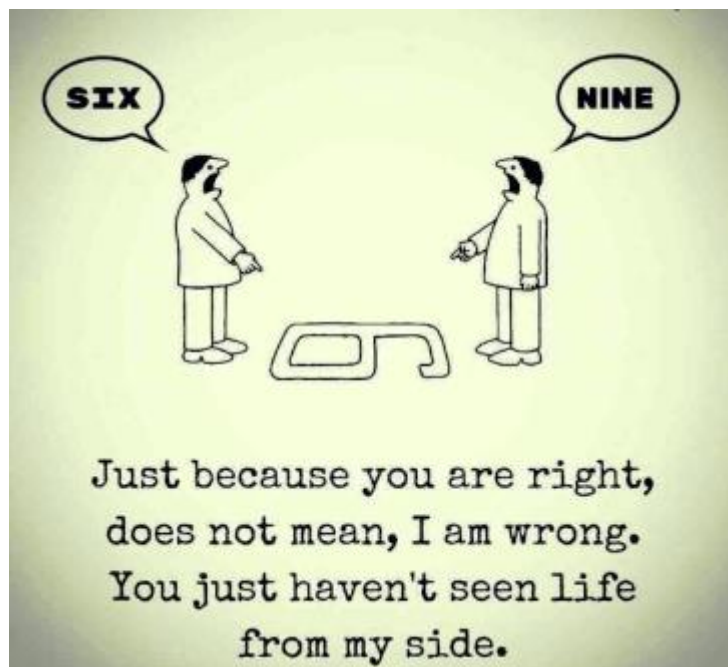


Modern ways of spatial data publication

Report of the research results in the Geonovum testbed
“Spatial Data on the Web”



14/09/2016

Authors

T. van Sprundel, Spotzi BV

E. Nagelkerke, Spotzi BV

R. Dolman, Spotzi BV

M. Pegels, Spotzi BV



Table of Contents

Abstract.....	2
Introduction	3
Research results.....	4
Data testing.....	4
Findable	4
Understandable.....	5
Connection to the data	9
Content-types.....	11
Findable by crawlers.....	13
Web application.....	14
Category	14
Zip code search.....	14
Area search.....	14
Connection to the data.....	15
Usability.....	15
Adding the data to the platform (without storage).....	16
Application performance.....	17
Other connections.....	18
Evaluation	19
Lessons learned.....	19
Lesson 1: Everyone in a platform or community has their own needs and capacities.....	19
Lesson 1A: Make sure the needle can be found in the haystack	19
Lesson 1B: Keep it simple	19
Lesson 1D: Each speaks its own language and lives in his own world.....	19
Lesson 2: Make search engines feel comfortable to discover you.....	20
Lesson 2A: Show a search engine the direction with an XML sitemap	20
Lesson 2B: Foster to link everything with everything.....	20
Lesson 2C: Think of the future, use persistent URIs.....	20
Lesson 2D: Make use of the structure, include Schema.org markup.....	20
Lesson 3: Deal with the unknown set of developers and devices.....	21
Lesson 3A: Serve your data in many different flavors.....	21
Lesson 3B: Improve performance, reduce payload.....	21
Lesson 4: Don't copy data, use proxy	22
New lessons:	22



Abstract

This document is the report containing the research results from Spotzi in the Geonovum testbed "Spatial Data on the Web". For this testbed Spotzi researched the usability and correctness of Geonovum's lessons learned¹ on modern ways of geospatial data publication.

The testbed started in the last weeks of 2015 with topic #2, #3 and #4. The results of these topics were combined into a list of lessons learned and were presented on GitHub. In the summer of 2016 three parties, Triply, Spotzi and Alterra worked together on topic #1. For this topic, Triply acted as a data provider and researched how the lessons learned helped data providers with providing their data. Spotzi and Alterra approached this topic from the perspective of the data user. It was their goal to find out how the lessons learned improved the conditions of the data users.

Spotzi's approach for this testbed was to develop a web application that would use the geo data provided by Triply's data platform. In the different phases of development, Spotzi checked what lessons learned would apply for that step, and then reflected on the lessons learned. The different steps in our development process can be summarized in testing the data platform, connecting to the data platform, querying the data platform and evaluating the data platform.

The result of this research is an evaluation of the lessons learned from the perspective of the web developer. Spotzi describes what lessons learned are well formulated, which lessons learned need adjustment and offers propositions for missing lessons learned. All other information about this testbed is published on the geo4web-testbed GitHub page².

¹ <https://github.com/geo4web-testbed/lessons-learned/wiki>

² <https://github.com/geo4web-testbed>



Introduction

Geonovum sees the web as an important channel and wants geospatial data to be accessible to web developers with no specific geospatial expertise. That is why Geonovum organized a Testbed to explore the possibilities of making spatial data a useful, integral and a common part of the web.

Previous tenders have focused on spatial data usability on the web for users and accessibility for machines. These tenders³ resulted in a list of lessons learned. Finally, the following research question was formulated:

How do these lessons learned meet the constraints (e.g. budgets) and capabilities (e.g. in-house know-how) of governmental organizations on the one hand, and of data users on the other?

For this research, Spotzi focuses on the data user side of the main research question as listed above. To come to an answer, we focused on the following tasks (as listed in the tender):

- a. Creation of a working application using the data as described in the tender.
- b. Is the data findable, usable and programmable?
- c. Does the information in the lessons learned helps us, web developers, to work with the data?

