Detailed Project Proposal and initial literature review

Part 1

Project Title

A comparison of GraphQL and REST as API frameworks for searching Open Data services

Project Topic

Originally Open Data was snapshots of large datasets that developers had to repeatedly download to keep their copy up to date.

The government now recommends that Open Data should be published as a service and access via APIs. The government recommends REST as the API framework.

During my industrial placement at Porism Limited, I worked on setting up Open Referral UK REST APIs for some councils as a part of the Local Government Association's loneliness project. Open Referral UK is a data standard for local councils to publish a local directory of community services. I have also used the REST API for Open Active which is developed by the Open Data Institute to provide access to sports data.

The REST API is good at

- listing resources like services or organisations
- retrieving details of an individual resource
- creating or updating a resource

However, the REST API was not good at handling a complex search request, because the request must be contained within the query string of a GET request.

Open Referral UK could be queried for multiple combinations of criteria such as

- service types
- the needs and circumstances of a person
- dates and times when services are run
- the location of venues
- eligibility conditions

This can lead to complex ANDs and ORs which it is not practical to achieve in a single call to the REST API. This can mean that developer must make several calls when dealing with a customer's request and amalgamate them in their app. This can be hard which might put off some app developers or reduce the customer experience.

The literature review found that many people have found this to be a problem in REST APIs, but no one seemed to have a good solution.

Open Referral UK was build following the Government Digital Services API specifications, Government Digital Services are now looking at including GraphQL in these specifications. Open Referral UK is a API that could benefit from the advantages of GraphQL which is what I am going to investigate.

GraphQL was publicly released by Facebook in 2015 as an alternative to REST. GraphQL allows the client to specify the fields that it wants to be returned so this prevents excessively large amounts of data from being returned, which can be the case in REST because the fieldset is determined by the API.

GraphQL doesn't have an inbuilt solution for complex searching, but an object parameter can be sent with the request. There are some examples on the web to allow objects that contain complex search criteria. This approach can be used to specify ANDs and ORs to allow complex searches to be achieved in a single call.

GraphQL has the potential to provide app developers with a simpler interface to search an Open Data service. This should encourage more apps to be developed giving greater choice to the public with how they access open data.

During this project, I will create a GraphQL API endpoint to access Open Referral UK data. If time permits I will also create a GraphQL endpoint to access Open Active data. I will build in a complex search capability into these APIs.

I will create a client website to consume the GraphQL endpoints and demonstrate how a customer's complex query can be easily handled in a single call. Using these tools I will compare approaches for using complex queries for searching and filtering that might help a client to find a suitable service from the Open Data.

I will create a report of my finding and conclusions from feedback.

Client, Audience and Motivation

My literature review has found that the government wants to encourage app developers to use Open Data to improve people's access to data and services using that data. Government need to promote a set of data and API standards so that developers find it easy to build Open Data into their apps and so that users get a good experience and come back to use the app again.

The government are currently working up some guidance on when to use GraphQL, although this doesn't yet cover how to achieve complex searches. They have offered to review my project findings and add them to their guidance if appropriate.

A GraphQL endpoint to access data is aimed to help app developers, by making the querying of data simpler and allow more control of the data being returned by the server.

With GraphQL I am intending to allow for complex querying including ANDs and ORs of multiple queries. Currently in REST APIs complex queries can be very complicated by using their own query syntax and a large number of parameters. This can make the use of the API hard to understand for new developers and increase the complexity of client apps.

By implementing a GraphQL API for an Open Data endpoint it is intended to allow app developers to create better apps and provide better search results due to the increased flexibility of GraphQL's searching and filtering possibilities. This helps the developer and in turn the client of the app to provide better results that are more tailored to the user.

I will ask for feedback from the Open Referral UK app community, which is led by Porism, to ask their opinion if they feel that this work has demonstrated that they could provide a better service if they use a GraphQL API.

My literature review was unable to find an academic paper that proposes a solution for complex searches through an API, although there are plenty of websites that are asking for the best way to do this. My report will add to this debate.

Primary Research Plan

For this project, I intend to split it into these sections

- Building GraphQL endpoint
- Building client demonstration app
- Getting client feedback on the endpoint and client app
- Evaluating the feedback
- Produce the report

Week 1	Researching and planning GraphQL Endpoint
Week 2	Accessing Open Data for the API
Week 3	Building GraphQL endpoints
Week 4	Building GraphQL endpoints
Week 5	Testing / automate tests for GraphQL endpoint
Week 6	Planning client app
Week 7	Building the client app
Week 8	Building the client app
Week 9	Testing / automate tests for the client app
Week 10	Getting Client feedback
Week 11	Evaluating and producing the report
Week 12	Evaluating and producing the report
Week 13	Evaluating and producing the report

Dependencies

I have defined the plan so that enough outputs are ready to work with into the next task.

