QUERYING WITH EASE TO RETRIEVE SPATIAL DATASETS THROUGH AN APPLICATION PROGRAMMING INTERFACE (API)

Ang Dawa Sherpa

Dissertation submitted in partial fulfilment of the requirements for the Degree of *Master of Science in Geospatial Technologies*

QUERYING WITH EASE TO RETRIEVE SPATIAL DATASETS THROUGH AN APPLICATION PROGRAMMING INTERFACE (API)

Dissertation supervised by:
Auriol Degbelo, PhD
Institute for Geoinformatics,
University of Münster, Münster, Germany

Co-supervised by:
Carlos Granell, PhD
Universitat Jaume I,
Castellon de la Plana, Spain

Co-supervised by:
Joel Dinis Silva, PhD
Universidade Nova de Lisboa,
Lisbon, Portugal

February 24, 2019

DECLERATION

I, Ang Dawa Sherpa, master student of the Geospatial Technology Erasmus Mundus program, am aware of my responsibility of the penal law, declare and certify with my signature that my thesis entitled

“Querying with ease to retrieve spatial datasets through an Application Programming Interface (API)”

is entirely the result of my own work I have faithfully and accurately cited all my sources, including books, journals, handouts, and unpublished manuscripts, as well as any other media, such as the Internet, letters or significant personal communication.

I understand that

- literal citing without using quotation marks and marking the references
- citing the contents of work without marking the references
- using the thoughts of somebody else whose work was published, as of our own thoughts

are counted as plagiarism.

I declare that I understood the concept of plagiarism and I acknowledge that my thesis will be rejected in case of plagiarism.

Münster, February 24, 2019

ACKNOWLEDGMENTS

First and foremost, I am duly grateful to Erasmus Mundus Program for granting me with the scholarship to pursue my master degree. I would like to express my deepest gratitude to my main supervisor **Dr. Auriol Degbelo** for continuous encouragement and criticism in inspiring to work to the fullest level with proper guidance and support throughout the thesis period. His willingness to give his time and proper guidance was always motivating.

Additionally, I would like to extend my sincere gratitude to my co-supervisors **Dr. Carlos Granell** and **Dr. Joel Dinis Silva** for their encouragement, guidance, constructive suggestions and criticisms. I am highly obliged in taking the opportunity to sincerely thank all my professors from Ifgi, UJI, and Nova IMS who bestowed me with multidisciplinary knowledge and skills and helped to develop professionalism in me.

I would also like to extend my thanks to all the administrative staffs from UJI and Ifgi for their help in offering me proper guidance and timely information during the entire master program. Finally, I wish to thank my friends and family for their encouragement, unconditional love, and support throughout my study.

QUERYING WITH EASE TO RETRIEVE SPATIAL DATASETS THROUGH AN APPLICATION PROGRAMMING INTERFACE (API)

ABSTRACT

In the context of searching for Open datasets, there is a still a limited understanding of the efficient ways to access them. Using an Application Programming Interface (API) is one of the ways that developers mostly use. However, there is still a need for refined APIs with greater learnability in the context of accessing Open data. In this work, we developed a RESTful API with a capability to retrieve spatial datasets easily from API level by querying in terms of parameters, namely Space, Time, and Theme. To achieve that we made two assumptions. Firstly, expressing the temporal parameter in a natural language (e.g. today, last year, etc) benefits the developers in many ways as it avoids having to deal with rigid date format. Secondly, if the API can accept parameters in any order, it saves the developers time as they don't have to spend extra time learning about the parameters order and they can simply focus on the main task. A user-study with 20 participants was done to evaluate the API in terms of its learnability. Half of the participants had prior experience of querying datasets from API level and remaining did not. The evaluation result was promising as both categories of participants found the API easily learnable. This means such API has the potential to restrain the problem of developers compulsion to spend a lot of time before becoming really productive. Likewise, it also had the potential to pave a way towards the next generation API as it addresses the problem of lacking refined APIs to access Open Government data easily.

Keywords: API, learnability, Open Government Data, Spatial data

TABLE OF CONTENTS

1 INTRODUCTION

1.1 Thesis context

Different public bodies of government generate and gather an astronomical amount of data over a period of time [44]. Realizing the importance of these data, the concept of Open Government Data (OGD), has been moving forward at a tremendous rate as all over the world people are becoming more aware towards the topics like participation, collaboration and transparency [13]. Here, OGD stands for a philosophy or a policy that advocates above-mentioned topics by giving access to government data to all those interested citizens [44]. The main reason for implementing such a policy by the government is an assumption that society will produce activities that have both social and economic value by using public data [39]. Data can be anything from traffic-data, scientific publications, medical studies, geo-data, and statistics, etc [13]. Along with the advancement of technology like an Internet of things (IoT), hundreds of authorities (both private and public) are able to produce large data every single day. One of the major challenges has been the way of accessing to the flood of data that is out there [13]. Because of its voluminous state, sometimes it will be difficult to continuously preserve the expected quality and assure easy access to these data [11].

As a means to increase technical interoperability among various accessibility options such as information platforms, information services, portals, and tools, Application Programming Interfaces (API) should be proffered [13]. APIs in general play a great role in easing up the process of making the data easily accessible to the public as APIs are capable of bridging different devices by accessing their core system [8]. Not only it allows the user to upload the data but also curb an error when correcting data and metadata manually. Currently, more than 80% of EU28+ national Open Data portals host the data accessing service through API [8]. For instance, Estonia's 'API first' policy highlights the necessity of providing access to their Open Data through APIs. The policy emphasizes the importance of machine-to-machine communication for increasing efficiency in-service distribution [8].

The main advantage of correct delivery of a service by an API is that the developers can easily reuse the same service and accordingly design varieties of customized app based on users' demand [43]. However, despite delivering the service correctly, a bad design could obstruct the developers from making the most out of the services that API provides [4]. For an example, badly designed APIs demand lots of time to learn about that API which hinders the main motivation behind APIs i.e. improving programmer's productivity by enabling code reuse instead of code writing from scratch [5].

There is a still a limited understanding regarding the ways to enable efficient access and searching for open datasets. As pointed by different studies like [3] and [4], because of lacking standards that are more universal, the problem of writing numerous APIs for each city and maintaining them individually still persists. The Open Data Barometer (The World Wide Web, 2015) notified that open data is rapidly becoming part of the mainstream, nonetheless, there is still a lack of refined APIs that could easily retrieve government data [2]. Often times solutions are rather stand-alone, that is supposed to solve just one specific barrier developers have to overcome and as a result, hundreds of API exist [8].

1.2 Research gap

Different studies focusing on improving access to Open Government Data (OGD) have been carried out in the past. The authors used various methodologies for their studies. [40], [41] and [42] used linked data principles. Similarly, [2] proposed semantic APIs as a way of improving access to open data.

However, none of the authors carried out the study by integrating spatial, temporal, and thematic aspects together in their methodologies used such as SPARQL, API, etc. They have emphasized their attention mostly towards the thematic dimension.

With the increasing amount of temporal and spatial data occurring on the web, this thesis is focused on developing an easily learnable API in order to access OGD taking Time, Space, and Theme into account.

1.3 Research objectives

The research question that propels this study is: How to retrieve spatial datasets with a user-friendly query in terms of Space, Time, and Theme?

The presumption is that finding an answer to this question is through the designing and testing of a method for retrieving datasets with a user-friendly approach on an API level.

1.4 Innovation

This study tries to resolve the two key issues. First is to see the possibility of curbing the problem i.e. the developers having to spend a lot of time to learn about the API before becoming productive. Second is to address the problem of lacking refined APIs to access government data by checking if the approach taken in this study means one step closer towards the next generation API.

This study designs a user-friendly RESTful API that allows the users to retrieve spatial datasets with the user-friendly way by querying the datasets in terms of parameters namely Space, Theme, and Time. Here, Space refers to the spatial attributes of the datasets. The theme refers to a thematic description of the datasets. Time indicates the temporal aspect of the datasets.

1.5 Assumptions

This study is based on the following assumptions:

a. Following steps increases the learnability of the API:
   i. Expressing the query parameters with natural language.
      Eg: Today, yesterday, last month, etc for expressing time.
      City name rather than geo-coordinates for expressing space.
   ii. Order in which the query parameters are entered should not matter.
      Eg: api/crime/bonn should work in the same manner as api/bonn/crime

1.6 General method

The method used in this work was comprised of the following steps:

- Carry out an extensive review of related literature.
- List out the sample of potential general questions in terms of Space, Time and Theme that the users might be interested to query the datasets about.
- Filter out only those questions from a poll of questions listed in step 2 to which this API can provide an answer.
- Design and develop an API to query datasets in terms of Space, Time and Theme. The design has followed existing guidelines for API design by [10].
- Develop a web-app integrating both front-end and back-end for an evaluation purpose.
- Conduct a user study for the evaluation of newly developed API especially focusing on its learnability.

1.7 Thesis Organization

The thesis is divided into six main chapters. Chapter 2 reviews the related work on sub-topics such as API design, API usability, and API Design Guidelines. Chapter 3 elaborates on the Design of the API which will mainly include the questions this API answers, design guidelines that were used while designing this API, an architecture of involved technology and also the presentation of the API. Chapter 4 deals with the evaluation carried out for this study. Chapter 5 includes result and discussion. Chapter 6 deals with the conclusion of the thesis.

2 RELATED WORK

2.1 API design

The three significant stakeholders of an API are API designers (whose major objective is to maximize the adoption of an API and minimize its support costs), API users (developers with an intent to write error-free applications), and consumers of products based on the built API [15]. This section is most relevant to API design.

All the disciplines such as algorithms, databases, programming languages, and operating systems would be useless without suitable APIs [18]. Most significantly, the modern software relies on robust, re-usable, and efficient APIs for its core and secondary functionalities [18] which means in today's world, almost every device has some kind of API running within them. Similarly, one of the fundamental entities of the World Wide Web (www) is also API [2]. In other words, it enables the web-based system and mobile-applications to communicate between a myriad of services and bring them together. For an instance, incorporation of Google Maps in location-aware applications by using Google Maps API with just a few lines of codes [19].

The main objective behind making use the same API across multiple platforms means eliminating the necessity of rewriting the code from scratch thus amplifying the programmers' productivity [15]. In the context of API designing, it is a paramount step to keep in mind that enabling data access to machines and enhancing the developers productivity must go hand to hand. These APIs fetch data stored externally and make a system complete to provide the service to its users [16], [17]. However, it is the developers who instruct these hyper-connected devices regarding the fundamental questions like how to fetch the data and which API to use [2].

Functionalities are developed by programming every line of code in the context of API development, nonetheless, programming is not the most significant aspect of API development [16]. Rather, the API including its development process should be considered as a user interface [16]. Therefore, the context of designing an API narrows down to two important steps machine focused and developers focused. Firstly, issuing an efficient interface with which devices can approach resources such as datasets. Secondly issuing suitable interface that allows the developers to approach resources in order to develop an application [17].

API design thus can be considered as “human-machine-human” conversation problem especially when the process of information sharing is involved [17]. The main specificity of API design is “stakes of getting the design right in the first place” [15]. One of the tricky things about the API is that once they are deployed for service then after that even a fractional change in their interface results in hundreds of defunct apps, ultimately leading to loss of time and financial resources [19], [28]. RESTful is one of the prominent architectures in the world of API design.