Regarding the lessons learned we focus on answering the questions listed below (see chapter evaluation):

- Evaluate the usability of the lessons learned. How useful are the lessons learned as documentation? Are the contents understandable, the examples workable and is the form of the documentation useful?
- Report any errors found in the lessons learned.
- Identify any gaps in the lessons learned, i.e. aspects of spatial data publishing on the web about which guidance is needed, but missing in the lessons learned.
- Identify any bad advice in the lessons learned. Are any parts of the lessons learned arguably not good practice at all?
- Give suggestions, preferably as text proposals, for the improvement of the lessons learned.

This report contains the following aspects:

- *Testing the data provided by task 1 (data publisher)*
- *Creating the web application*
- *Create a connection to the data*
- *Create other connections*
- *Add logic to the web application*
- *Evaluation of the lessons learned*

In the next chapters our research results are presented.

³ <https://github.com/geo4web-testbed/>

Research results

In this chapter we will reflect on the research period of the project. The main focus of the research was finding errors in the lessons learned. To come to an evaluation, we decided to test the API of the selected data provider, Triply, together with known API's from the previous tender. In this test we planned to start with some simple data testing. From there we would expand our research with integrating data from multiple platforms into one web application.

Data testing

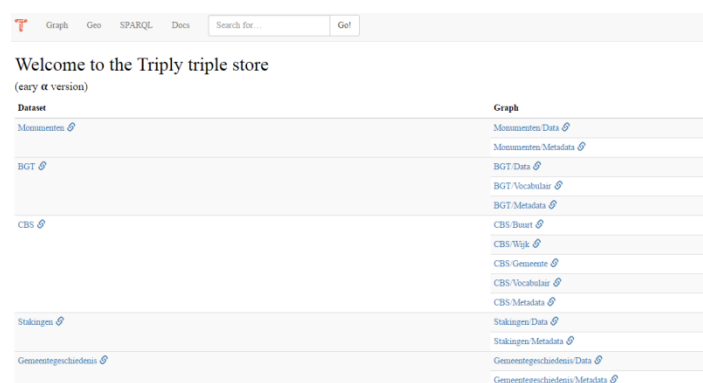
Triply acts as data publisher, they will publish data, like the Cultural Heritage data, in compliance with the lessons learned from the project Spatial data on the Web. This section focuses on testing the Cultural Heritage data provided by Triply. Answers will be provided on the questions: Is the Cultural Heritage data findable and understandable? Is it possible for us to retrieve data from this dataset and how did the retrieval of the data go? We will also look into the different content-types Triply provides. Finally, we look at the question if the provided data is findable by crawlers. Finding, accessing and using data is often difficult for non (geo)-experts.

Findable

Triply realized the Geo API which can be accessed via:

<http://geonovum.triply.cc/>, see Figure 1.

This data providing platform provides (accessed on 18th August 2016) multiple datasets, namely BGT, Monuments (Cultural Heritage data), CBS data, "Stakingen" and "Gemeentegeschiedenis".



Welcome to the Triply triple store (early alpha version)	
Dataset	Graph
Monumenten	Monumenten/Data
	Monumenten/Metadata
BGT	BGT/Data
	BGT/Vocabulary
	BGT/Metadata
CBS	CBS/Burst
	CBS/Wijk
	CBS/Gemeente
	CBS/Vocabulary
	CBS/Metadata
Stakingen	Stakingen/Data
	Stakingen/Metadata
Gemeentegeschiedenis	Gemeentegeschiedenis/Data
	Gemeentegeschiedenis/Metadata

Figure 1: Triply Home

With future developers in mind we decided to take on multiple perspectives in testing the data. For testing the findability, we started with the perspective of a developer, who is looking for a certain dataset and arrives on the data providing platform for the first time. Here the developer expects to find the first steps he needs to know for using the API, including a demo for retrieving (example) responses in the different formats the API provides.

Looking through the website, we found the graph-, geo- and SPARQL-endpoints where we could retrieve data in different formats. In the paragraph *Understandable* we will go further on these. Next to the endpoints, we also found some documentation on the graph- and geo-endpoints. Documentation on the SPARQL-endpoint was not found. From the documentation page we went back to the main page. Here we found the table with the available datasets and graphs listed. We decided to explore the Cultural Heritage dataset.

Firstly, we clicked "Monumenten" under the header "Dataset". This brought us to the metadata⁴ for the Cultural Heritage dataset. Which contained no useful information, apart from the ID's connected to this dataset. Next to that, we were a bit confused since we expected to see the actual dataset.

⁴ <http://geonovum.triply.cc/graph?graph=http%3A//rdfs.org/ns/void%23c39e1092fd8387233e60222952f11a2a>

Secondly, from the home page, we found the actual data⁵ of the Cultural Heritage dataset. This was done by clicking the link “Monumenten/Data” under the header “Graph”. Here we found out that the data was not served in objects. It was however served as object-fields. Which means that one object consists out of multiple objects, that all refer back to “main” object. For example, a house would have a number, a street, a zip code, a city and et cetera, which all refer back to the house with an ID.

When a website is built with a user-friendly web design, understanding the structure of the data will be easy and the developer will not have to search for a long time. Keep in mind that users want to know where they go before clicking a link and that people think in objects, not in properties.

Understandable

Now the developer has found the data he is going to use for his application he needs to understand how he can use it to his working. And in consideration of Triply, how he should be using it. For this step we expect the developer to look at his application or testing software next to his browser. That means that understanding the Triply-API should be done in multiple perspectives. So next to understanding the API from the browser, the developer should also be able to understand the API as seen from the application (or testing software). Which reads the raw response from the server in a requested format. For example, in JSON-LD or GeoJSON.

Human understandability

In the previous paragraph we already looked at the in-browser data presentation of the Cultural Heritage data under the graph-endpoint. We couldn't understand immediately what we were seeing. This is because we expected to see multiple objects listed beneath each other, with each one's properties next to each other. However, this is not how the graph-endpoint works. Here, the following structure is understood:

- One feature on a map is called a **subject** in the graph-endpoint.
- The properties of a subject are defined in **objects**.
- The **predicate** defines the type for the object.

