GraphQL / GraphDefinition

Grahame Grieve
GraphQL and GraphDefinition

• Both are concerned with scanning across resources

• GraphQL
  • Select a subset of information across a graph of resources
  • Return content is not actually resources

• GraphDefinition
  • Specify a set of rules about relationships between resources
  • May be used to retrieve a set of resources
Problem Statement

• Well described modular schema + CRUD Operations
• Client applications built against the API
• User interface applications require a set of resources
• To build a view e.g. List of encounters
  • For each encounter, fetch the patient to get their name
  • For each encounter, fetch the consulting physician to get their name
  • For each encounter, fetch the ward location to get it’s name
  • etc
• Similar for almost every operation
Getting a set of resources (1)

• Fetch the encounters
• Iterate the encounters fetching the related resources
• Cache up local copies of fetched resources

• Price: bandwidth, latency * N, cache management
Getting a set of resources (2)

- Fetch the encounters
- Use _include to fetch the related resources

- Price: bandwidth (up), latency * 1 (down)
Getting a set of resources: ideal

• Just fetch what you want to display:
• Fetch the elements from encounter that you want to display
• Get the server to replace the links to the other resources with your choice of display for them

• No work on the client...

• Price: bandwidth (down), latency * 1
GraphQL

• A language to tell the server what you want
• Query against the graph of resources it has
• Get just the structure you want returned
Simple GraphQL example

```
{  
  name { text given family }  
}
```

```
{
  "data" : {
    "name" : [{
      "given" : ["Peter","James"],
      "family" : "Chalmers"
    },{
      "given" : ["Jim"]
    },{
      "given" : ["Peter","James"],
      "family" : "Windsor"
    }]
  }
}
```
Polymorphic Fields

```json
{
  valueQuantity { value unit }
}

{
  "data" : {
    "valueQuantity":{
      "value":185,
      "unit":"lbs"
    }
  }
}
```
Lists

• fhirpath - a FHIRPath statement selecting which of the subnodes is to be included
• [field] - the name of a sub-property with a specified value that must be matched for the field to be included
• _offset - specify the offset to start at for a repeating element (see below)
• _count - specify how many to elements to return from a repeating list
Polymorphic Fields

```json
{
  name(use: official)
  {
    text given family
    myext: extension(url: "[url]")
    {
      value : valueString
    }
  }
}
```

```json
{
  "data": {
    "name": [{
      "given": ["Peter","James"],
      "family": "Chalmers"
    }],
    "myext": {
      "value": "some value"
    }
  }
}
```
Lists

```json
{
  name(fhirpath: "family.exists()")
  { text given family }
}
{
  "data":
  {
    "name": ["Peter","James"],
    "family": "Windsor"
  },
  {
    "given": ["Peter","James"],
    "family": "Chalmers"
  }
}
```
References

• `{ resource }` – walk into the target of a resource
  • Optional – return an error if not resolvable
  • Type selector – specify type of target (always required, may be parameter)

• Can also search on reverse references, and add other resources that refer to the focus resource
Lists

```json
{
  subject {
    resource {
      ...on Patient {
        birthDate
      }
    }
  }
  code {coding {system code} }
}
```

```json
{
  "data" : {
    "subject":{
      "resource":{
        "birthDate":"1974-12-25"
      }
    },
    "code":{
      "coding":[
        {
          "system":"http://loinc.org",
          "code":"29463-7"
        }
      ]
    }
  }
}
```
Managing a list of Resources

• On the server root, or an a particular resource
• By search parameter
• 2 forms:
  • List: Simple
  • Connection: Complex - With cursors
Simple GraphQL example

```
[base]/$graphql?query={Patient(id:example){id,name{given,family}}}
{
    Patient(id: example)
    { id, active }
}
```
Search / List

```json
{
  ConditionList
    (clinical_status: relapse, patient: example)
  { id, clinicalStatus }
}
```

```json
{
  "data" : {
    "ConditionList" : [{
      "id" : "100",
      "clinicalStatus" : "relapse"
    },{
      "id" : "100",
      "clinicalStatus" : "relapse"
    }]
  }
}
```
Search / List

```json
{
   ConditionConnection
      (clinical_status: active, patient: example) {
      count offset pagesize
      edges {
         mode, score, resource
         { id, active }
      }
      first previous next last
   }
}
```

```json
{
   "data": {
      "ConditionConnection": {
         "count": 50,
         "offset": 0,
         "pageSize": 25,
         "next": "45f9ada8-db37-4498-ba7d...:3"
      }
   }
```
Flattening

- Drop spacer nodes
- Break Lists up
Flattening

```json
{
  identifier @flatten
  { system value },
  active
  name @flatten
  { text given family }
}

{
  "data": {
    "system": ["urn"],
    "value": ["12345"],
    "active": true,
    "given": ["Peter", "James"],
    "family": ["Chalmers", "Windsor"]
  }
}
```
GraphQL

• Powerful tool for client to reshape data to suit their own purposes
• May be particularly relevant where data is passed to analysis tools
• Supported by
  • test.fhir.org
  • HAPI
  • Assymetrik
  • Others ?
GraphDefinition

• A definition of a set of relationships between resources
• A fixed starting point (known type)
• Defines a nested series of links
  • What they are
  • Cardinality rules
  • Direction: Forwards / backwards
  • Target Types / Profiles
  • Rules about compartment consistency
GraphDefinition Example

Patient(http://hl7.org/fhir/us/core/Patient) {
    managingOrganization cardinality 0..1 'managing org' :
        Organization(http://hl7.org/fhir/us/core/Organization) {
            endpoint : Endpoint
        };
    Group {
        item : Patient
    },
    generalPractitioner : Organization,
    search Observation?patient={ref} cardinality 0..10 'Observations for the patient' {
        performer : Practitioner,
        related.where(type='has-member').target : Observation require matching Patient,
        related.where(type='derived-from').target : Observation where identical Patient
Uses of GraphDefinition

• Summarize a set of profiles on resources
  • e.g. documentation in an Implementation Guide

• Define a graph of resources to return in a query
  • e.g. GET [base]/MedicationDispense/example/$graph?graph=med-package

• Define a graph of resources to include in a document
  • e.g. GET [base]/Composition/example/$document?graph=example

• Document rules about the relationship between a set of resources
  E.g. must all resources concern the same patient?
  • E.g. $validate?graph=med-package
GraphDefinition

• Is over the horizon for most users
• Will become more important as people get further into their development processes
GraphQL and GraphDefinition

• Both are concerned with scanning across resources
• GraphQL
  • Select a subset of information across a graph of resources
  • Return content is not actually resources
• GraphDefinition
  • Specify a set of rules about relationships between resources
  • May be used to retrieve a set of resources
  • But has other uses too