As per the [35] the main basis of RESTful-style are Uniform Interface, Stateless, Cacheable, Client-Server, Layered System, and Code on Demand (optional).

A RESTful API represents a set of resources along with various operations that can be executed on those resources [14]. The execution of those operations is possible through any HTTP client inclusive of web browser's client-side JavaScript [20].

Just like a context root in the local file directory, the REST API has a base-path. Beside this they also have methods such as URL and media types. This base-path acts as a point of reference for all resources in a REST API [14]. Likewise, the base-path can also be used to separate different entity and versions exposing and manipulating through different HTTP methods as a route between different REST APIs. Different HTTP methods are GET, POST, DELETE, PUT, and PATCH. For an instance, a GET Method of REST API to expose a database of hospital patients would be /patients/v2, similarly, the base path for the third version of same REST API could be /patients/v3.

To design efficient RESTful API as the first step semantic descriptors should be listed out [20]. Here, the semantic descriptors refer to that information which the API users are expecting to get as a response of the request or want to put into the API [2]. With respect to various application cases, these semantic descriptors can be grouped together and structured into hierarchies [16].

2.2 API usability

In the core, the two most fundamental qualities of an API are its power and usability [22]. The demands of APIs usability is increasing tremendously than ever before [26]. The usability includes i) learnability (how easy is it to learn) ii) efficiency (how productive the programmers can be even with little effort) iii) good error-handling iv) simpleness v) consistency vi) suitability with the users mental model. Similarly, the power is comprised of i) cost (how expensive it is to use) ii) its extensibility (how flexible it is to customize to meet the users' specific needs) iii) its evolvability (how easy is it for API designers to create the new version of API) iv) performance (how resource efficient is it regarding the memory, speed, and energy consumption) v) security, robustness and integrity of the API implementation and the resulting application.

Much studies previously done uncovered that developers while developing client code often struggle with the usability problems of APIs they use [27]. These kinds of usability problems create unwanted difficulties in the client code and ultimately diminishes the developers productivity [25]. Therefore, API usability has a huge effect on API adoption i.e. if the time that is required to learn the API is too long then developers or companies will choose different API or even write from scratch a functionality that provides similar service [22].

Realistically, since, a large number of applications depends on the same API, unlike traditional software tests, intensive API usability tests may be not always feasible. Thus, the design phase is the most suitable to consider the API usability concerns [24]. Therefore, both the developers and API designers should have a detailed understanding of API usability and use them during the designing and developing phases. This as a consequence will reduce the maintenance complexities that might arise because of the usability issues related to such APIs [25].

One of the factors that can affect the usability is the need of making many decisions at various levels while designing an API [15]. The range of this decision could go from global issues like the overall architecture of the API down to low-level issues like the peculiar name of the exported class, method, exception, and parameter. This means that the scope which needs to be considered during usability measurement is really huge i.e. it is often time and resource consuming and require experienced evaluators [24]. However, currently, lots of Human-Computer Interaction (HCI) techniques namely surveys, field-observations, etc. are being used with an objective to identify API usability factors and suggest guidelines for enhanced usability [29].

The space of competing for design decisions and API quality aspects were mapped by [15]. [30] suggested the 12 cognitive dimensions for explaining and measuring API usability. Similarly, the seven measures for profiling usability of APIs were recommended by [31]. The matrices about how to resolve the domain concepts that are inwardly expressed within the API in order to assist the API developer was suggested by [32]. Though these are primly applied to Graphic User Interfaces, they have gone through certain alteration to reflect the more in-depth of situations experienced by developers, and eventually address the specific challenges of evaluating API designs [23]. Likewise, in the year 2009, http://www.apiusability.org was created to disseminate comprehensive resources on API usability [25].

2.3 API design guidelines

In the world of API designing, it is significant to investigate the extent to which the guidelines in the sets are consistent with each other, and whether or not there are universal best practices [33]. Most of the research about usability and APIs have given more emphasis to API user's experience, software patterns and documentation with an analogy with UX and DevX for User Experience and Developer experience respectively. However, there has been only a few studies on the way API Designers work and their needs [33].

An API designer uses URI/URL to indicate the hierarchy of resources and identifiers. In other words, URI/URL can give a clue regarding the relationships among identifiers and resources thought from the perspective of API guideline authors [32]. Some API guidelines such as (REST cheat sheet, Geert Jansen, Australian Taxation Office) advise not to stop the nesting after one sub-resource (resource/identifier/sub-resource/sub-resource). Whereas, GoCardless demote the nesting concept and rather suggests the application of filters to separate identifiers from resources. They claim that “Nested resources enforce relationships that could change and make clients harder to write” [34].

Nonetheless, a handful of guideline sets issue a rationale for constructing the structure of the resources and identifiers relationship. Some guideline sets just state the plain syntax for implementation of URI/URL structure [32]. Similarly, fractional of them tries to answer whether a resource can function as sub-resource or not. However, in the broader picture guideline sets do not often emphasize on low-level application part of the URI/URL structure [31].

Most importantly, all these guidelines emphasize the simplicity and intuitiveness while designing API [33]. Not nesting a sub-resource more than once seems to be a consistent rule for path segments amidst most of the guidelines. This results in simple and short URL with an advantage for encouraging intuitiveness as it will be convenient for users to draw a mental model of the hierarchy of resources for better and quick learnability of API [34].

2.4 API and City development

In today's world, most of the development activities are dominated by technology, a context of City development is not an exception. In every step from design to management of the city, an IT solution has been extensively used [7]. It has been pushed to the extent that the quality of embedded IT technology has been one of the key components determining the overall standard of the city [13].

One of the best ways IT solutions supports the cities’ everyday demand is through its digital interfaces such as Application Programming Interface (API). In the core, API is a tool and communication protocols, a set of subroutine definitions that provides an interface for a developer to create a new application. An API bridges the communication between different computer components i.e. backend IT systems, code modules, and applications. In other words, API determines the way for making the possibility for interaction and sharing of data among these components [38]. This is also the reason why it can be used in different forms, for an instance object classes, remote calls, specifications for routines and data structure [13]. Having such capabilities, APIs have opened a myriad of ways to exchange the ideas and make the cities’ digital interface more harmonic [7]. As a result, it has helped to set up the connection between multiple cities and possibilities of development grows exponentially for all the involved parties [8].

3 METHODOLOGY

3.1 Questions that are answered by API designed for this study

Parameters details:

The main focus of this API was to come up with an easily learnable and a user-friendly way of querying the datasets in terms of parameters namely SPACE, THEME, AND TIME from API level. Brief detail about each parameter is as follows:

- **SPACE**

  Space refers to spatial attributes (represented through the name of the city for this study). Currently, for this study datasets are available for these cities only:

  1. Bonn
  2. Dortmund
  3. Dusseldorf
  4. Dresden
  5. Kathmandu
  6. Münster
  7. Stuttgart

- **THEME**

  Theme refers to the resources for this RESTful API. As of now following datasets are available.

  1. Population
  2. Migration
  3. Crime
  4. Weather
  5. Transport
  6. Economy
  7. Landuse

- **TIME**

  Here the Time means the date. API accepts different semantic ways of stating date as shown below:

  Date without ranges:

  1. Today
  2. Yesterday
  3. Day before yesterday
  4. Last month

5. Last year
6. 3 November 2017 or November 3rd 2017

Date with ranges:

1. Last month to today
2. Last year to last month
3. 16 november 2013 through 18 october 2017

3.2 Steps taken to filter the questions

First step

As the first step, the sample of general potential queries was listed out for 7 different combinations of Time, Theme, and Space. Those combinations were i) Space ii) Time iii) Theme iv) Space and Time v) Space and Theme vi) Theme and Time vii) Space and Time and Theme.

- [SPACE]
  - Give me datasets about cities [space] that are within a radius of 5km from Münster city.
  - Give me dataset about the routes of the Castellon [space].
  - Give me datasets about the largest lakes in Germany [space].
  - Give me datasets about the highways that connects two cities [space].
  - Give me datasets about an urban area [space].

- [TIME]
  - Give me datasets about today [time].
  - Give me datasets about last month [time].
  - Give me datasets about last quarter [time]
  - Give me datasets about the 80s [time].

- [THEME]
  - Give me datasets about unemployment [theme].
  - Give me datasets about households [theme].
  - Give me datasets about events [theme] conducted by European space agency for students.
  - Give me datasets about shopping centers with the best quality products [theme].
  - Give me datasets about diversity among students in the Erasmus program [theme].

- [SPACE] vs [TIME]
  - Give me datasets about Münster [space] last Friday [time].
  - Give me datasets about Europe [space] from 1980 [time].
  - Give me datasets about the Arctic [space] from 1980 through 2018 [time].

    - Give me datasets about Namche Bazar, Nepal [space] from 2007 [time].

- [SPACE] vs [THEME]
    - Give me datasets about crime [theme] in Europe [space].
    - Give me datasets about places in Asia [space] that are vulnerable to Climate-change [theme].
    - Give me datasets about a highly populated city [theme] around Münster [space].
    - Give me datasets about bird species [theme] that are in Frankfurt [space] zoo.
    - Give me datasets about international students [theme] studying in France [space].
    - Give me datasets about election [theme] of Munster [space].

- [THEME] vs [TIME]
    - Give me datasets about migration [theme] between 1990 and 2018 [time].
    - Give me datasets about unemployment [theme] in the year 2014 [time].
    - Give me datasets about weather [theme] in January [time].
    - Give me datasets about land cover [theme] change over the period of the last ten years [time].
    - Give me datasets about burglaries [theme] that took place in the year 1997 [time].

- [SPACE] vs [TIME] vs [THEME]
    - Give me datasets about the events [theme] happening in next month [time] in Germany [space].
    - Give me datasets about the most committed crime [theme] in Portugal [space] in the year 2016 [time].
    - Give me datasets about population [theme] of India [space] from the time period of 2012 to 2017 [time].
    - Give me datasets about major political events [theme] of Spain [space] that took place in the year 2002 [time].
    - Give me datasets about population [theme] from last month [time] in Germany [space].

Second step

In the second step only few queries from above general questions were selected based on data availability and relevancy. Accordingly, following are the final questions this API can provide answer to.

- [THEME]
    - Give me datasets about migration.
    - Give me datasets about the weather.
    - Give me datasets about crime.

    - Give me datasets about population.
- [TIME]
    - Give me datasets about last year.
    - Give me datasets about today.

- [SPACE]
    - Give me datasets about Münster city.
    - Give me datasets about Düsseldorf.

- [SPACE] vs [THEME]
    - Give me datasets about migration that are within the radius of 1km from the specified location (long, lat).

- [TIME] vs [THEME]
    - Give me datasets about the transportation of last year.
    - Give me datasets about crimes from February 4th 2013 to 3rd July 2016.
    - Give me datasets of international tourists visiting Europe in last year and today.

- [SPACE] vs [TIME]
    - Give me datasets about Düsseldorf from last month to yesterday.
    - Give me datasets Münster land-use change from the last year to today.