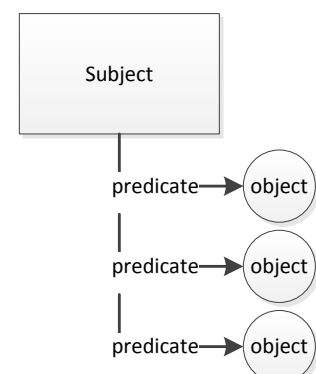



Figure 2: Graph Structure

In *Figure 3* you can see that this structure is applied. Here the first nine rows belong to the first subject. The remaining rows belong to the second subject, with each row having a different object with feature data and predicate with a data type.

⁵ <http://geonovum.triply.cc/graph?graph=http%3A//lodlaundromat.org/data/c39e1092fd8387233e60222952f11a2a>

<div>  Graph Geo SPARQL Docs <input type="text" value="Search for..."/> <input type="button" value="Go!"/> </div>			
Subject	Predicate	Object	Graph
file:/F:/LOD/tbc/tbcworkspace/TopBraid/NDE/alle_m...	rdf:type	owl:Ontology	Monumenten/Data
file:/F:/LOD/tbc/tbcworkspace/TopBraid/NDE/alle_m...	owl:imports	dbo:	Monumenten/Data
file:/F:/LOD/tbc/tbcworkspace/TopBraid/NDE/alle_m...	owl:imports	dc:	Monumenten/Data
file:/F:/LOD/tbc/tbcworkspace/TopBraid/NDE/alle_m...	owl:imports	dct:	Monumenten/Data
file:/F:/LOD/tbc/tbcworkspace/TopBraid/NDE/alle_m...	owl:imports	http://topbraid.org/schema/	Monumenten/Data
file:/F:/LOD/tbc/tbcworkspace/TopBraid/NDE/alle_m...	owl:imports	http://www.w3.org/2000/01/rdf-schema	Monumenten/Data
file:/F:/LOD/tbc/tbcworkspace/TopBraid/NDE/alle_m...	owl:imports	http://www.w3.org/2003/01/geo/wgs84_pos	Monumenten/Data
file:/F:/LOD/tbc/tbcworkspace/TopBraid/NDE/alle_m...	owl:imports	http://www.w3.org/2006/vcard/ns	Monumenten/Data
file:/F:/LOD/tbc/tbcworkspace/TopBraid/NDE/alle_m...	owl:imports	foaf:	Monumenten/Data
http://data.cultureelerfgoed.nl:3333/10	dbo:city	Amsterdam	Monumenten/Data
http://data.cultureelerfgoed.nl:3333/10	dbo:code	Gebouwen, woonhuizen	Monumenten/Data
http://data.cultureelerfgoed.nl:3333/10	dbo:codeNationalMonument	10	Monumenten/Data
http://data.cultureelerfgoed.nl:3333/10	dbo:coordinates	121778	Monumenten/Data
http://data.cultureelerfgoed.nl:3333/10	dbo:coordinates	487409	Monumenten/Data
http://data.cultureelerfgoed.nl:3333/10	dbo:municipality	Amsterdam	Monumenten/Data
http://data.cultureelerfgoed.nl:3333/10	dbo:province	Noord-Holland	Monumenten/Data
http://data.cultureelerfgoed.nl:3333/10	dbo:type	gebouwd	Monumenten/Data
http://data.cultureelerfgoed.nl:3333/10	http://example.com/huisnummer	43	Monumenten/Data
http://data.cultureelerfgoed.nl:3333/10	http://example.com/huisnummerCompleet	43	Monumenten/Data
http://data.cultureelerfgoed.nl:3333/10	geold:geometry	POINT(4.899352932345246 52.373550285458975)	Monumenten/Data

[Previous](#)
[1](#)
[20](#)
[Next](#)

Figure 3: Graph endpoint

As we browsed through the pages we saw that we are dealing with some kind of stream output. We looked at the data in the output and the links that are printed on the page. With each page we went further, the browser requested the next set of objects referring to different subjects. Looking through the dataset raised the following questions:

- Is the graph endpoint **always** ordered by subject so that objects of one subject always come one after another (without objects from different subjects coming in between)?
- Is there a possibility to order the blocks of subjects on one of their predicates?
- We encountered invalid links. What are further the requirements of forming a ID-value, regarding:
 1. **Validation**, should all links be working links or can they be part of an API-call?
 2. **Domains**, should subjects from one dataset exists on one domain only, or can there be a variation?
- Objects of the same subject can be split over multiple pages. Can, and how can developers check if they have retrieved all objects (from one subject) without making a second request? For example, when requesting the objects of just one subject exceed the limit of *page_size*.

Computer understandability

After having seen the browser it was time to compare the API-requests with a different format. For this we decided to use the Google-plugin Postman⁶. With this plugin it is possible to define and simulate requests. As feedback this application gives the response together with its response code, response time and other response data.

