Master Data API story:

REST is dead, long live GraphQL?
Solution architect in Customers & Insights, SEB Global Services

Currently focuses on data and everything surrounding it

Data Engineer in Information platform, SEB Global services

Passionate about coding
In love with cryptocurrencies
Agenda

- MDM definition & our strategic solution
- Talk about GraphQL…
- … and REST
- Demo
Optional.of(listen).orElse(handsOn)

Want to listen and get some details?
Stay tuned!

Itchy hands?

http://mdm.seb/graphiql
http://mdm.seb/voyager
Provide master data with high quality to master data consumers

Master Data Management Platform

http://mdm.seb/graphiql
http://mdm.seb/voyager
Our MDM pillars

Master Data Management Platform

Customer  Product  Arrangement

http://mdm.seb/graphiql
http://mdm.seb/voyager
Why MDM?

- Internal system 1
- Internal system 2
- Internal system 3
- Internal system 4
- Internal system 5
- Internal system 6

Mainframe

Reporting system
BUZZ: REST or... GraphQL?

REST in Peace. Long Live GraphQL | Hacker News
https://news.ycombinator.com/item?id=14839576
Jul 24, 2017 - As the article points out at the end, there are very real engineering tradeoffs with

REST 2.0 Is Here and Its Name Is GraphQL — SitePoint
https://www.sitepoint.com/rest-2-0-graphql/
May 17, 2017 - What is GraphQL, and why should you use it? Michael Paris looks at this extremely popular

Is GraphQL The Future? - Artsy Engineering
artsy.github.io/blog/2018/05/08/is-graphql-the-future/ Øversätt den här sidan
8 maj 2018 - I have seen the future, and it looks a lot like GraphQL. ... It is borderline heresy in some

Goodbye REST, Hello GraphQL? | InfoQ
https://www.infoq.com/presentations/rest-graphql
Oct 4, 2018 - Cory House discusses GraphQL, what makes it attractive, how GraphQL ... He has

Is GraphQL The End of REST Style APIs? | Nordic APIs |
https://nordicapis.com/is-graphql-the-end-of-rest-style-apis/
Mar 9, 2017 - What's the hubbub with GraphQL? GraphQL performs select functions better than REST,

http://mdm.seb/graphiql
http://mdm.seb/voyager
Quick poll

http://mdm.seb/graphiql
http://mdm.seb/voyager
Who uses REST?

http://mdm.seb/graphiql
http://mdm.seb/voyager
Who uses GraphQL?

http://mdm.seb/graphiql
http://mdm.seb/voyager
What is GraphQL?

“GraphQL is a query language for your API, and a server-side runtime for executing queries by using a type system you define for your data.”

(http://graphql.org/learn)
GraphQL: intro

Created for Facebook internal use in 2012, open sourced in 2015

Moved to GraphQL foundation, hosted by non-profit Linux Foundation on 2018 November

Data query language for your api

Centered around a schema with continuous evolution in mind

Not a graph database!

Language agnostic. Database agnostic

Uses single HTTP endpoint

Used by variety of companies in production already
Quick peek

http://mdm.seb/graphiql
http://mdm.seb/voyager
GraphQL: high level overview

- **Type**

- **Schema. 3 operation types**
  - Query (data fetching)
  - Mutation (data modification)
  - Subscription (web hook like)

- **Resolvers**
GraphQL: key features

- One endpoint for API
- Schema required
- Client determines which data he/she needs, not the server
- Hierarchical
- Strongly typed
- Can map to multiple data sources
- Easy API versioning
- Monitor each field usage
- GraphQL endpoints stitching
- GraphiQL as documentation
What is GraphQL Schema Language?
It is a shorthand notation to succinctly express the basic shape of your GraphQL schema and its type system.

What does it look like?
Below is an example of a typical GraphQL schema expressed in shorthand.

```graphql
# define Entity interface
interface Entity {
  id: ID!
  name: String
}

# define custom URL scalar
scalar Url

# User type implements Entity interface
type User implements Entity {
  id: ID!
  name: String
  age: Int
  balance: Float
  is_active: Boolean
  friends: [User]!
  homepage: Url
}

# root Query type
type Query {
  me: User!
  friends(limited: Int = 10): [User]!
}

# custom complex input type
input ListUsersInput {
  limit: Int
  since_id: ID
}

# root mutation type
type Mutation {
  createUser(params: ListUsersInput): [User]!
}

# GraphQL root schema type
schema {
  query: Query
  mutation: Mutation
  subscription: ...!
}

Input Arguments

Basic Input
```
```graphql
  type Query {
    users(limited: Int): [User]
  }
```

Input with default value
```
  type Query {
    users(limited: Int = 10): [User]
  }
```

Input with multiple arguments
```
  type Query {
    users(limited: Int, sort: String): [User]
  }
```

Input with multiple arguments and default values
```
  type Query {
    users(limited: Int = 10, sort: String = "asc"): [User]
  }
```

Input Types
```
input ListUsersInput {
  limit: Int
  since_id: ID
}
```

Type Definitions
```
scalar Int
scalar String
scalar Boolean

interface User {
  id: ID!
  name: String
  age: Int
  balance: Float
  is_active: Boolean
  friends: [User]!
  homepage: Url
}
```

Type Modifiers
```
String
String!
String
String!
String
String!

enum EnumType {
  Value1
  Value2
}
```

Built-in Scalar Types
```
Int
Float
String
Boolean

union UnionType {
  Field1
  Field2
}
```

Interfaces
```
interface User {
  id: ID!
  name: String
}

interface User {
  id: ID!
  name: String
}
```

Union of one or more Objects
```
type Foo {
  name: String
}

type Bar {
  is_bar: String
}
```

Unions
```
union SingleUnion = Foo
union MultipleUnion = Foo | Bar
```

Enumerations
```
enum USER_STATE {
  NOT_FOUND
  ACTIVE
  INACTIVE
  SUSPENDED
}
```

Enums
```
type User {
  name: String
  homepage: Url
}
```
What about REST?

REST and GraphQL side by side

http://mdm.seb/graphiql
http://mdm.seb/voyager
GraphQL

Centered around **data**

Provides **single endpoint** for consumers

Reduces **network hops** – allows to consume several related data items

**Client** needs **first** – pick your payload

**Custom caching** strategy needed

REST

Centered around **resource**

**Different endpoints** for different resources

**Possible** round-trips due to **under-serving**

You get what is given - possible **over-fetching**

HTTP **caching by default**
<table>
<thead>
<tr>
<th>GraphQL</th>
<th>REST</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Schema evolution</strong> with new fields addition and deprecation</td>
<td><strong>Need to version</strong> your <strong>API</strong> – which <strong>strategy</strong> to use?</td>
</tr>
<tr>
<td><strong>Custom error handling</strong> – what if one of three backend calls fails?</td>
<td><strong>More consistency and predictability</strong> by using status codes and HTTP verbs</td>
</tr>
<tr>
<td>GraphQL is <strong>by design</strong> natively discoverable</td>
<td><strong>HATEOAS</strong> allows easy to &quot;discover&quot; resource relations whilst using the API</td>
</tr>
</tbody>
</table>
Live demo

Any similarity to actual data, relations, structure or code is purely coincidental
GraphQL is not a silver bullet
Where to now?

Introduce GraphQL incrementally to MDM API

Start from using queries

Exposé types that have different attributes needed for different clients

Expand using GraphQL when API client number grows

Have both – REST and GraphQL in parallel. For now.
Contact us

✉️ augustas.nekrosius@seb.se

LinkedIn: augustas-nekrosius

✉️ irma.spudiene@seb.se

LinkedIn: irma-spudiene