- [SPACE] vs [TIME] vs [THEME]
    - Give me datasets about the crime in Münster city that took place between the 1st of October 2013 and 28th of December 2016.
    - Give me datasets about crime in Kathmandu city from last year.
    - Give me datasets about land-use of Bonn from 3rd November 2017 through today.

3.3 Design guidelines used

Most of the design part for this API was based on the book called "REST API Design Rulebook" by Mark Masse [10]. This book was chosen because the author has carefully presented the API design rules which are picked mainly from current best practices of the Web's REST architectural style.

URIs:

Firstly, in this study in order to address the resources (datasets), URI is designed to point to an object rather than _ID comprised with set of numbers and strings characters.

Eg: The API designed for this study takes the URI structure stated in (a). (b) is typical URI structure.

(a) api/Münster/crime/today
(b) api/munster/crime/4567gh0-4567jhk-ckl456

Doing this assures the benefit to both parties i.e. API designers as well as the developers. Firstly, from the designer perspective such design assures that URIs are being created in a way that convey a resource in correct manner. Similarly, from the developer perspective they have convenience in treating URIs are opaque identifiers and at a same time following the linking paradigm of the Web.

Except some customization in the query involving the radius and pair of latitude, longitude coordinate, the generic format of the URI for this API is based on format RFC 3986 [36]. That being said, specifically the structure of this RESTful API involving the radius tends to be slightly different than the RFC 3986 based generic URI structure in a way that "?" is not stated to pass the query parameter as can be seen in structure designed for this study.

URI = scheme "://" authority "/" path ["?" query] ["#" fragment] (generic)

URI = scheme "://" authority "/" path "/" [query] (structure designed for study)

A case in point: api/weather/radius=10000&long, lat=[85.2861785888672,27.7002479940336]

Here, the parameter for SPACE entity expressed as radius and (long, lat) has been separated by "&" rather than "?" as mentioned above. This still falls under the guideline of Mark Masse as he clearly states that "the core job remains the same: write code that handles HTTP-level details and translates a backend system's data model into a Web-oriented resource model. Some of this code most certainly needs to be written on a per-API basis, specifically the portion that directly communicates with the backend system or data store." The main reason for implementing this particular way for this specific query has been mentioned in the discussion section.

Secondly, another important rule implemented from the book was not using the forward slash at the end of URI as it becomes the source of confusion and brings no real meaning in URI's path. The other reason is also that numbers of web frameworks treat both URI with/without last trailing forward slash "/" equally. However, some application sensitive to URI character count might end up differentiating the above condition as two separate URI based on character count and thus pointing to different resources than desired one.

For convenience, this RESTful API was designed with preference to both lowercase letters and uppercase letters. However, in the core algorithm, the preference was given to lowercase letters following the guidelines of the RFC 3986 which states that uppercase might sometime throws an error.

For example:

(a) https://GIVEOURDATA.HEROKUAPP.COM/api/bonn/crime/yesterday
(b) https://giveourdata.herokuapp.com/api/BONN/CRIME/YESTERDAY

In the core system the API will understand both (a) and (b) from above like this: https://giveourdata.herokuapp.com/api/bonn/crime/yesterday (every word in lowercase)

Thirdly, this RESTful API does not allow to include file extensions in the URI as it is one of the paramount rules for any RESTful API. During the addition of curated datasets for this study, the file extension was relied on media type and communicated across Node environment and Postman by the Content-type header. In other words, the portal owner should be aware of the file type while uploading the data in the server such that while the API is visited by developer GET method will take care of it automatically.

3.4 Data collection

Few city datasets were collected based on their relevancy with the study context and later on curated with spatial, thematic and temporal information in their metadata. Secondly, they were stored in a non-relational database i.e. MongoDB (https://www.mongodb.com/). The underlying principle for the integrated spatial search functionality was based on Dimensionally Extended Nine-Intersection Model (DE-9IM) [12]. Particularly, inbuilt MongoDB spatial operators such as WITHIN, and maxDISTANCE were used. Similarly, the TIME-related queries were based on the Allen's Interval relations [11].

3.5 Architecture

Different technologies were brought together while developing this RESTful API from the scratch. In general, a full stack application was built which was divided as the frontend section, backend section and deployment. Backend section was solely focused on the logic that takes place on the server side. Similarly, Frontend and deployment were specifically designed for the evaluation purpose. As the first step the backend was focused. While working on backend to simulate a client making a request, Postman was largely used.

Backend:

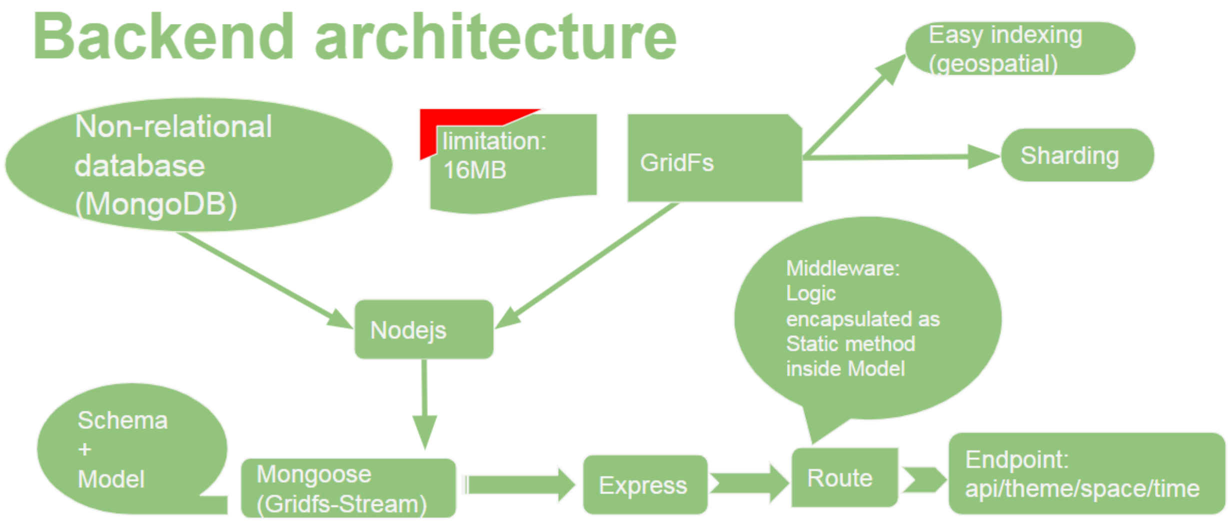

Backend architecture
Non-relational database (MongoDB)
limitation: 16MB
GridFs
Easy indexing (geospatial)
Sharding
Middleware: Logic encapsulated as Static method inside Model
Nodejs
Schema + Model
Mongoose (Gridfs-Stream)
Express
Route
Endpoint: api/theme/space/time

Figure 1: A backend architecture implemented for this API.

Backend refers to all those activities that happens in the server side without user being aware about. In other words, it is the region where all the logic rest and gets executed upon request from the client. That being said, once the client make the request through the frontend interface and pass the parameters to the server then it is in the server side where it is determined whether the query parameter belongs to TIME, THEME or SPACE.

For this study Node.js (https://nodejs.org/en/) was used as it is a JavaScript runtime environment that handles server-side. Node.js handles the designing the of API with Representational State Transfer (REST) architectural style very well. REST architecture is defined as an architectural element's abstraction within a distributed hypermedia system [9]. It is built on Google Chrome's JavaScript V8 Engine which is very powerful. Some of the reasons for choosing Node.js were as follows:

- Node as an infrastructure is very efficient in handling network applications thus making them scalable in the long run.
- Relatively, on a local machine it can be set up with much ease and server will be up and running with just few lines of codes.
- Within the Node environment, developers can build software or REST APIs and invoke external services through their APIs using plain JavaScript. This will ultimately benefit all the involved parties as the codebase will be under a single programming language.
- It is completely open source and used by thousands of developers around the world.

In Node environment, a minimalist and flexible web framework called Express (http://expressjs.com/) was extensively used. The reasons for using an Express were:

- It provides many advantages over other libraries to develop web and mobile applications.
- Supports middleware functionality in order to response HTTP requests.
- It increases the rate of Node based Web applications development.
- Easily set up the routing table to execute different actions on the basis of URL and HTTP Method.
- Upon passing arguments to template, it renders HTML pages dynamically.

Since, everything was developed in Node environment, Mongoose (https://mongoosejs.com/), an Object Data Modeling (ODM) library was heavily used for the interaction between MongoDB and Node.js. The Schema and Models for storing and handling the curated datasets to MongoDB was developed using the mongoose. While MongoDB being the schema-less, here the schema refers to a document data structure as table definition defined by relational databases. Whereas, Models are constructors of higher-order that create an instance of a document based on schema. This phenomenon resembles a record of relational databases. This model was a major basis upon which all the asynchronous request was developed for this study. Doing this allowed to enable a promise (asynchronous operation) while retrieving the datasets from MongoDB that involved space entity.

The reason for using Mongoose was that it easily manages the relationships among data and also provide schema validation. Most importantly it translates objects into code and carry out the process of representing translated objects in MongoDB.

One of the prime reasons for choosing MongoDB was also that it provides a very efficient approach to store files directly in database rather than in file system. In other words, what it means is that if users want to store any files such as image, video file, audio file, zip file or pdf than he/she can directly store that in MongoDB itself. This is where the GridFS (a blueprint for storing and retrieving files that is more than the BSON-document size limit of 16 MB) becomes handy.

The working approach of GridFS is little bit different in a way that it divides the file into chunks and store respectively as one separate document. With the exception of last chunk, by default, GridFS separates a file into chunks of 255 kB. The last chunk could be of any size as needed. Likewise, files that are smaller than the chunk size simply have a single chunk along with addition of some space for its metadata.

Basically, GridFS have two main collections in order to store data. These collections are called chunks and files. Chunks is specifically for storing binary chunks that are divided into 255kB as stated above, whereas, files collection is to store the associated metadata of that data. The best thing is that it accepts any data type as metadata field and also allows to store any numbers of extra information that the users feel is necessary. For this study geo-coordinates and theme of the dataset are stored as objects in metadata.

A bucket is a common place where the GridFS put all the collection. While storing the collection it simply does by prefixing each collection with the respective bucket name. If the user gives no name then it creates a default bucket called “fs” with two collections named as fs.files and fs.chunks.

Within a single database, users can have have numerous buckets each with different names. When a query is made based on metadata then GridFS recollect and assemble the related chunks together as required. In other words, it means that the users can approach an information randomly, like skipping the end of the audio file. Likewise, the specific reasons for using MongoDB for this study were:

- Being a schemaless NoSQL document database, it allows to store JSON documents directly with maximum flexibility unlike the restriction of SQL database. This as a result reduces the complexity of deployments and increase the rate of application development.

- With GridFS it extends beyond its storing capacity of files larger than 16 MB rather it also improves the performance by not loading the entire file into the memory.

- For a better execution and greater efficiency of search based on a query, GridFS uses a index on both collections i.e. chunks and files. For a convenience, GridFS automatically create an index as per the nature of the stored data and installed MongoDB driver. Additionally, users can create as many indexes as necessary depending on the application being built. In this study, geospatial index namely 2dsphere was created for geo coordinates of the data stored as documents in MongoDB.