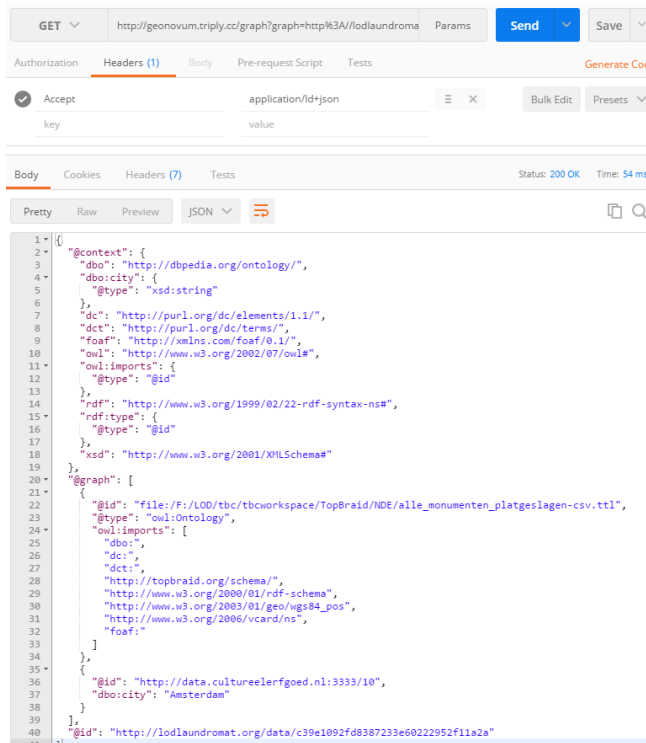


Figure 4: Graph Response

We looked up the JSON-LD version of the graph response for the Cultural Heritage dataset, see *Figure 4*. Here we saw a different structure than we saw earlier in the browser. Firstly, under `@context`, the given field-types were explained with links or basic value types (string, integer, et cetera). This information was either missing or unclear in the browser-version of this request. Secondly, under `@graph`, we saw a list of features with fields. These were the same features as in the browser. However, here they were all grouped by the subject. This made the illusion that for this request the *page_size* was determining the number of features to be requested.

In *Figure 5* and *Figure 6* on the next page, you can see the difference. First, the expanded feature only has a city and a code. When the *page_size* was set to 28, in *Figure 6*, you can see all the feature's objects.

⁶ <https://www.getpostman.com/>


```
{
  "@context": {,
  "@graph": [
    {,
    {
      "@id": "http://data.cultureelerfgoed.nl:3333/10",
      "dbo:city": "Amsterdam",
      "dbo:code": "Gebouwen, woonhuizen"
    }
  ],
  "@id": "http://lodlaundromat.org/data/c39e1092fd8387233e60222952f11a2a"
}
```

Figure 5: Graph page size 11



```
{
  "@context": {,
  "@graph": [
    {,
    {
      "@id": "http://data.cultureelerfgoed.nl:3333/10",
      "@type": "dbo:Monument",
      "dbo:city": "Amsterdam",
      "dbo:code": "Gebouwen, woonhuizen",
      "dbo:codeNationalMonument": "10",
      "dbo:coordinates": [
        "121778",
        "487409"
      ],
      "dbo:municipality": "Amsterdam",
      "dbo:province": "Noord-Holland",
      "dbo:type": "gebouwd",
      "dct:description": "Huis (XVII of XVIII) waarvan de gevel bestaat uit een houten pui (XVIII) en een bovenstuk onder rechte lijst met consoles (XIXc).",
      "dct:subject": [
        "Woningen en woningbcomplx",
        "Woonhuis"
      ],
      "geold:geometry": "POINT(4.899352932345246 52.373550285458975)",
      "http://example.com/huisnummer": "43",
      "http://example.com/huisnummerCompleet": "43",
      "sorg:relatedLink": "http://monumentenregister.cultureelerfgoed.nl/php/main.php?cAction=show&cOBjnr=10",
      "vcard2006:hasStreetAddress": "Oudezijds Achterburgwal",
      "vcard2006:postal-code": "1012 DB",
      "vcard2006:street-address": "Oudezijds Achterburgwal 43 te Amsterdam"
    }
  ],
  "@id": "http://lodlaundromat.org/data/c39e1092fd8387233e60222952f11a2a"
}
```

Figure 6: Graph page size 28

As described in the paragraph *Findable*, a web design should help to understand the structure of the data. The Semantic data model or vocabulary of the published data should be implemented with an easy overlook of the full dataset or should be at least easy accessible. For example, give the user a simplified representation of the full dataset. Currently on the Triply-API the data, metadata and vocabulary can be accessed, but understanding this from browser view is quite complex.

Firstly, we advise data providers to take a good look at the report of topic 4⁷, and especially chapter *Representations*. We noticed structure within the data on the Graph API of triply, but this is mainly because we have foreknowledge about this dataset. It should be clear to developers that “a feature on a map” is defined as a Subject and that each Subject consists of multiple objects that refer to its subject. Secondly, we advise data providers to take a look at their website and improve the usability of their website. Usability is one of the key elements of understanding. For instance, making it available to switch between response formats on the website will make it easier to understand the differences between them.

Connection to the data

For connecting the data, we used graph, geo and SPARQL endpoints. Our goal in this part of the research was to test if it was possible for a data user to create a connection to the different datasets and discover what obstacles our web developers would encounter. Another important aspect is the time to the first successful call, as described in lesson 3 of the lessons learned.⁸

Connecting

For testing the connection to the data provided on the data platform, we’ve let different developers from Spotzi create their own connection. All of these developers are experienced programmers, but not all of them have experience with linked and/or geo data.

Concerning the time to the first successful call ([TTFSC](#)), we’ve encountered no problems. All of our developers were able to make a connection to the platform and retrieve data within 10 minutes. This short TTFSC was mainly the result of the clear examples found on the [GitHub](#) of topic one.

Parameters

We were able to use a number of mandatory parameters in our calls to the platform, like the *lat*, *lng*, *page* and *page_size*. However, for using the geo-api the current parameters were not sufficient. For linked *spatial*-data to work an API should be equipped with some base functionality.

Firstly, geo-processing takes a huge amount of performance from the server. The more objects fall within your select, the more data has to be processed. We recommend adding a *range* combination to the *lat* and *lng* parameters, or to replace them with a *bounding box* parameter. Then, the server can pre-filter its output to save processing-time.

