PUBLICAMUNDI

SCALABLE
REUSABLE
OPEN
GEOSPATIAL
DATA

Yannis Kouvaras

Data API

Data API – Introduction I

- JavaScript client API for querying catalogue's vector data
- Query both local and remote data
  - **CKAN DataStore local data e.g. CSV, spreadsheet data stored in PostGIS**
  - **Remote data sources e.g. WFS, remote data files, etc**

Data API – Introduction 2

- Access resource metadata from the catalogue
  - **Resolve resource unique Ids**
- Support for most common operations
  - **Projection with support for computed fields**
  - **Filtering with support for spatial operations**
  - **Join multiple data sources**
  - **Sorting (wherever applicable)**

Data API – Design Concerns

- Implementation
  - **How to implement query execution at the server side?**
- Programming Interface
  - **How the user interacts with the query service?**
- Query Language Syntax
  - **How queries are formatted?**

Data API – Implementation I

- Implement Data API as a standalone service
  - ✔ **No external dependencies**
  - ✔ **Allows for better optimization**
  - ✖ **Building query execution plans for handling remote data requires more effort**
  - ✖ **Managing asynchronous processing becomes more tedious**

Data API – Implementation 2

- Implement data API as a WPS operation
  - ✔ **Can handle remote data sources more efficiently and transparently**
  - ✔ **Natively supports asynchronous queries with progress callbacks**
  - ✖ **Requires a dependency on a WPS service (PublicaMundi already has a requirement for the ZOO project)**

Data API – Programming Interface I

- Fluent API: Building queries by chaining method calls
  - **The final result represents a query tree**
  - **Execution is deferred until the user requests data**

```
var query =
   PM.Dataset('cities','c').Select(['c.id','c.name','c.geom']);

query.toJSON({ … });        // Triggers query execution

query = query.Filter('c.name', PM.Operations.EQUAL, 'Athens');

query.toWFS({ … });         // Triggers query execution
```

Data API – Implementation 2

- Client side service proxy
  - **Building queries using a custom Query Language**
  - **Less method calls**
  - **Allows for higher flexibility**

```
client.getJSON('from cities c select c.name, c.geom',{ … });

client.getWFS('from cities c where c.name=\'Athens\' select
  c.name',{ … });
```

Data API – Query Language I-I

- SQL-like API based on a subset of PostGIS and PostgreSQL functionality

```
select p.id, p.name, p.type, ST_Buffer(p.geom, 5)
from point_of_interest p
where ST_Distance ( ST_GeomFromText ( 'POINT(-72.1235 42.3521)', 4326), p.
    geom) < 100
```

Data API – Query Language I-2

```
select p.id, p.name, p.[type], ST_Buffer(p.geom, 5), c.name
from   point_of_interest p join cities c
where ST_Intersects(c.geom, p.geom) and p.type = 3
```

Data API – Query Language 1-3

- ✔ Extremely powerful for expressing complex queries and performing data analysis
- ✔ Utilizes developer existing experience with PostGIS
- ✔ Friendly syntax for the average developer with basic knowledge of SQL
- ✖ All custom functions and operations should be declared and implemented
- ✖ Parsing queries and building execution plans is more difficult

Data API – Query Language 2-I

- SQL like API inspired by LINQ

```
from p in point_of_interest
where distance(point(-72.1235,42.3521, 4326), p.geom) < 100
select    p.id, p.name, p.type, buffer(p.geom, 5)

from p in point_of_interest
from c in cities
where    intersects(c.geom, p.geom) and p.type = 3
select    p.id, p.name, p.type, buffer(p.geom, 5), c.name
```

Data API – Query Language 2-2

- ✔ More intuitive from the previous syntax style
- ✖ Same drawbacks as the Query Language 1 syntax

Data API – Query Language 3-I

- Fluent API with deferred query execution

```
var query = PM.Dataset('point_of_intested', 'p').
                Dataset('cities', 'c').
                Filter('p.geom', Const.INTERSECTS, 'c.geom').
                Filter('p.geom', Const.EQUAL, 3).
                Select(['p.id', 'p.name', 'p.type',
  'buffer(p.geom, 5)']);


query.toJSON({ … });
```

Data API – Query Language 3-2

- ✔ Simple programmatic API gives more control to the client
- ✔ Abstracts query tree representation. Query tree internal representation can be switched at any given time
- ✔ Allows for incremental query building
- ✖ Less expressive especially if we keep the method arguments simple

Data API – Query Language 4-1

- Expressing queries using comprehensions (inspired by https://www.mingle.io/)

[{p.id, p.name, p.type, **buffer**(p.geom, 5)} | p <~ points_of_interest,

distance(**point**(-72.1235,42.3521, 4326), p.geom) < 100]

[{p.id, p.name, p.type, **buffer**(p.geom, 5)} | p <~ points_of_interest, x <~ cities,

intersects(p.geom, c.geom), p.type = 3]

Data API – Query Language 4-2

- ✔ Extremely simple API for fast query composition
- ✖ Less expressive especially if we keep the method arguments simple

Data API – Query Language 5-1

- Queries expressed as JSON similar to existing CKAN DataStore API

```
var query = {
    datasource: [
        { id: 'points_of_interest', alias: 'p' }
    ],
    select: [
        'p.id', 'p.name', 'p.type',
        { method: 'buffer', parameters: ['p.geom', 5] }
    ],
    filter: {
        method: 'distance',
        parameters: [
            'p.geom',
            { 'type': 'Point', 'coordinates': [-72.1235, 42.3521], crs: 4326 },
            100
        ]
    }
};
```

Data API – Query Language 5-2

```
var query = {
    datasource: [
        { id: 'points_of_interest', alias: 'p' },
        { id: 'cities', alias: 'c' }
    ],
    select: [
        'p.id', 'p.name', 'p.type',
        { method: 'buffer', parameters: ['p.geom', 5] }
    ],
    and: [{
        method: 'equal',
        parameters: ['p.type', 3]
    }, {
        method: 'intersect',
        parameters: ['p.geom', 'c.geom']
    }]
};
```

Data API – Query Language 5-3

- ✔ Simple syntax
- ✔ Most modern editors support basic JSON formatting (JavaScript) and syntax highlighting reducing potential syntax errors
- ✔ Easy to parse and build query execution plans especially for the WPS implementation
- ✔ Easier to parse and generate an OGC:Filter
- ✖ More verbose than any other alternative

Data API – Current Status

- WPS implementation
- Fluent API
- JSON query representation

Consortium

IMIS / Athena RC (Coordinator)

Research Center (GR)– Data Economy (open, big, linked)

rasdaman

SME (DE) – Big Geospatial Data

GeoLabs

SME (FR)– Processing Services

GET

SME (GR) – Spatial Data Infrastructures

Contact us

PUBLICA MUNDI

www.publicamundi.eu