- As the volume of data increases in database, single machine won't be enough to efficiently perform an operation so multiple machines are required. This process of splitting data across multiple machines is called Sharding and MongoDB executes it with greater accuracy and much convenience.

Frontend:

As the front-end, minified ReactJS (https://reactjs.org/) based web-app[1] was created. Here, the front-end refers to the interface which allows participants to interact with backend system through buttons and input field box. The web-app’s frontend was designed as per the interface

[1] https://giveourdata.herokuapp.com/api/

design guidelines i.e. “10 Usability Heuristic for User Interface Design” by Jakob Nielson and Rolf Molich, 1995.

1. Visibility of system status:
   - ❖ Participants can easily see if the status is OK or have an Error.
2. Match between system and the real world:
   - ❖ All the labels in the user interface is targeted for moderate English speakers so that they get familiar within the first glance to the interface.
3. User control and freedom:
   - ❖ All request is based on GET method, so participants have freedom to request any number of times to server without having to worry if their action could make permanent change in the system.
4. Consistency and standard:
   - ❖ Consistency was mentioned between different platforms such as mobile and laptops. Participants can evaluate using their smart phone as well.
5. Recognition rather than recall:
   - ❖ Single page application with distinct colors which make the participants easily recognize the input and output section.
6. Flexibility and efficiency of use:
   - ❖ Flexibility and efficiency were maintained by chunking the entire script. Tested the loading time with Google Chrome slow 3G network and still had 8.33sec.
7. Aesthetic and minimalist design
   - ❖ Simple and plain design was applied.
8. Help users recognize, diagnose and recover from errors:
   - ❖ Participants will be notified about what could have potentially gone wrong while querying such as typos in the parameter.
9. Help and documentation
   - ❖ Documentation link was embedded in the footer section.

ReactJS was used because it was easy to connect with Node based app. Likewise, the build functionality of create-react-app was convenient to bundle all the static web assets and deploy later as the live site. This app was deployed on Heroku (https://www.heroku.com/), a container-based cloud Platform as a service. Heroku was chosen because it was flexible, user-friendly and most importantly it was free for testing web-app.

Postman:

Once the curated datasets were stored in MongoDB along with necessary indexing and running Node server, client making a request was simulated using a Postman. This was because until this point only backend was developed, and it was necessary to check that it was working as expected. As the main objective of the API was to retrieve spatial datasets which were the representation of the resource, among the 4 main HTTP methods only the GET method was used. Postman was used because in general it propels the rate of API development process in a much easier and better way. Specific reasons for choosing postman were:

- Its API Documentation feature allows to easily create documentation, view private API documentation and at a same time share public API documentation in well formatted web page.
- All the created collections for particular can be generated and hosted as a browser-based API documentation in real-time.
- Further dissecting the user created collection, postman provides a private and public documentation view from synced data in the servers which makes the process of updating the API documentation really fast and efficient.
- The debugging is also really easy as users can inspect API response within Postman.
- The support for variables and environments allows to save and reuse values in multiple collections.
- Each single query is stored as history unless saved to comprehensive postman collection.
- Most importantly it allows to check and verify if the API was behaving as expected or not.

3.6 About this API

One of the significant features of this API is its capability to handle the request regardless of any given order of parameters. The typical API has the necessity of following some sort of rigid order that limits the developer's efficiency in terms of time as such API demands more time to learn about it. As the detail about each parameter i.e. (SPACE, TIME, and THEME) is already mentioned in the previous section, this section will focus on the overall picture of this API including an error handling, natural language support for denoting time, and permutation and combination of possible number of handling the routes.

Permutation and Combination

In the simplest language combination means the selection of all the objects or only part of a set of objects without caring the order by which the objects get selected. Similarly, permutation on the other hand considers the arrangement of objects including an emphasis on the order they are arranged.

For the single parameter both the permutation and combination remain one as only single parameter is involved.

1. api/:time

2. api/:theme
3. api/:space

Hence, nPr = 0! = 1 and nCr = 0! = 1.

For the query involving two parameters the permutation is 6 and combination is 3. Following are the possible orders:

1. api/:time/:theme and api/:theme/:time
2. api/:time/:space and api/:space/:time
3. api/:space/:theme and api/:theme/:space

Similarly, for the query involving three parameters the number of permutations is 6 with 1 combination. Below are the possible orders:

1. api/:time/:theme/:space
2. api/:time/:space/:theme
3. api/:theme/:time/:space
4. api/:theme/:space/:time
5. api/:space/:time/:theme
6. api/:space/:theme/:time

In this way, the API can handle a total of 15 different orders of the request providing the developers with immense flexibility.

Error Handling

Since, the HTTP method used is GET only, the API communicates the developers with two significant code status 200 and 400. For this API these codes are generated under following conditions.

Status code 400:

Status code 400 means server did not understand what the client is asking about. If the parameter from request fails to satisfy any of entity (THEME, TIME, SPACE) then it will throw an error with custom message {“error”: “Oops, typos in the parameter”} with status code 400. The main objective of this custom message is to provide solution to developers showing where the error might have come from.

Status code 200:

Status code 200 means "OK" which means server understood the request and sending something as a response back to client. The case of status code 200 from this API produces following response.

Datasets or Custom message:

The RESTful API developed for this study undergoes following steps to decide whether to send datasets or custom message:

Once the server gets the request first it checks if the request is associated with the THEME or TIME entity. If it belongs to neither of these then it goes to check whether it is SPACE entity or not. For the space, it calls the Openstreetmap (OSM) nominatim API (https://nominatim.openstreetmap.org/) which searches OSM data by the provided name of the city in order to generate synthetic addresses and its associated geojson coordinates.

The open street map is a gigantic poll of data related with myriad of place names ranging from small corner street of very primitive area to mega cities of the world. Thus, it has a potential to produce geojson for almost any place on earth.

Accordingly, if it finds the place name then it sends a geojson coordinates to the node server that was set up for this study. Back in the server, again these geojson coordinates are passed as arguments for geowithin algorithm of MongoDB. This algorithm checks if there is any datasets within these coordinates. If dataset is found, then it sends those datasets to the client as a response. Similarly, if it does not find anything then it will send the custom message. The custom message is {"Length": 0}. This message means that the name of the place was found but no data within that area was found thus the length of data becomes 0.

Natural language support

In most of the typical APIs, mentioning date is one of the factors that has a rigid format to follow for developers. Some of the common date formats are shown in the table below:

S.NO.	Description	Format
1	American month, day and year	mm "/" dd "/" y
2	Four digit year, month and day with slashes	YY "/" mm "/" dd
3	Four digit year and month (GNU)	YY "-" mm
4	Year, month and day with dashes	Y "-" mm "-" dd

Looking at this table one can easily say it is quite confusing to see these many date formats. Above four are just the most common one, in-fact there are tens of others we well. The most hindering task is when one date format has to be converted into another and vice-versa. It demands extra logic and additional line of codes making the whole codebase more error prone. With such realization, this RESTful API is designed in a way that supports the semantic description of date.

In the core of this RESTful API lies the powerful module called sherlock.js (https://github.com/neilgupta/Sherlock) that has an immense ability to parse the natural language and extract the date parameters from the given text. While implementing such feature relations between intervals based on analysis of Allen Interval algebra was considered. Most significantly the relation Before, After, and Equals were used as the main objective of API is to retrieve datasets. These intervals clearly denote the beginning, during and end points of the date for main logic to function in the server. It has provided API an ability to allow the developers to express the date in different input formats which are common in US English. For an instance, retrieving a crime dataset of Bonn city for yesterday using this API.

api/:space/:theme/:time = api/bonn/crime/yesterday

Such capability will not only increase the rate of developers' productivity but also helps in overall code maintenance and scalability in the long run.

4 EVALUATION

4.1 Study design

Conducting a user study to test a user interface with target users is regarded as "gold standard" in HCI [37]. A similar test can be performed with APIs as well [22]. With this statement in mind, here, in this study the participants were provided with a set of questions for which they had to use the API to get an answer. The goal of the study was to find out that if expressing the parameters in the natural language along with flexibility in the parameter's order really ease the process of retrieving datasets from an API level or not.

The hypothesis was that querying the datasets in terms of Space, Time, and Theme in the API level by expressing them in natural language expression increases the API learnability.

These two variables were considered for the evaluation procedure.

i. Independent:
Different type of queries in API level involving Space, Time and Theme.

ii. Dependent:
Satisfaction and learnability of this API.

4.2 User sampling

The total number of participants were 20 and gender balance were taken into account. Among the participants were both developers (having an experience using other APIs before taking part to this study) and non-developers (Having no prior experience with any APIs before taking part to this study). The only criteria were that the participants need to understand at-least the moderate English as the study was designed in an English language. The recruitment was done through the Facebook messenger, WhatsApp and word-of-mouth. No rewards of any kind were given to the participants. The only infrastructure needed was the laptop with good internet connection. The within-group study design was carried out for this study which means that all the participants were exposed with same set of questions throughout the evaluation process.

4.3 Procedure

Firstly, the participants were presented with an Informed consent form which will state their approval for their willingness to volunteer in the study. Secondly, they were asked to go through the API documentation[2] to know about the context of API. Once they were done with the documentation, they were asked to visit the Google form[3] presented with a set of questions. Here, the main objective of google form was to record if the participants had experience with API or not. Also, it was to observe the overall effectiveness, participants satisfaction and learnability of the API.

Those participants who were unable to attend physically were scheduled via Facebook messenger video call. The screen of the participants was shared through “Facebook share screen extension” so that I could see how they were proceeding the task. Once they were connected they were also asked to follow the same steps as for participants who physically participated.

The screen of the participants was recorded in order to find out questions like:

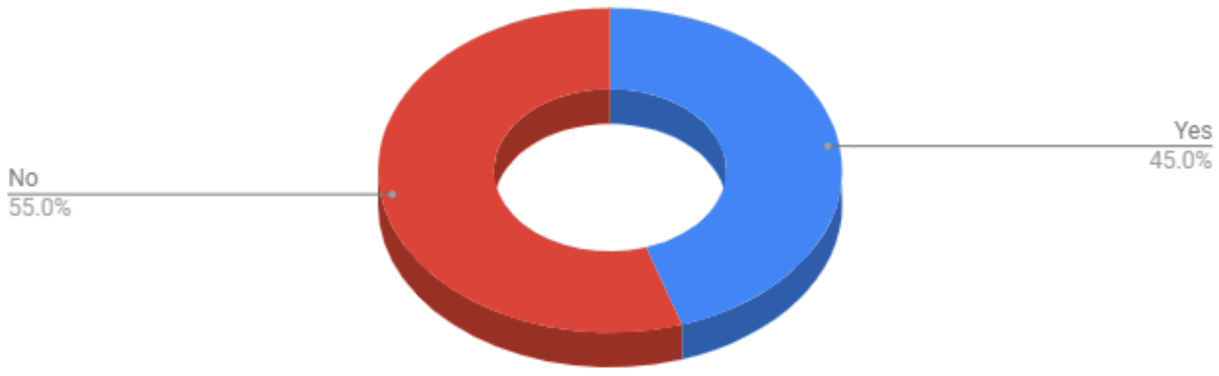
1. For how long they spend the time reading the documentation before beginning the task and how often they checked back per task?
2. How many attempts did the participants take to solve a task?