Secondly, other geospatial filtering is welcome. For example, being able to search specific geometry types like “point”, “line” or “polygon”.

Both parameters are in *parallel* with lesson 3b and will reduce the payload of the server. Spotzi advises to make a list of spatial functionalities that should be included in every geospatial-linked-data API. This list should give linked data specialists an idea of where to begin when they want to implement geospatial in their linked data platform.

⁷ <https://github.com/geo4web-testbed/topic4>

⁸ <https://github.com/geo4web-testbed/lessons-learned/wiki/Lesson-3:-Deal-with-the-unknown-set-of-developers-and-devices>

SPARQL

We've also let our developers make a connection trough the [SPARQL endpoint](#). The TTFSC here was a lot higher, with an average time over 30 minutes. This was mainly due the lack of documentation and explanation on the website. We will go deeper into this subject in the chapter *"Connection to the data"*.

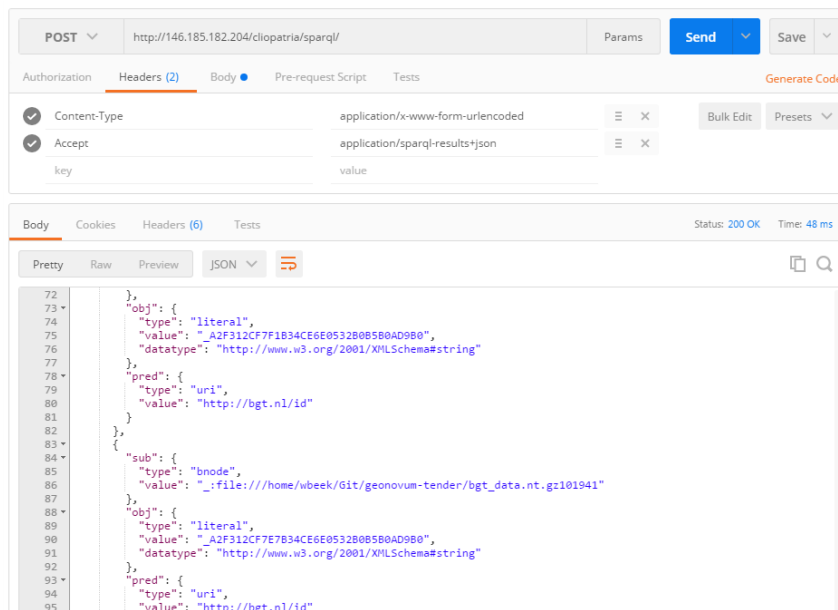


Figure 7: Querying the SPARQL endpoint with Postman

Content-types

We tried to request different content-types from the platform, we used the geo-endpoint for this since our biggest interest lies in the geometry data. We will compare the JSON-LD format (*linked-data*) to the Geo-JSON (*G/S*) format. Overall providing multiple content types, according to the lessons learned, helped our developers a lot. They could quickly find a content type that they were used to work with and this resulting in a good comprehensibility of the data.

JSON-LD

The JSON-LD format was small and easy to understand. Every object has an id and a geometry field, which (if not immediately) can be displayed on a map with ease. Unlike the graph end-point, here the *page_size* was determining the number of objects to be requested. This could lead to some confusion on using the API and will give developers a bigger step-in time. Spotzi recommends to either change the *page_size* name, or to change the working of it in the graph endpoint. A data provider should provide clarity on his data providing platform.

```
{
  "@context": {
    "geold": "http://geojsonld.com/vocab#",
    "geold:geometry": {
      "@type": "wkt:point"
    },
    "nsid": "http://geonovum.triply.cc/id/",
    "wkt": "http://geojsonld.com/wkt#"
  },
  "@id": "nsid:paal/_A2F3132CD221F50D8E0532B0B5B0A22E9",
  "geold:geometry": "POINT(5.4608242216248915 51.35170268429771)"
}
```

Figure 8: JSON-LD

We also noticed a small error in the JSON-LD response. When requesting multiple geographic object types, only one object type was displayed. In the example below you can see a polygon and point in the response. However, only a polygon value is given in the context (*@context/geold:geometry/@type*). This might seem like a small bug, but that doesn't change the fact that the context is incomplete and not to be trusted by developers. Spotzi recommends limiting the response to one geometry type until multiple geometry types can be given in the context.

```
{
  "@context": {
    "geold": "http://geojsonld.com/vocab#",
    "geold:geometry": {
      "@type": "wkt:polygon"
    },
    "nsid": "http://geonovum.triply.cc/id/",
    "wkt": "http://geojsonld.com/wkt#"
  },
  "@graph": [
    {
      "@id": "nsid:overigBouwwerk/_A2F3132C7FDE850D8E0532B0B5B0A22E9",
      "geold:geometry": "POLYGON((5.46078970262002 51.351714319585845,5.460805315381654 51.35171101982838,5.460881343327164 51.3518381890709,5.4608676253727255 51.351841487637756,5.46078970262002 51.351714319585845))"
    },
    {
      "@id": "nsid:paal/_A2F3132CD221F50D8E0532B0B5B0A22E9",
      "geold:geometry": "POINT(5.4608242216248915 51.35170268429771)"
    }
  ]
}
```

Figure 8: Only 1 geomtry type is given

GeoJSON

Looking at the Geo-JSON response of the geo-API we saw a good response which is immediately useable by Leaflet. The major difference we saw with the JSON-LD is in the *@id*, here a full link was given. We encountered two problems using this type of id.

Firstly, during our experiments, we noticed that this type of id is not very linkable from other websites. When a domain name changes, people linking to this data, will not want to save geonovum.triply.cc in every object if it can also be saved only once (like in the JSON-LD format). Spotzi recommends using the Geo-JSON format only when directly plotting on a map, without further saves.

