Graphs with GraphQL in Timbuctoo

Martijn Maas, Jauco Noordzij, Menzo Windhouwer
KNAW Humanities Cluster - Digital Infrastructure
{martijn.maas, jauco.noordzij, menzo.windhouwer}@di.huc.knaw.nl

Timbuctoo [1] is the RDF [2] store used and developed by HuygensING/HUC DI. One of its biggest features is that it generates a GraphQL-schema [3] based on the RDF-dataset the user uploaded, while other systems require you to manually create the schema. Timbuctoo makes use of the graphql-java library [4].

A large chunk of the schema is literally generated by the implicit schema of the RDF dataset. For each unique RDF type a GraphQL-type is generated. For each predicate a property is added to to the corresponding GraphQL-type. For the predicate value types there is a property created with a type that contains both the value and the rdf-type of the property. For predicates that reference another resource a property is created with the type of the referenced resource.

For both types of predicates Timbuctoo's behaviour will change when a reference has multiple predicates of the same type. It will create a List property, that can show all the data or can show the data in chunks with help of a cursor.

For predicates between resources Timbuctoo will add both a property to the type of the subject and a property, where the name is prefixed with inverse, to the type of the object.

The Timbuctoo GraphQL schema can be changed in two ways. The first one is by uploading new RDF files. The other way is when a user explicitly changes the schema. Both ways can cause properties or types to be added. They can also cause single valued properties to become multivalued properties. To be backwards compatible Timbuctoo will also leave the single valued property. However, it will add a GraphQL deprecation warning for the property to the schema. Multivalued properties will never become single valued properties, because that would break the backwards compatibility.

GraphQL has basic support for CRUD actions, but the retrieval action focuses on searching specific fields. There is no defined way to query the data in more advanced ways, e.g., boolean predicates, full text search or facets. To enable this Timbuctoo has defined a GraphQL query where ElasticSearch [5] queries can be used to filter which resources to show. For programmers it is possible to create a Timbuctoo plugin for their favorite search engine.

The main reason for Timbuctoo to use a GraphQL API is its discoverability. Timbuctoo first used plain REST. It can be made discoverable, but you have to do work for it. GraphQL is discoverable automatically, because it also exposes its schema. Also GraphQL allows us to control the performance of the API, i.e., we know and control which queries to our datastore can be generated, opposed to arbitrary queries which would be possible if we provided a SPARQL endpoint.

GraphQL is an excellent language to define a schema and for selecting which properties to retrieve. It provides a good basis for accessing graphs in Timbuctoo. However, we did have to define some extensions to cover all aspects, e.g., more advanced searching.

[1] Huygens ING Institute. *Make the most of Humanities Research Data*. timbuctoo.huygens.knaw.nl Accessed on October 12, 2018.
[2] W3C. *Resource Description Framework*. w3.org/RDF Accessed on October 12, 2018.
[3] Facebook. *GraphQL*. graphql.org Accessed on October 14, 2018.
[4] Marek, Andreas. *graphql-java library*. github.com/graphql-java/graphql-java Accessed on October 12, 2018.
[5] Elastic. *ElasticSearch*. elastic.co/guide/en/elasticsearch/reference/current/getting-started.html Accessed on October 12, 2018.