[2] https://documenter.getpostman.com/view/5553462/RzffJ9cc

[3] https://docs.google.com/forms/d/e/1FAIpQLSfUXi-eHgP6boi2vSjEJVG172hK18uHRFAH9I7_zUo8zmYyuQ/viewform.

5 RESULT

Chart showing the percentage of participants having the API using experience.



Frequency of API usage in the last 12 months

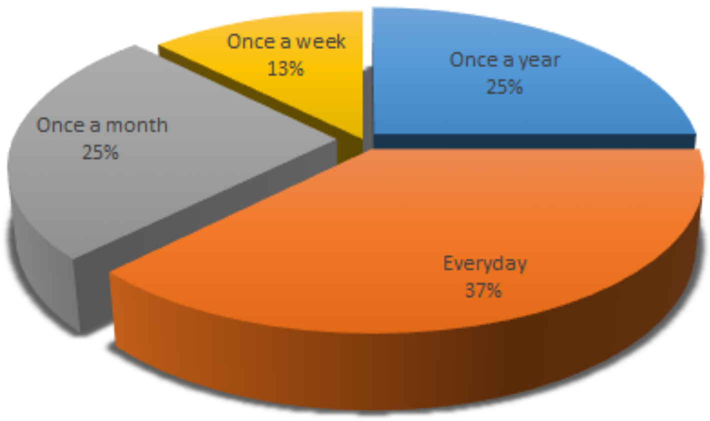
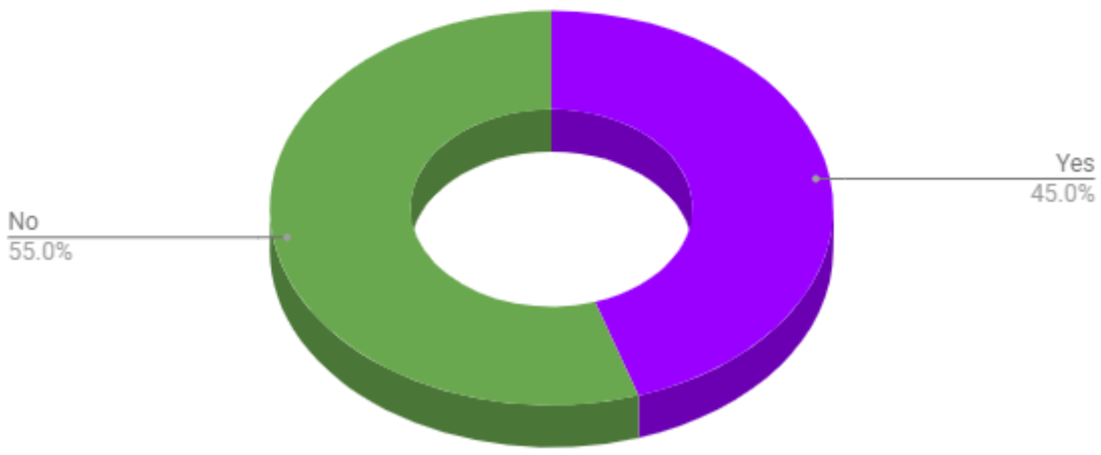
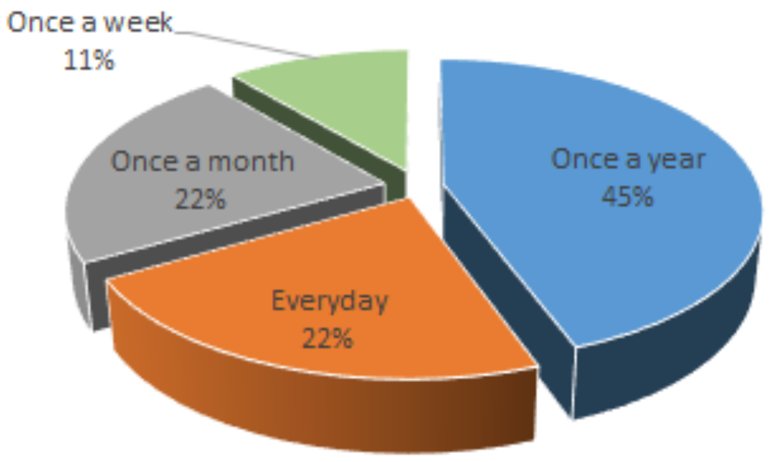


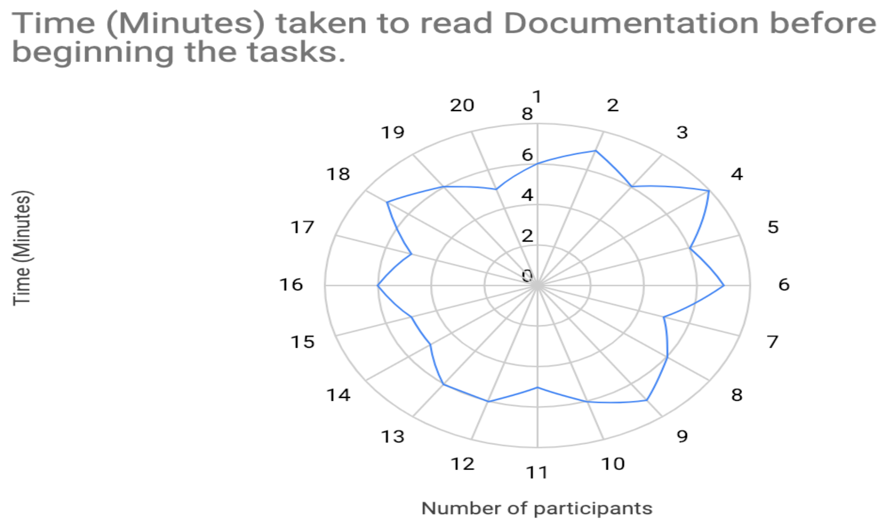
Chart showing the percentage of participants having the experience of querying the datasets from the API level.



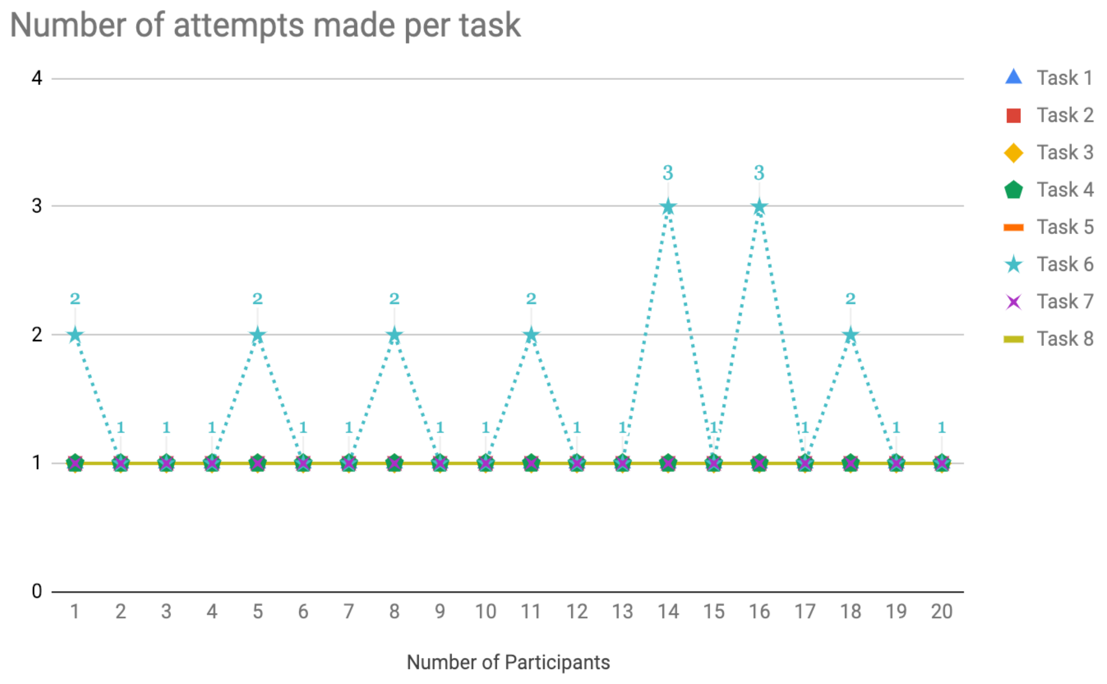
Frequency of API usage in the last 12 months to retrieve datasets



5.1 Effectiveness

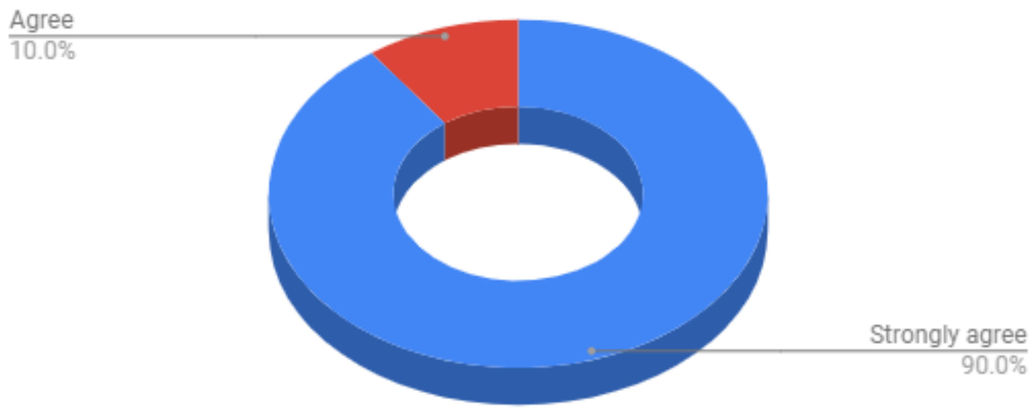


The time taken to read an entire Documentation before beginning the task was an average of 6 minutes with minimum being 5 minutes and maximum of 8 minutes. This clearly shows that the participants found the API learnability to be really high. In other words, the users did not have to check the documentation repeatedly while trying to solve each task.