Secondly, we expected to be able to use this link for retrieving the full object, this was not the case. Because of the way the Triply-API is build, we had to use the graph end point and prepend the link with <http://geonovum.triply.cc/graph?subject=>. This doesn't make a difference in use of the API, developers will just have to prepend every object.

However, if you take into account that people are going to make links to a lot of different data-sources, including this one. They will want to make sure the sources and its objects exists, possibly by performing an on time regulated check. These checks cannot be made to support each and every API. So to go for a well-defined standard, Spotzi recommends these ids should only be made out of existing links.

```
{
  "features": [
    {
      "geometry": {
        "coordinates": [
          5.4608242216248915,
          51.35170268429771
        ],
        "type": "Point"
      },
      "properties": {
        "@id": "http://geonovum.triply.cc/id/paal/_A2F3132CD221F50D8E0532B0B5B0A22E9"
      },
      "type": "Feature"
    }
  ],
  "type": "FeatureCollection"
}
```

Figure 8: GeoJSON



Findable by crawlers

The last section of the data testing chapter is about findability by crawlers. It is very important that data providers create a portal that is usable by web developers, but firstly they need to find the portal. The lessons learned state some points that can be used to make your platform findable by crawlers. And thus findable by web developers who search for data with searching machines like Google or Bing.

XML sitemap

One of the lessons learned states that the use of an XML sitemap is advised. By using an XML sitemap, web crawlers like Google and Bing can easily see what content you provide on your platform and a data provider can point the web crawlers to new or updated content. At this moment the data platform provided by Triply does not provide an XML sitemap.

Persistent URIs

During the project, the data providing platform doesn't use persistent URIs for its data. There are for example references to the local directory from the Triply server and nonexistent links are used. For example, <http://lodlaundromat.org/data/c39e1092fd8387233e60222952f11a2a>. We advise data providers to either create persistent URI's if they are the owner or authorized administrator of the dataset or to use persistent URI's from the data authority. This does not only prevent duplication and errors, but also offers trust to possible data users. By not using persistent URI's, websites and applications of users can crash if the URI changes.

Links

In the vocabulary graph on the Triply data platform there are links to formal definitions of the BGT. We found some working links to e.g.

http://definitives.geostandaarden.nl/concepten/imgeo/doc/begrip/Administratief_gebied

However, most links we found in the BGT data are all nonexistent e.g.

<http://geonovum.triply.cc/def#Bak>. We expect that this is a temporary substitute of the definitive link.

Data providers should follow the lessons learned and should offer working and authorized links to prevent problems with the web applications and to offer trust to their users.

Web application

We've created a simple web application on <http://geonovum.spotzi.com/task2>, see figure 9. The web application provides a way to test the Triply-API in a real-time environment we created. The main question we try to answer is: *How to find data by specifying a location?* To test this, we have implemented though of two different scenario's.

In the first scenario the user can search by entering his/her own *search word*. This search word had to have some connection to a geographical location. For instance, a zip code, as part of a form the user has to complete, for retrieving information about the surroundings.

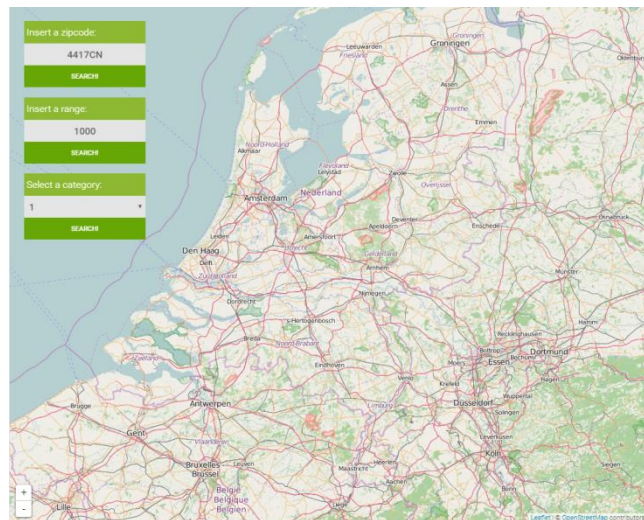


Figure 9: web application

In the second scenario the user can search data by specifying a search area on an interactive map. Using the user specified search area, the server searches for data that intersects its boundaries. This requires some geospatial functionality to be done on the server.

Category

In our web application we use different datasets provided by Triply, namely the Cultural Heritage data and the BGT. The web application offers both scenario's the possibility to select the datasets they want to use. Our goal is to expand this category column with dozens of datasets from different sources.

Zip code search

The first function provides a way for the user to find cultural heritage data by entering a Dutch zip code (e.g. 9999ZZ). Once submitted, the zip code will be located through a Spotzi-service and the location will be used to retrieve data. Which will be send back to the browser to plot on the map.

Area search

The second function provides a way for the user to specify an area to find data. The user selects a range and clicks on the map to plot a circle. The user then has the option to submit the area to the server or drag the circle to overlap a different location. Once the user is satisfied about the location s/he selected and submits, the server will then take the input and look for data on the Triply-API to retrieve back.

Connection to the data

This section focuses on creating a connection to the Cultural Heritage data provided by Triply. Answers will be provided on the questions: Is the Cultural Heritage data clear for developers? How, and can we query and visualize the data? Delivers the applications a good user experience? We describe the problems we encountered as these problems could prevent other web developers from using a data providing platform.