The step of getting feedback is important. I need feedback from

- Porism who lead the app community for Open Referral UK
- Government API standards group

I will keep both these groups aware of my progress so that they are ready to feedback at the right time.

Project Management

I will track progress on a Kanban board

The planning phases will build a feature list for the API and client app. The feature lists will be prioritised using the agile MoSCoW technique to define a minimum viable product. The approach will ensure that I will have enough to evaluate all allow me to develop more if I have time.

Part 2

Initial Literature Review

Approach to literature searching

In this literature review I have looked for evidence to answer these questions:

- What is the ambition for open data?
- What do app developers need when using open data?
- What API frameworks have been used for accessing open data?
- Strengths and weaknesses of REST vs GraphQL?
- How is search implemented in REST vs GraphQL?

What is the ambition for open data?

The UK Government commissioned an independent review into the Power of Information in 2007 (Mayo & Steinberg, 2007) which proposed giving citizens and organisations better access to information held by government by republishing information in open standards or as web services.

This started an open data movement. The most recent Government Transformation Strategy (HMG, 2017) aims to release open government data to spur innovation and economic growth.

So the ambition of open data from government is to encourage developers to use government data in their apps to provide people with information that will improve their lives.

What do app developers need when using open data?

In (Susha et al., 2015) the authors established that one of the key success factors for open data is to provide Application Programming Interfaces (API's) so that 'open data' can become 'open services'.

One of the three case studies they looked at was Open Data State of New York, where they noted that it went beyond the release of datasets by also providing services to developers by offering an API endpoint for every dataset.

So whereas open data started as being provided as snapshots of large datasets that were published periodically, app developers need instant access to small amounts of up-to-date data.

What API frameworks have been used for accessing open data?

The World Wide Web Consortium in their Linked Open Data Glossary (W3C, 2013), refer to Tim Berners-Lee's five-star deployment scheme, in which the ultimate aim is to provide a SPARQL endpoint for 'open data'. SPARQL enables a client to construct complex searches and return only the fields that are requested. However, SPARQL is typically used on snapshots of large amounts of data that changes infrequently. For example (HMG, 2017) in their Government Data Strategy intend to build an Integrated Data Platform for government, which will use linked data, to enable policymakers to draw on the most up-to-date evidence.

In (Spasev et al., 2020) the authors explain that SOAP was used in the past, which is a message protocol that allows communication between remote application elements, but that SOAP did not fit the need of the web and mobile application market and was replaced with the REST API architecture. This is

because SOAP has to use XML which in general has larger size then JSON which is commonly used in REST, as response size is a large factor when designing a mobile app REST became the standard.

To support more real-time access to data for app developers, government has published guidance on setting up REST APIs (HMG, 2019). Roy Fielding defined REST in his 2000 PhD dissertation (Fielding, 2000) to reuse existing HTTP features such as content-types, caching and status codes. Therefore REST APIs provide a simple interface to query or update data and get a predefined response back.

Surwase (2016) explains that a REST API can be called from any HTTP client, including client-side JavaScript code that is running in a web browser. For GET requests the resource is identified in the URL path, and the results can be further qualified or filtered using the query string. The paper explores schema definition languages that can be used to describe the structure of the response bodies that are returned from an operation. The response to an operation is fixed by the server and cannot be defined by the client.

As an alternative to REST, GraphQL was developed internally by Facebook in 2012 before being publicly released in 2015 (Facebook, 2018), as "a query language designed to build client applications by providing an intuitive and flexible syntax and system for describing their data requirements and interactions". GraphQL allows the client to specify the fields that they want to be returned so this prevents excessively large amounts of data from being returned, which can be the case in REST because the fieldset is determined by the API.

Strengths and weaknesses of REST vs GraphQL?

(Spasev et al., 2020) compared the strengths and weakness of REST and GraphQL and found that, the two most important advantages of GraphQL are the reduction in the number of API calls to endpoints and the reduction in the number of fields that are returned. This allows a client to only receive back the exact field that is needed for the application, this reduces the size of the returned data and reduces the amount of processing needed by the app.

They explain that with REST there is a tendency to both over and under fetch data. It over fetches because the fieldset that is returned is set by the API and therefore the API is often developed so it contains all of the information that might be required. GraphQL on the other hand, allows the client to specify which fields are required which reduces the size of the data returned. REST also under fetches data where related data is stored on different endpoints, requiring multiple API calls to receive all of the data required. GraphQL on the other hand has a single API endpoint where data from the whole service can be accessed in a single call.