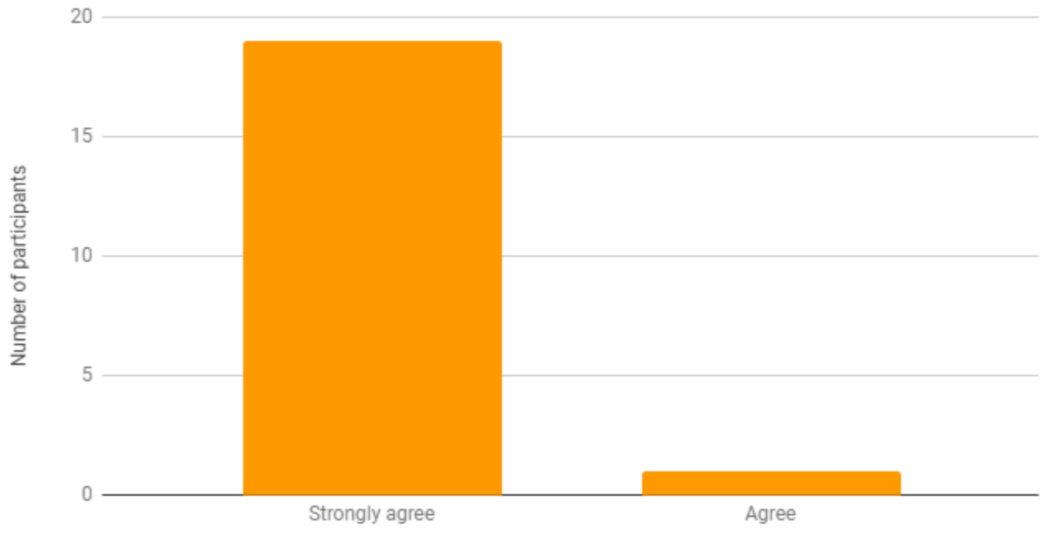


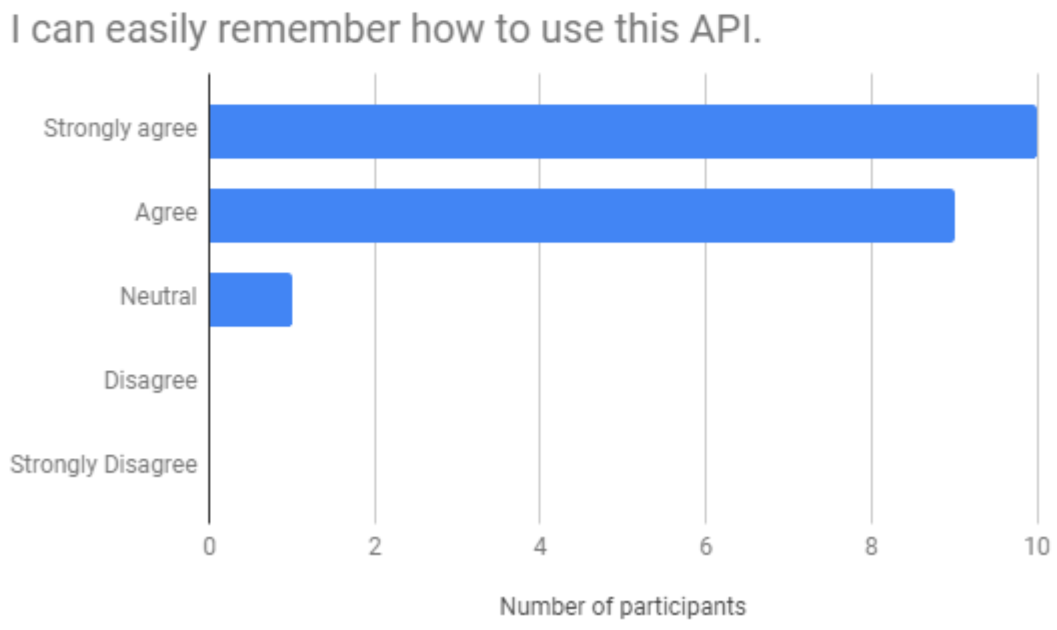
All the participants successfully completed the entire task with ease without facing any problems. As can be seen in the above plot except task number six all others were solved with first attempt. Participants who had to make 3 attempts were those with no prior experience of retrieving datasets from API level. The reason for participants attempting task six up to 3 times is mentioned in the discussion section of this report. The participants who had prior experience with other API commented that this API was different than others and found to be really user-friendly and had great learnability. Overall, the participants found both the documentation and API really effective.

Percentage of participants saying that they quickly learned to query using this API.

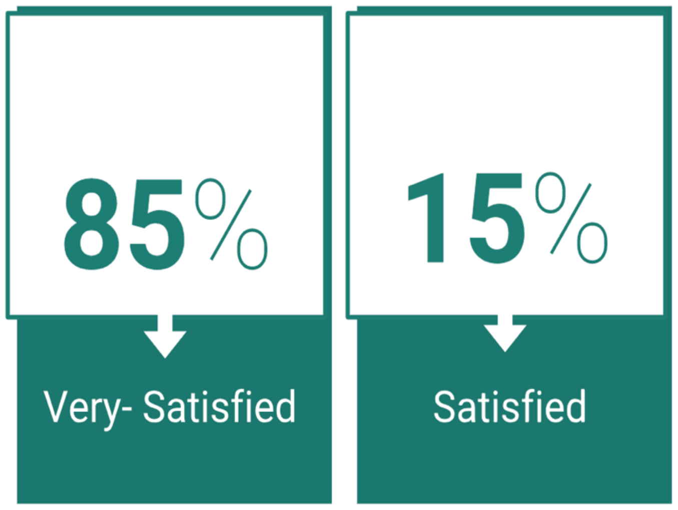


Bar digram showing the number of participants agreeing that this API was easily learnable.

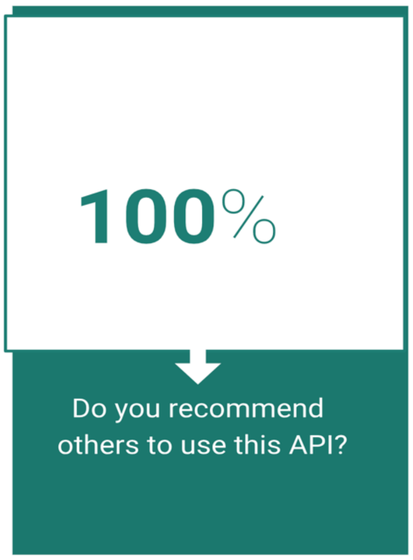




5.2 Satisfaction

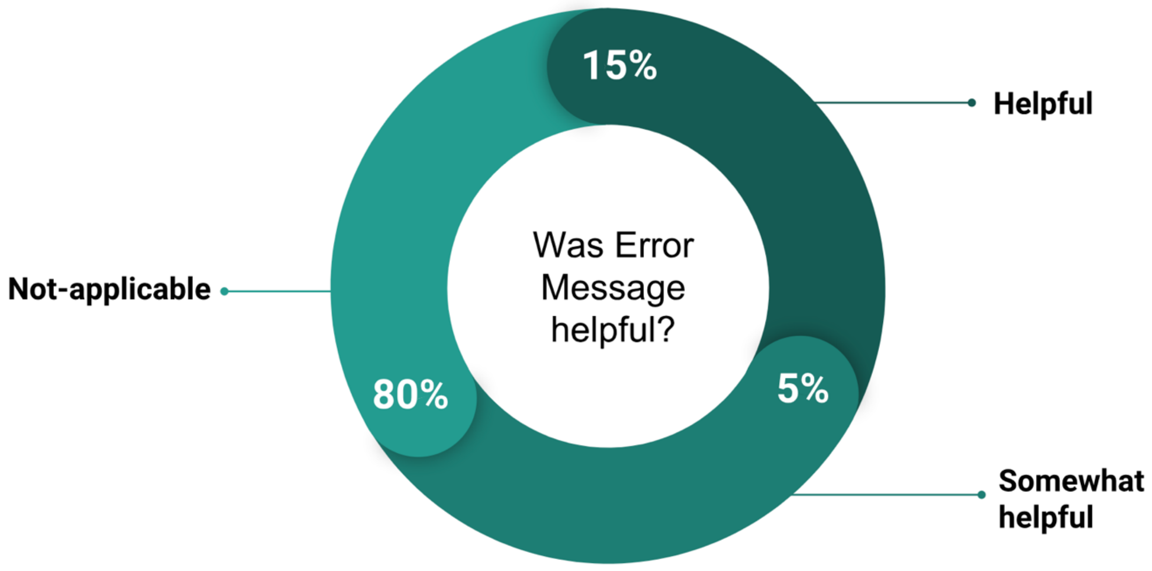


The participants were asked to rate their satisfaction level based on the documentation and first-hand experience of using the API to solve the set of questions. All the participants were able to answer the entire questions successfully. In the above figure, we can see that more than 80% of participants were very satisfied and felt a sheer convenience while learning about this API.



One of the significant steps for the evaluation was to know whether the participants are willing to recommend others to use this API or not. The above figure depicts that all the participants are willing to recommend this API based on their experience with this API. Among the participants were a complete novice (who have never used any APIs before) to someone who had been programming for last 20 years. The most common thing between these two kinds was that both groups found the API really intuitive. In other words, it means that participants really felt that this API was user-friendly and easily learnable.

5.3 Errors



The above figure shows that 80% of the participants did not encounter any error while solving the questions. The error made by remaining participants was the slips error. In general slips occur when users expect to do one thing, instead end up doing some other action. Usually, this is common when the users are in autopilot mode lacking enough attention to the task at hand. Sometimes it also occurs by too much thinking or overlooking the task at hand and making it more complicated than it really is. The most common slips observed in this evaluation was the missing of backslash "/" to differentiate the parameter. For an example, to query the Crime datasets of Bonn the participants would simply do "Bonn Crime", forgetting to put the "/". Especially, it was seen among the participants with no prior experience of using any API.

6 DISCUSSION

6.1 Limitations

The MongoDB version 4.0.0 was used for this study. It supports powerful geospatial query operators like $geoIntersects, $near and $nearSphere. However, only $geowithin and $geonear were used for this study. Likewise, in terms of geometry specifiers, among ($box, $center, $maxDistance, $minDistance, $centersphere, $geometry, $polygon, $uniqueDocs) only the $maxDistance was used.

The incorporation of remaining query operators and geometry specifiers would have provided more spatial capabilities such as bounding-box method to retrieve datasets.

6.2 Interpretation of Results

The overall result was very promising as more than 80% of participants were very satisfied with the work.

Especially, the 45% of participants who were developers (having prior experience using other APIs) were more excited about this API as they had the real experience of using the typical API where the datasets are retrieved using ID such as:

api.powervb.com/v1.0/myorg/datasets/cfafb-8037-4d0c-896e/34908-afgh-5d9c-592f/2018-12-23

The query parameters shown in above example does not speak for itself, in other words, the intuitiveness is very low. Developers shared their experience saying that the situations get even more difficult while using more than one API at a same time as each of them demands its own format for parameters like Date.

Remaining 55% of participants who were non-developers (having no prior experience using other APIs) also really enjoyed using the API. Overall, more than 95% of the participants stated that it was really easy to learn. Similarly, 90% stated that they were quickly able to query to retrieve the datasets.

While querying, 10% of the non-developers were seen to be slightly forgettable to put the "/" as separator in between the parameters, which is completely normal, as they had never queried datasets from API level in such manner before.

Regarding the remembrance of the API 95% of participants told that they could easily remember about how to query using this API without any real mental effort as the time and space parameters were expressed in natural language (today, yesterday) and scope of entire datasets is categorized as SPACE, TIME and THEME. Remaining 5% said they were neutral regarding an opinion on remembrance. The main reason for this was the nature with which the question number six was intentionally designed to resemble the typical API. Having said that participants were asked to manually enter longitude and latitude value with rigid format in question six as shown below.