Usability

Firstly, we browsed the monument data through the graph endpoint with help of different response-formats. Here the data could be browsed/filtered on base of 'normal' non-geographic data. There was no notice of the availability of response formats within the graph page on the browser, the only accessible format was through HTML. According to *10 Heuristics for User Interface Design*⁹ of Jakob Nielsen, this is a usability issue on the point *Recognition rather than recall*. Which tells "*Instructions for use of the system should be visible or easily retrievable whenever appropriate*". Next to that, the following issues came up:

- **Clicking on Next within the /graph page would lead to the home page**
This was fixed by setting the page attribute in the request-url
- **Responses on erroneous requests were not clear**
Requesting too many items per page or an out of range page number lead to a 500 error
- **Links in data incorrect**
Link to <http://data.cultureelerfgoed.nl:3333/1000001> leads to an unavailable website

Secondly there was a geo endpoint, here the data could be retrieved on base of location. Through the browser this could be done by clicking on a map. Then the browser would make a request and plot the retrieved objects in the response. Also here missed the link to the different response formats which will be used by computers. Thirdly there was an SPARQL endpoint, for this endpoint there was no documentation/example available. The documentation for this could be found in the doc endpoint of the Triply-website. Here the documentation on the graph and geo endpoint could be found. Documentation on the SPARQL-endpoint was still missing.

⁹ <https://www.nngroup.com/articles/ten-usability-heuristics/>



Adding the data to the platform (without storage)

In our web application we use a well-known JavaScript library for interactive maps, namely Leaflet¹⁰. For this project we were asked to directly connect to the data from the Triply platform. Normally Spotzi saved and/or caches all the data instead, so this was something new for our developers.

As you can see in the section “Data testing”, our developers quickly were able to request the right data and show it in a platform. The usage of the lessons learned helped a lot here. For example, the lesson that describes to offer data in different flavors. The data provider offered its data in GeoJSON, which was known by our developers and could be used as direct input for Leaflet. This ensured that our developers did not need to make extra transformations e.g. and could make quick progress.

One of the main things our developers missed was something we call the 2nd degree conditions. The 1st degree conditions to make publishing spatial data work are the format of the data and the rules applicable to this. The 2nd degree conditions are conditions like delivering extra data to the users on how to work with the platform. Although our developers could quickly add data to our platform, the question remained: “Are we adding the data in the right way?”.

In particular during the steps described in section “Other connections” were steps where our developers could use some documentation. As linked data was a brand new topic for some of our developers, we did not know how to use the data in the most optimal way. With the use of feedback and a lot of web searching for answers, we finally were able to add the data directly to our application, and link data from other sources to it.

¹⁰ <http://leafletjs.com/>



Application performance

Another point we've encountered which was not mentioned in the lessons learned was the application performance. When our developers were developing our application, the developers of Triply worked on their platform. Because the platform was still in beta, the uptime was not that good. This resulted in our developers who stopped working with the platform for a while, because they could not test their application.

In practice, web developers and other users will stop using a platform if it cannot guarantee a good performance and uptime. Users who try to develop an application with this data cannot offer their application to third parties, because they need it to be reliable.

We've decided to also add some performance tests to our research, of which the results are presented below. The application performance decreased as the page number increased, but the platform did not crash.

Graph endpoint performance

Page number	Load time (ms)
1	198
5	361
25	383
50	368
100	418
250	706
500	972
5.000	3.170
50.000	31.380



Other connections

In this section we are going to expand the application by adding other datasets from different data sources. For this we decided to use the BGT-dataset in combination with the Cultural Heritage dataset. Both published at geonovum.triply.cc. Before we wanted to link these two dataset we decided to test getting geo-objects from the BGT, including only the object ID's, and link it with the same Geo-object on www.ldproxy.net.

Linking the BGT

To test the possibility of linking data we decided to retrieve data from the BGT from different sources. This gave us an easy step-in to linking the data, we though. However, while trying to find the same object from the first site in the second website we encountered problems.

Firstly, we tried to identify the object on both websites using the ID. Both the websites used */object-type/_object-id* (e.g. `nsid:Pand/_A2F312D00F0CC4CE6E0532B0B5B0AD9B0`) as part of its ID.

However, ID's included the website. One object-id became geonovum.triply.cc/id/pand/_A2F312D00F0CC4CE6E0532B0B5B0AD9B0. As developer it was easy to go from there to `pand/_A2F312D00F0CC4CE6E0532B0B5B0AD9B0`, since we could expect geonovum.triply.cc/id to be always part of the ID. Trying to look up the same object at Ldproxy, we came with the link www.ldproxy.net/bgt/pand/_A2F312D00F0CC4CE6E0532B0B5B0AD9B0, as this followed the URI-strategy of Ldproxy. However, this link did not work. After some speculation, we found out that Ldproxy uses a case sensitive lookup. So, changing 'pand' to 'Pand' did the trick. This lead us to the following question: *Should case sensitivity play a role in object-id's (URI's)?* If this is true, what will be the standards and how should developers handle these ID's.



Evaluation

As stated in the introduction Spotzi focused on the data user side of the main research question. Regarding the lessons learned we focused on answering the questions listed below:

1. Evaluate the usability of the lessons learned. How useful is the lessons learned as documentation? Are the contents understandable, the examples workable and the form of the documentation useful?
2. Report any errors found in the lessons learned.
3. Identify any gaps in the lessons learned, i.e. aspects of spatial data publishing on the web about which guidance is needed, but missing in the lessons learned.
4. Identify any bad advice in the lessons learned. Are any parts of the lessons learned arguably not good practice at all?
5. Give suggestions, preferably as text proposals, for the improvement of the lessons learned.

In this section we sum up the lessons learned from the previous research topics and evaluate them in an attempt to answer the main research from the view of the data user:

How do these lessons learned meet the constraints (e.g. budgets) and capabilities (e.g. in-house know-how) of governmental organizations on the one hand, and of data users on the other?