Brito et al. (2019) migrated some REST API into GraphQL and their key finding was that GraphQL can reduce the size of the document returned by 94%. This was achieved by shifting the decision on the precise data returned, from the server to the client. They explain that this is particularly important for mobile applications, which often face limited bandwidth and speed, and is being implemented by well-known web applications such as GitHub and Pinterest.

How is search implemented in REST vs GraphQL?

Both REST and GraphQL are good at returning a particular resource, for example, if the ISBN of a book is known, both REST and GraphQL could easily return information for just that one book. They are less good at returning a result set for a complex search, for example, books with a title containing 'Fish' written before 2005 together with books with a title containing 'Bird' written before 2008.

Performing a Google search for “Implementing search in REST API” returns a lot of results of people asking how to do it, but no commonly agreed answers as experts think that proposed solutions would break the principles of REST. I was unable to find any journal articles that propose a solution, but web forums focused on two types of solution, for which the experts had objections.

- Using a URL query string to contain the search parameters
    - Hard to represent ANDs and ORs and operators such as greater, less, contains
    - Query strings over 2000 characters can be a problem
- Using a HTTP POST instead of a HTTP GET to send a JSON object containing the search parameters
    - POST should always create a new entity, so using it in this way is not good practice

I was unable to find any journal articles about implementing a complex search in GraphQL. A Google search finds some ad-hoc libraries where a JSON object can be sent as a search argument.

Conclusions

Government want to encourage app developers to create web and mobile applications that improve people’s lives, by using ‘open data’. Developers need that data to be high quality so that it is complete and up-to-date, so a large snapshot that needs to be constantly downloaded won’t do. Instead, app developers need access to data via a lightweight API so that they can get instant results from the most up-to-date data.

API access was originally done through tools such as SPARQL and SOAP that require large amounts of experience to use efficiently, but neither of there was appropriate for web and mobile access to volatile data. Government has produced guidance for ‘open data’ publishers, to use REST APIs for this type of service.

More recently GraphQL, which was originally specified by Facebook, has emerged as an alternative API framework to address some weaknesses in REST.

Although REST is easy to use, the contents of the response are defined by the API which can mean that more data than is required is returned and that more than one call might be required to retrieve all of the necessary information. GraphQL can combine results into a single API call and returns only the fields that the client asks for.

A drawback of REST is that any filtering parameters have to be put in the query string, which limits the amount of complex searching that can be achieved in a single call. An app may have to make many calls for ANDs and ORs and then match the results at the client end.

The research has found that REST is not good at complex searches, but that some people have succeeded using GraphQL although it doesn’t have an official built-in approach.

The project will take some existing REST APIs and implement them as GraphQL APIs to analyse if the improvement to complex searching would enable app developers to create a better experience for their end-users.

References

Brito, G., Mombach, T., & Valente, M. T. (2019). Migrating to GraphQL: A Practical Assessment. *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 140–150. https://doi.org/10.1109/SANER.2019.8667986

Facebook. (2018). *GraphQL*. https://spec.graphql.org/

Fielding, R. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

HMG. (2017). *Govenment Transformation stategy*. HMG. https://www.gov.uk/government/publications/government-transformation-strategy-2017-to-2020/government-transformation-strategy-better-use-of-data

HMG. (2019). *API technical and data standards*. https://www.gov.uk/guidance/gds-api-technical-and-data-standards#use-restful

Mayo, E., & Steinberg, T. (2007). *The Power of Information*. *June*, 1–57. https://ntouk.files.wordpress.com/2015/06/power_information.pdf

Spasev, V., Dimitrovski, I., & Kitanovski, I. (2020). An Overview of GraphQL : Core Features and Architecture. *ICT Innovations Conference Web Proceedings*. https://repository.ukim.mk/handle/20.500.12188/9488

Surwase, V. (2016). REST API Modeling Languages -A Developer's Perspective. *IJSTE -International Journal of Science Technology & Engineering*, *2*(10), 634–637.

Susha, I., Zuiderwijk, A., Charalabidis, Y., Parycek, P., & Janssen, M. (2015). Critical Factors for Open Data Publication and Use: A Comparison of City-level, Regional, and Transnational Cases. *JeDEM - EJournal of EDemocracy and Open Government*, *7*(2), 94–115. https://doi.org/10.29379/jedem.v7i2.397

W3C. (2013). *Linked Data Glossary*. W3C. https://www.w3.org/TR/ld-glossary/