"Suppose your current position is long,lat=[85.2861785888672,27.7002479940336] then find the total number of transport datasets that are within the radius of 9000? (unit is meter) (THEME/SPACE)"

The plan worked well because all the written suggestions for the further improvement of this API (provided in the appendix of this report) at the end of survey was about the question number six.

In the real-world situation, query involving the current coordinates are usually read from the laptop location reading system. Here, in this study rather a manual system was chosen because firstly, we wanted those participants who had no prior experience of using any APIs to see what it really feels like to manually enter the query parameter in rigid format. Secondly, it was also the fact that users have to go extra step to enable the location service on their laptop if they already have not turned it ON. In most of the cases, participants will be reluctant to do so.

7 CONCLUSION AND FUTURE WORK

Considering spatial, temporal and thematic aspects in metadata provides an effective and robust way to retrieve the datasets from API level. The API built for this study was solely based on non-relational database namely MongoDB and found to be really efficient because of its capability to create metadata property as normal JavaScript Object.

The result from the user-study showed that API can be easily learnable if users can express their query in natural language and at a same time do not have to worry about the parameters order. During the user-study, even those participants who never had an experience of querying datasets from API level solved almost all the tasks within the first attempt. They just spent some time reading the documentation before beginning the task.

In conclusion, the easily-learnable API that integrates spatial, temporal, and thematic aspect to access open government data can be the refined API, which the World Wide Web, 2015 was searching for.

As the future work much more scenarios can be explored further for all three aspects (space, time, and theme).

- More spatial operators like overlap, intersect, and covers can be implemented.
- It would be interesting to incorporate logic gates (AND, OR) for retrieving multiple datasets at once. Right now, only one thematic type can be retrieved at a time.
- The geo-coordinates of current location can be taken from the location reading system of the devices directly.
- The support for POST method can also be integrated so that users can add the reliable datasets themselves. However, a robust validation method for maintaining the quality of the datasets should be introduced beforehand.
- Usability of the API could be further improved considering the throughput time, memory consumptions etc.

Bibliography

[1] Open Government Data - OECD. (n.d.). Retrieved January 24, 2019, from http://www.oecd.org/gov/digital-government/open-government-data.htm

[2] Degbelo, A., Trilles, S., Kray, C., Bhattacharya, D., Schiestel, N., Wissing, J. and Granell, C. (2016) '*Designing semantic APIs for open government data*', JeDEM - eJournal of eDemocracy and Open Government, 8(2), pp. 21–58.

[3] Lee, M., Almirall, E. and Wareham, J. (2016) '*Open data and civic apps: First-generation failures, second generation improvements*', Communications of the ACM, 59(1), pp. 82–89. doi: 10.1145/2756542.

[4] Myers, B. A. and Stylos, J. (2016) '*Improving API usability*', Communications of the ACM. ACM, 59(6), pp. 62–69. doi: 10.1145/2896587.

[5] Stylos, J. and Myers, B. (2007) '*Mapping the space of API design decisions*', in Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2007). Coeur d'Alène, Idaho, USA: IEEE, pp. 50–57. doi: 10.1109/VLHCC.2007.36.

[6] Farooq, U., Welicki, L., & Zirkler, D. (2010). *API Usability Peer Reviews: A Method for Evaluating the Usability of Application Programming Interfaces*. Retrieved from http://dmrussell.net/CHI2010/docs/p2327.pdf

[7] Forum Virium Helsinki. (2016). *Harmonized Smart City APIs*. Retrieved from https://www.citysdk.eu/wp-content/uploads/2016/12/CookBook_web_v1.1.pdf (Forum Virium Helsinki, 2016)

[8] Berends, J., Carrara, W., & Vollers, H. (2017). *Analytical Report n5*. Retrieved from https://www.europeandataportal.eu/

[9] Fielding, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.

[10] *REST API Design Rulebook* by Mark Massé (O'Reilly). Copyright 2012 Mark Massé, 978-1-449-31050-9.

[11] F. ALLEN, JAMES. (2013). *Maintaining Knowledge about Temporal Intervals*. Commun. ACM. 26. 510-521. 10.1016/B978-1-4832-1447-4.50033-X.

[12] Strobl, C. (2017). *Dimensionally Extended Nine-Intersection Model (DE-9IM)*. In Encyclopedia of GIS (pp. 470–476). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-17885-1_298

[13] Donau-Universität Krems., C. P., & Lucke, J. von. (2012). *Open Government and (Linked) (Open) (Government) (Data)*. EJournal of eDemocracy and open government. JeDEM - eJournal of eDemocracy and Open Government (Vol. 4). Danube University of Krems. Retrieved from https://jedem.org/index.php/jedem/article/view/143/115

[14] Surwase, V. (2016). *REST API Modeling Languages-A Developer's Perspective*. IJSTE-International Journal of Science Technology & Engineering | (Vol. 2). Retrieved from www.ijste.org

[15] Stylos, J., & Myers, B. (2007). *Mapping the Space of API Design Decisions*. https://doi.org/10.1109/VLHCC.2007.44

[16] Henning, M. (2009) *'API design matters', Communications of the ACM*, 52(5), pp. 25–36. Doi: 10.1145/1506409.1506424.

[17] Scheider, S. and Kuhn, W. (2015) '*How to talk to each other via computers - Semantic interoperability as conceptual imitation*', in Zenker, F. and Gärdenfors, P. (eds) Applications of Conceptual Spaces. Springer International Publishing, pp. 97–122. doi: 10.1007/978-3-319-15021-5_6.

[18] Kechagia, M., Mitropoulos, D., Zeller, A., Kechagia, M., Mitropoulos, · D, Spinellis, · D, … Spinellis, D. (n.d.). *Charting the API minefield using software telemetry data*. https://doi.org/10.1007/s10664-014-9343-7

[19] ProgrammableWeb | ProgrammableWeb. (n.d.). Retrieved December 21, 2018, from https://www.programmableweb.com/about

[20] Petychakis, Michael & Lampathaki, Fenareti & Askounis, Dimitris. (2015). *Adding Rules on Existing Hypermedia APIs*. 1515-1517. 10.1145/2740908.2743041.

[21] Richardson, L. and Amundsen, M. (2013) *RESTful web APIs, RESTful Web application programming interfaces*. O'Reilly Media, Inc. doi: 10.1017/CBO9781107415324.004.

[22] Myers, B. A., & Stylos, J. (2016). *Improving API usability*. Communications of the ACM, 59(6), 62-69.

[23] Beaton, J. K., Myers, B. A., Stylos, J., Jeong, S. Y. (Sophie), & Xie, Y. (Clare). (2008). *Usability evaluation for enterprise SOA APIs. In Proceedings of the 2nd international workshop on Systems development in SOA environments* - SDSOA '08 (p. 29). New York, New York, USA: ACM Press. https://doi.org/10.1145/1370916.1370924

[24] API Concepts Framework - CodePlex Archive. (n.d.). Retrieved December 21, 2018, from https://archive.codeplex.com/?p=apiconceptsframework

[25] Zibran, M. F., Eishita, F. Z., & Roy, C. K. (2011). *Useful, but usable? Factors Affecting the Usability of* APIs. 2011 18th Working Conference on Reverse Engineering. doi: 10.1109/WCRE.2011.26

[26] J. Daughtry, U. Farooq, J. Stylos, and B. Myers. *API usability: CHI'2009 special interest group meeting*. In CHI, pages 2771–2774, 2009.

[27] B. Myers, A. Ko, S. Park, J. Stylos, T. LaToza, and J. Beaton. *More natural end-user software engineering*. In WEUSE, pages 30–34, 2008.

[28] J. Stylos, S. Clarke, and B. Myers. *Comparing API design choices with usability studies: A case study and future directions*. In AWPPIG, pages 131–139, 2006.

[29] J. Beaton, S. Jeong, Y. Xie, J. Stylos, and B. Myers. *Usability challenges for enterprise service-oriented architecture APIs*. In VL/HCC, pages 193–196, 2008.

[30] S. Clarke and C. Becker. *Using the cognitive dimensions framework to evaluate the usability of a class library*. In Joint Conf. EASE & PPIG, pages 359–366, 2003.

[31] C. Bore and S. Bore. *Profiling software API usability for consumer electronics*. In ICCE, pages 155–156, 2005.

[32] D. Ratiu and J. Jurjens. *Evaluating the reference and representation of domain concepts in APIs*. In ICPC, pages 242–247, 2008.

[33] Murphy, L., Alliyu, T., Macvean, A., Kery, M. B., & Myers, B. A. (n.d.). *Preliminary Analysis of REST API Style Guidelines*. Retrieved from https://codeplanet.io/principles-good-restful-api-design/

[34] J. Stylos. *Making APIs More Usable with Improved API Designs, Documentation and Tools*. PhD Dissertation: Computer Science Department, Carnegie Mellon University. 2009.

[35] Fielding, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.

[36] Berners-Lee, Tim, Roy T. Fielding, et al. *Uniform Resource Identifier (URI): Generic Syntax*, RFC 3986, RFC Editor, 1998 (http://www.rfc-editor.org/rfc/rfc3986.txt).

[37] Nielsen, J., *Usability Engineering*. 1993, Boston: Academic Press.

[38] API Strategy 101: What is an API? / API Academy. (n.d.). Retrieved January 15, 2019, https://www.apiacademy.co/lessons/2015/04/api-strategy-lesson-101-what-is-an-api

[39] Albano, C.S. 2013. *Open Government Data: A Value Chain Model Proposal*. In Proceedings of the 14th Annual International Conference on Digital Government Research, 285-86. ACM Press. doi:10.1145/2479724.2479775.

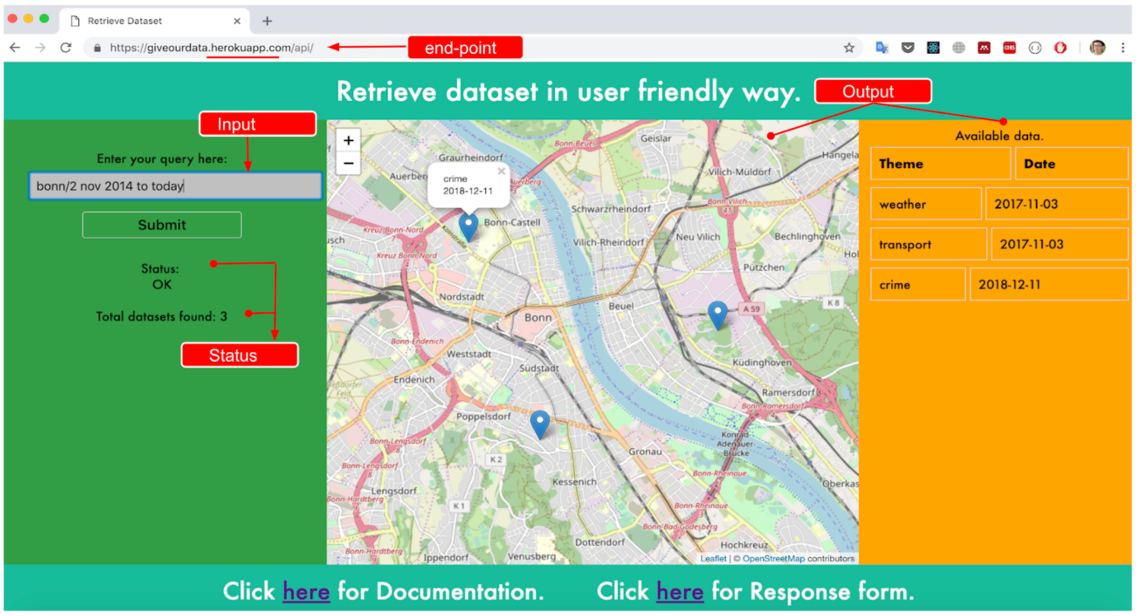
[40] Hoxha, J., & Brahaj, A. (2011). *Open Government Data on the web: A Semantic Approach*. In 2011 International Conference on Emerging Intelligent Data and Web Technologies. https://doi.org/10.1109/EIDWT.2011.24

[41] Verborgh R. et al. (2014). *Querying Datasets on the Web with High Availability*. In: Mika P. et al. (eds) The Semantic Web – ISWC 2014. ISWC 2014. Lecture Notes in Computer Science, vol 8796. Springer, Cham

[42] Kiryakov, A., Popov, B., Terziev, I., Manov, D., & Ognyanoff, D. (2004). *Semantic annotation, indexing, and retrieval*. Web Semantics: Science, Services and Agents on the World Wide Web, 2, 49–79. https://doi.org/10.1016/j.websem.2004.07.005

[43] Joshua Bloch (2006). *How to design a good API and why it matters* - ACM Digital Library. 22 Oct. 2006, https://dl.acm.org/citation.cfm?id=1176622. Accessed 19 Jan. 2019.