Lessons learned

Lesson 1: Everyone in a platform or community has their own needs and capacities

Lesson 1A: Make sure the needle can be found in the haystack

Lesson 1B: Keep it simple

Lesson 1C: Think carefully about who is allowed to do what

Lesson 1D: Each speaks its own language and lives in his own world

Evaluation lessons 1

Based on the experience of making the linked web application, when (a piece of) data leaves the platform, the definition of the retrieved data is changed. This is because the data is not being viewed as part of a whole dataset, but as the data that is going to help the end user. As data publisher you cannot know why the user is in need of some part of the data, or even which exact part of it is needed. Stating the intended outcome of this lesson learned "*Clear language, unambiguous terms, understandable interface, proper use of the data*", this lesson learned should include the following:

Options to filter the data should be clear to a developer. Even though the meaning of a dataset can be clear, it can still be a lot of work to sort out what are its possibilities. Metadata



should not only explain what the meaning is of a field, it should include what form the data can take. For example, when data is categorized, a developer would like to know which categories are there. Or in the case of zip codes in the Netherlands, the format is always four digits and two letters. When this information is available head on, developers will know what to expect, run into fewer errors and get what they want sooner.

Besides the data must be published with developers in mind, there should also be focus on different ways of retrieving the geospatial data. The user's needs in this case are for example points of the BAG instead of the full polygon BAG. We will continue this discussion in the Evaluation of lessons 3.

Lesson 2: Make search engines feel comfortable to discover you

Lesson 2A: Show a search engine the direction with an XML sitemap

Lesson 2B: Foster to link everything with everything

Lesson 2C: Think of the future, use persistent URIs

Lesson 2D: Make use of the structure, include Schema.org markup

Evaluation lessons 2

Having a site map is clear, but it should not only focus on crawlers but also on developer applications (e.g. an application should be able to learn what a dataset includes). In other words, the website should not limit itself with a site map to describe which datasets it contains (we are missing the application part, focus is primarily on crawlers). Finding is good, but understandability is even more important.

Furthermore, the sitemap itself should link to metadata and it should be clear where to begin (e.g. every page link to a dataset should include a link with metadata). This may rise to a discussion about how to go from sitemap to data map.

In the content of lesson 2B we miss a check on the data, linking to another website is one thing, keeping your data updated a correct is very important. Looking back at the paragraph *content-types*, where the object-link of a Geo-JSON was not a persist URI, we suggest making links only with working URLs. This makes it easier for 3th parties to check their data-references. We suggest a standardization of using links as id's.



Lesson 3: Deal with the unknown set of developers and devices

Lesson 3A: Serve your data in many different flavors

Lesson 3B: Improve performance, reduce payload

Evaluation lessons 3

As mentioned in the evaluation of lesson 1 user's needs may differ and *operations to filter data should be clear to a developer*, for example points of the BAG instead of the full polygon BAG. As mentioned in lesson 3B Spatial datasets are often highly accurate, highly detailed and are quite large in number of objects and megabytes. All the advice given in this lesson is very useful.

However, the sentence *"High Definition (HD) data is not always necessary. Performance counts for developers. So: downgrade your data, reduce the payload and improve performance"* – in Lesson 3B may rise to confusion. In our opinion this sentence should be reformulated. Downgrading your data does not always result in 'lower' high definition data. The data is as good as you want the data to be, this is different per (data)user.

Perception is a very important thing, especially in the geospatial world, where e.g. a color choose can create a whole other map with a whole new meaning.

As mentioned above downgrading your data does not always result in 'lower' high definition data, it all depends on the application. Another way of reducing the payload for example the BAG can be e.g. taking the centroids instead of the full polygon. This is basically a geospatial transformation from polygons to points. This function is not the same as simplification as mentioned in lesson 3B.

We advise to extent the list of possible approaches to get better performance with transformation. Users must know the existence of this kind of functions.

This way transforming high definition data from polygon to points will still be high definition data if the user has all the information he or she wants.



Lesson 4: Don't copy data, use proxy

Evaluation lessons 4

The results of the last testbed pointed out that the current SDI did not provide sufficient service for unknown developers and devices. To make it more suitable, the lessons learned recommend to use proxy's. Although the data providing platform in this research was not a proxy server, but a copy of the data, we still have comments on this.

Spotzi agrees that using a proxy does deliver extra benefits to connect to the group of unknown developers and devices. However, a proxy also means an extra step between the end user and the data. An extra step means an extra vulnerability, especially when it comes to performance and uptime.

As stated before in this report, Spotzi advises to add an extra lesson learned about performance and uptime. This however, is also very important when creating proxies. The proxy site should offer the same performance (or better) as the server of the authority. If one of the parts in this chain fails, no end user will use the spatial data.

New lessons:

Performance & uptime

The only way a spatial data platform will work is if it performs well. What this performance should be, relates to the nature of the data set. Not every dataset needs to have an uptime of 100% and query speeds below 10ms. However, the authority of the dataset and the publisher of the spatial data, should be aware that developers will not use their data, if they cannot offer a reliable service.

Versioning

During our test phase, the data delivery platform worked with different versions. When they were making changes to the production version, the beta version could be used. However, there were big differences between these versions. This resulted into crashes in our application. If a data provider works with versioning or makes updates to his platform, he must make sure the output to the users does not change.

2° degree conditions

Next to the 1st degree conditions, like the data format, content and rules, 2nd degree condition are very important. A data provider must make sure he offers sufficient documentation on his platform to help developers work with their data. This is not only in the interest of the developers, but also of the data provider himself. For example, if a data provider does not document the limitations of his platform, web developers will go search for the limits themselves, which could result in high payload for the data platform.