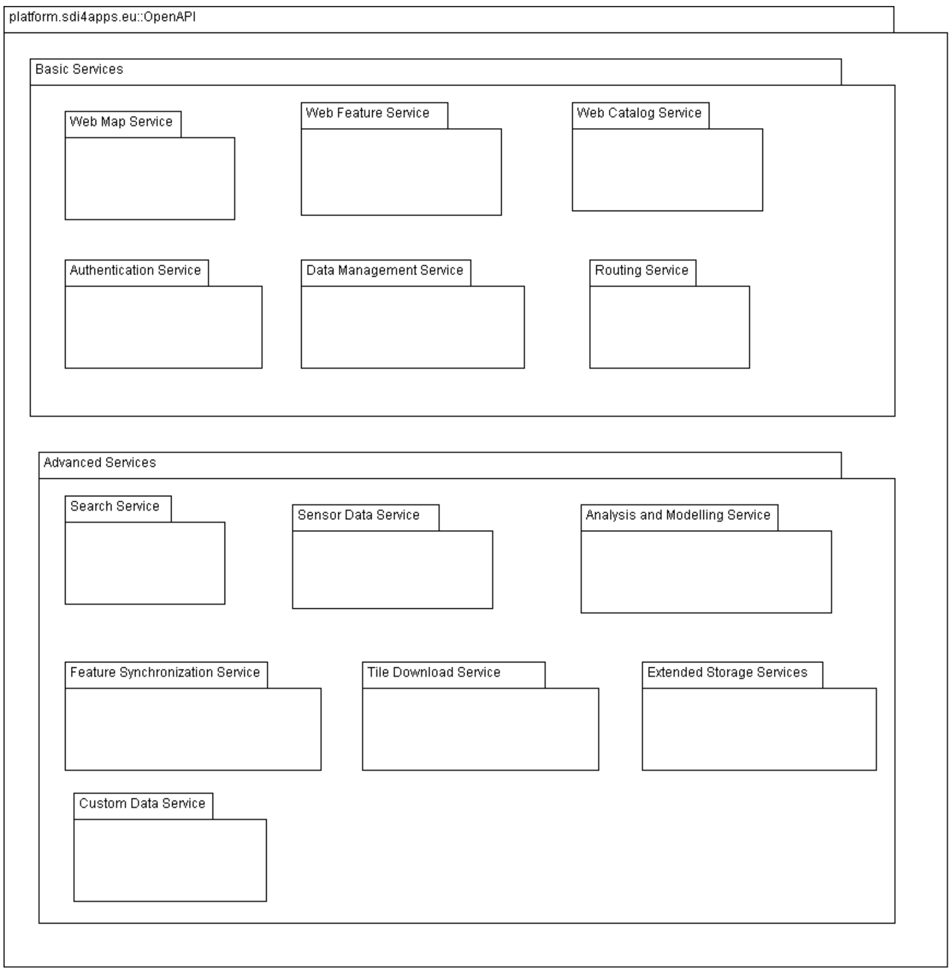
Furthermore, core technologies exploited by the services were assessed as well and the results achieved are documented in this report either. Tests for this assessment are included in this document as well.

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.

Improvements in final release

Since the first release, routing has been integrated with HSLayers NG through the s4a.js library. Through this integration, a large number of routing requests have been processed by the API and it has been possible to derive knowledge on bottlenecks to efficient operation in real-world usage.

The performance of the routing operations is bound by RAM and disk speed. The operation is based on reading all nodes of the network into memory and then performing calculations without further disk reads. This means that the optimal operation would be achieved if the entire graph at all times was loaded into memory on the server instead of loading it at query time.

In a shared server scenario, as is the case with the SDI4Apps platform, this is obviously not practical as it would permanently lock down system resources even when there is no load on the server.

As stated in the test scenario below, the majority of queries are ‘intracity’ and will therefore not lead to very large graphs. However, in order to tackle the border cases, two strategies were attempted.

1. Limiting the number of nodes by adding a spatial filter to the network.
    a. A bounding box was calculated based on the start- and end-point
    b. The bounding box was expanded by 200%
    c. Only nodes which intersected the bounding box was loaded into memory
2. Dividing the network into sub-graphs
    a. Find shortest route to a highway from starting point using local roads
    b. Find shortest route to a highway from the end-point using local roads
    c. Find the shortest route between highway point A and highway point B using highways

d. Combine the three results and present to user

Both methods yielded performance improvements by orders of magnitude for long routes. The second has the benefit that it does not falsely report 'no route' whereas the first might do so in the case of large horse-shoe shaped road networks.

To partly mitigate this issue, the buffered bounding box (1, above) will at minimum be 20 km wide, regardless of the distance between the start- and end-point. This lead to slight increases in calculation times for short routes (+/- 5ms) but major performance improvements to calculation times for intermediate routes (+/- 120ms). Very long route calculations are only improved by strategy 2.

Testing

The test-dataset employed for version 1.0 of the solution was the OpenStreetMap road network for Berlin. The rationale for that test dataset was that it illustrated a relatively large area where the dynamic routing service may be used successfully. In the final release, the test data set has been extended to two countries; Germany and Austria. Instead of OSM, the data are now sourced from the Open Transport Map .

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	160 ms	153 ms	174 ms	172 ms

All four iterations demonstrate that the service is capable of handling the envisaged volume of use as defined by the test. Improvements in performance over the previous version ranges from 70-80ms per request.

Per request performance

	Route A	Route B	Route C
Average performance (ms)	122 ms	246 ms	507 ms
Slowest performance (ms)	267 ms	702 ms	1 623 ms

The performance improvements in the final year of the project have shown good results: about ˜80 ms for simple routing requests, ˜140 ms for intermediate requests and more than half a second for long route requests. It is however to be noted that very long route requests still are slow.

3.1.6Technology Assessment: PostgreSQL-XL vs PostgreSQL

Performance of clustered solution was measured on a 5-node Postgres-XL cluster versus a classical non-clustered PostgreSQL installation. Hardware configuration of all nodes:

CPU	RAM	HDD
4	4 GB	300 GB

Database data was stored on Ceph (http://ceph.com/) storage platform equipped with SSD for journal.

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	View 6 (sec)	View 7 (sec)	View 8 (sec)

					(sec)			
Postgres-XL	13.670	5.323	6.173	1.163	5.156	3.610	1.651	0.914
PostgreSQL	5.073	8.373	6.998	0.510	11.369	4.732	1.243	0.390

The results show, that classical PostgreSQL was slower in most test runs than the clustered solution. Just in the first view is clustered solution significantly slower, so it can't be recommended as universal replacement for PostgreSQL.
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	0.794	1.395
Geometry building	0.616	2.821
Index building	0.363	2.442

Writing into clustered database showed significant improvement in all measured areas.

4 CONCLUSION

The tests defined in the methodology were applied to the final version of the platform, and reported in the final version of this test report. It is noted that the use of the platform allowed us to do some tuning measures in the components and this led to a performance improvement.

Also, the current version of the platform, as proven by the use by the Pilot both the camp and hackatons code, is now stable and suitable to be exploited as planned related deliverables (exploitation plan, sustainability plan).

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.