APPENDIX


Retrieve Dataset
https://giveourdata.herokuapp.com/api/
end-point
Retrieve dataset in user friendly way.
Output
Input
Enter your query here:
Submit
Status
Click here for Documentation.
Click here for Response form.

bonn/2 nov 2014 to today

Status:
OK

Total datasets found: 3

crime
2018-12-11

Available data.

Theme	Date
weather	2017-11-03
transport	2017-11-03
crime	2018-12-11

1. Front-end interface of the web-app

I think its confusing that Time and rasius queries work differently. Time gives all themes. Space only the specified one.

Maybe improve the syntax for searching within a radius? It's not as natural as the rest of the syntax. Keep up the good work!

If query supports lat long without specifying theme, it would be much helpful.

Well done. Its really interesting

The question which consisted longitude and latitude, theme and radius (3 parameters) has documentation or functioning of API is based on two parameters (THEME/SPACE), so I was bit confused why processing of question consisting three parameters is conducted using two parameters approach. Could be an issue to brainstorm if it has significant role for the analysis. Else, cheers.

2.Suggestions for the further improvement of the app.

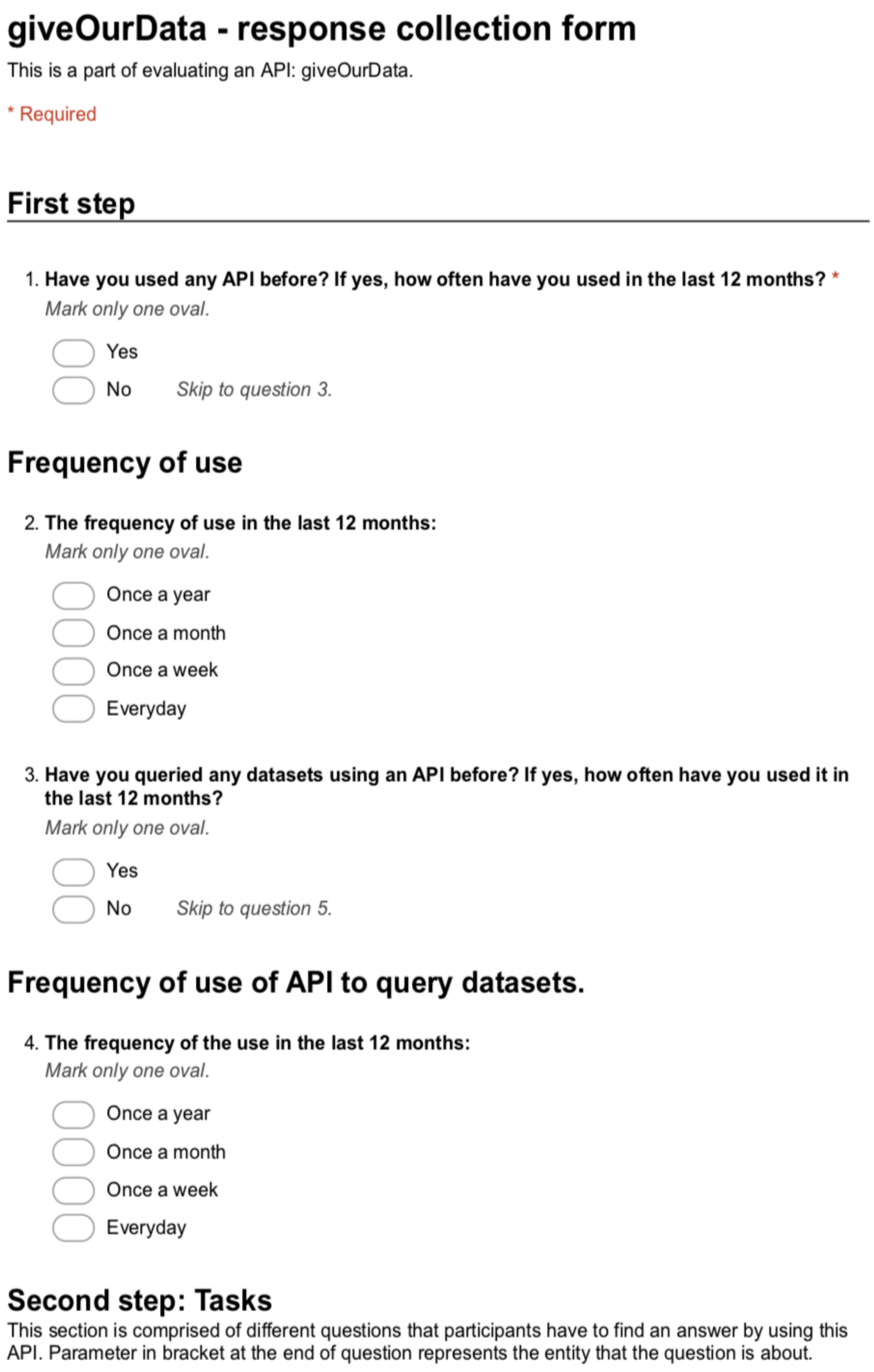
No. of Participants	Number of attempts made per task							
	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8
1	1	1	1	1	1	2	1	1
2	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	1
5	1	1	1	1	1	2	1	1
6	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1
8	1	1	1	1	1	2	1	1
9	1	1	1	1	1	1	1	1
10	1	1	1	1	1	1	1	1
11	1	1	1	1	1	2	1	1
12	1	1	1	1	1	1	1	1
13	1	1	1	1	1	1	1	1
14	1	1	1	1	1	3	1	1
15	1	1	1	1	1	1	1	1
16	1	1	1	1	1	1	1	1
17	1	1	1	1	1	3	1	1
18	1	1	1	1	1	2	1	1
19	1	1	1	1	1	1	1	1
20	1	1	1	1	1	1	1	1

3.Number of attempts made by participants per task

No. of Participants	Time spent reading documentation. (minutes)
1	6
2	7
3	6
4	8
5	6
6	7
7	5
8	6
9	7
10	6
11	5
12	6
13	6
14	5
15	5
16	6
17	5
18	7
19	6
20	5

4.Time spent reading the documentation before beginning the task

5.Evaluation response form



giveOurData - response collection form

This is a part of evaluating an API: giveOurData.

* Required

First step

1. **Have you used any API before? If yes, how often have you used in the last 12 months?** *

   Mark only one oval.

   - Yes
   - No *Skip to question 3.*

Frequency of use

2. **The frequency of use in the last 12 months:**

   Mark only one oval.

   - Once a year
   - Once a month
   - Once a week
   - Everyday

3. **Have you queried any datasets using an API before? If yes, how often have you used it in the last 12 months?**

   Mark only one oval.

   - Yes
   - No *Skip to question 5.*

Frequency of use of API to query datasets.

4. **The frequency of the use in the last 12 months:**

   Mark only one oval.

   - Once a year
   - Once a month
   - Once a week
   - Everyday

Second step: Tasks

This section is comprised of different questions that participants have to find an answer by using this API. Parameter in bracket at the end of question represents the entity that the question is about.

5. **What is the total number of datasets found for 3 November 2017? (TIME)** *

Mark only one oval.

- ⬭ 10
- ⬭ 11
- ⬭ 12
- ⬭ 13

6. **Find the oldest date and latest date for migration datasets? (THEME)** *

Mark only one oval.

- ⬭ 2016-10-22 and 2017-11-03
- ⬭ 2016-10-23 and 2017-11-04
- ⬭ 2014-11-22 and 2017-11-03
- ⬭ 2014-09-22 and 2017-10-22

7. **What are different datasets that are available in Bonn? (SPACE)** *

Check all that apply.

- ☐ Weather
- ☐ Transport
- ☐ Landuse
- ☐ Crime

8. **What is the total number of datasets available from 1 June 2016 through Today? (TIME)** *

Mark only one oval.

- ⬭ 17
- ⬭ 18
- ⬭ 19
- ⬭ 20

9. **How many datasets are there in Dortmund from October 2nd 2016 to yesterday? (SPACE/TIME)** *

Mark only one oval.

- ⬭ 1
- ⬭ 2
- ⬭ 3
- ⬭ 4

10. **Suppose your current position is long,lat=[85.2861785888672,27.7002479940336] then find the total number of transport datasets that are within the radius of 9000? (unit is meter) (THEME/SPACE)** *

Mark only one oval.

11. **How many crime datasets are there between 12 November 2014 and day before yesterday? (THEME/TIME)** *
Mark only one oval.
   - 1
   - 2
   - 3
   - 4

12. **Is there any difference in the number of datasets returned by these two queries: Stuttgart/3rd November 2017/weather (SPACE/TIME/THEME) and weather/stuttgart/3 November 2017 (THEME/SPACE/TIME)** *
Mark only one oval.
   - Yes
   - No

Third step

This section is to check the participants satisfaction.

13. **Overall, how satisfied are you with this API?** *
Mark only one oval.
   - Very satisfied
   - Satisfied
   - Neither satisfied nor dissatisfied
   - Dissatisfied
   - Very dissatisfied

14. **Would you recommend others to use this API?** *
Mark only one oval.
   - Yes
   - No

15. **Not necessarily but if you got any error message then was it helpful?**
Mark only one oval.
   - Extremely helpful
   - Helpful
   - Somewhat helpful
   - Not helpful at all
   - Not applicable

Fourth step

16. **I learned to write queries quickly using this API** *

Mark only one oval.

- Strongly agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

17. **It is easy to learn how to use this API** *

Mark only one oval.

- Strongly agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

18. **I could easily remember how to use this API** *

Mark only one oval.

- Strongly agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

Suggestions (Optional)

19. **Do you have any suggestions regarding this API for further improvement?**