Better APIs with GraphQL

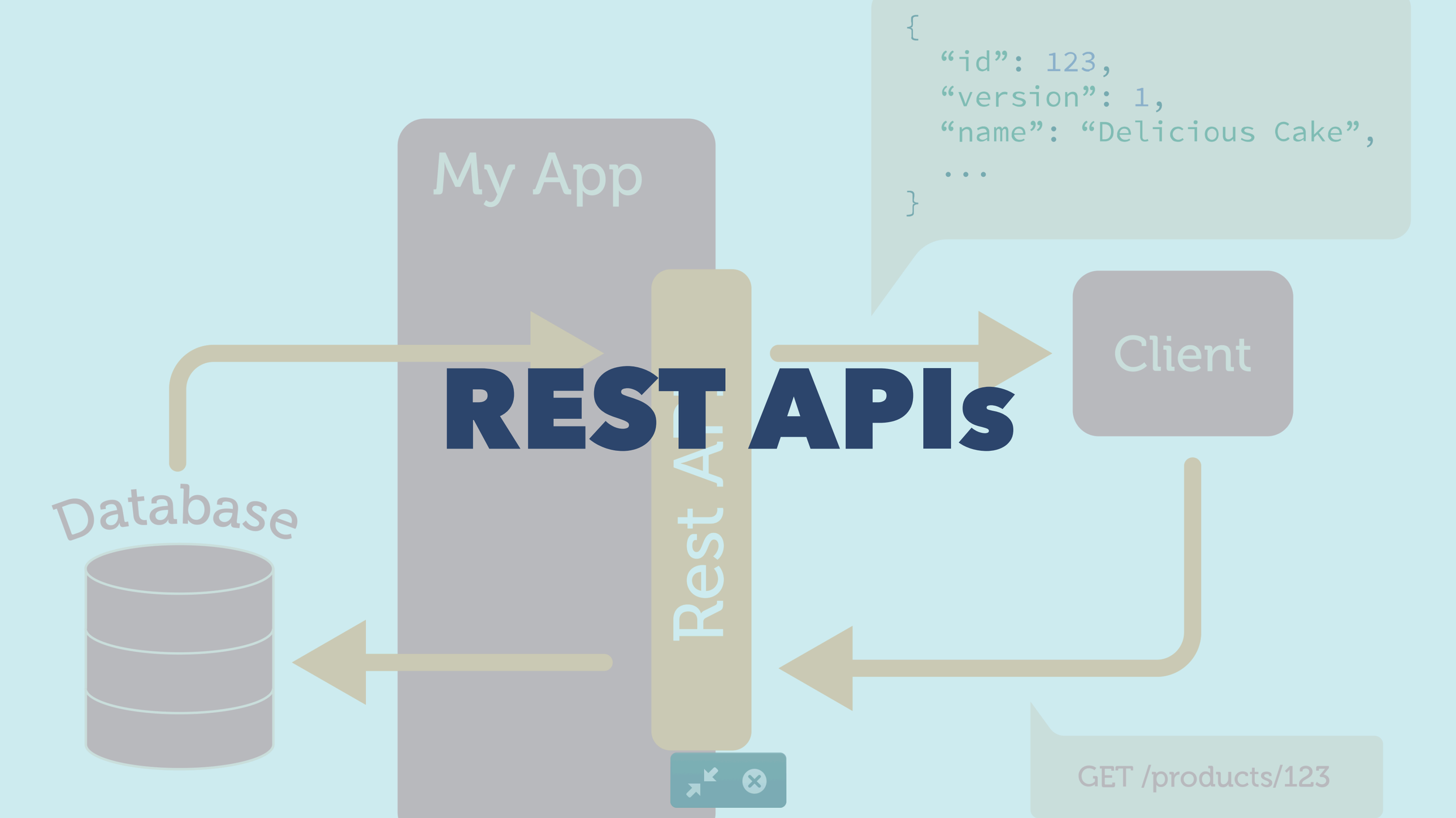
Josh Price

github.com/joshprice

@joshprice

Agenda

- Understand GraphQL and why you'd use it
- Build a simple schema
- Run queries against the schema
- Understand important GraphQL concepts
- Summarise client options


{
"id": 123,
"version": 1,
"name": "Delicious Cake",
...
}
My App
Rest API
Client
Database
GET /products/123

REST APIs

REST is great

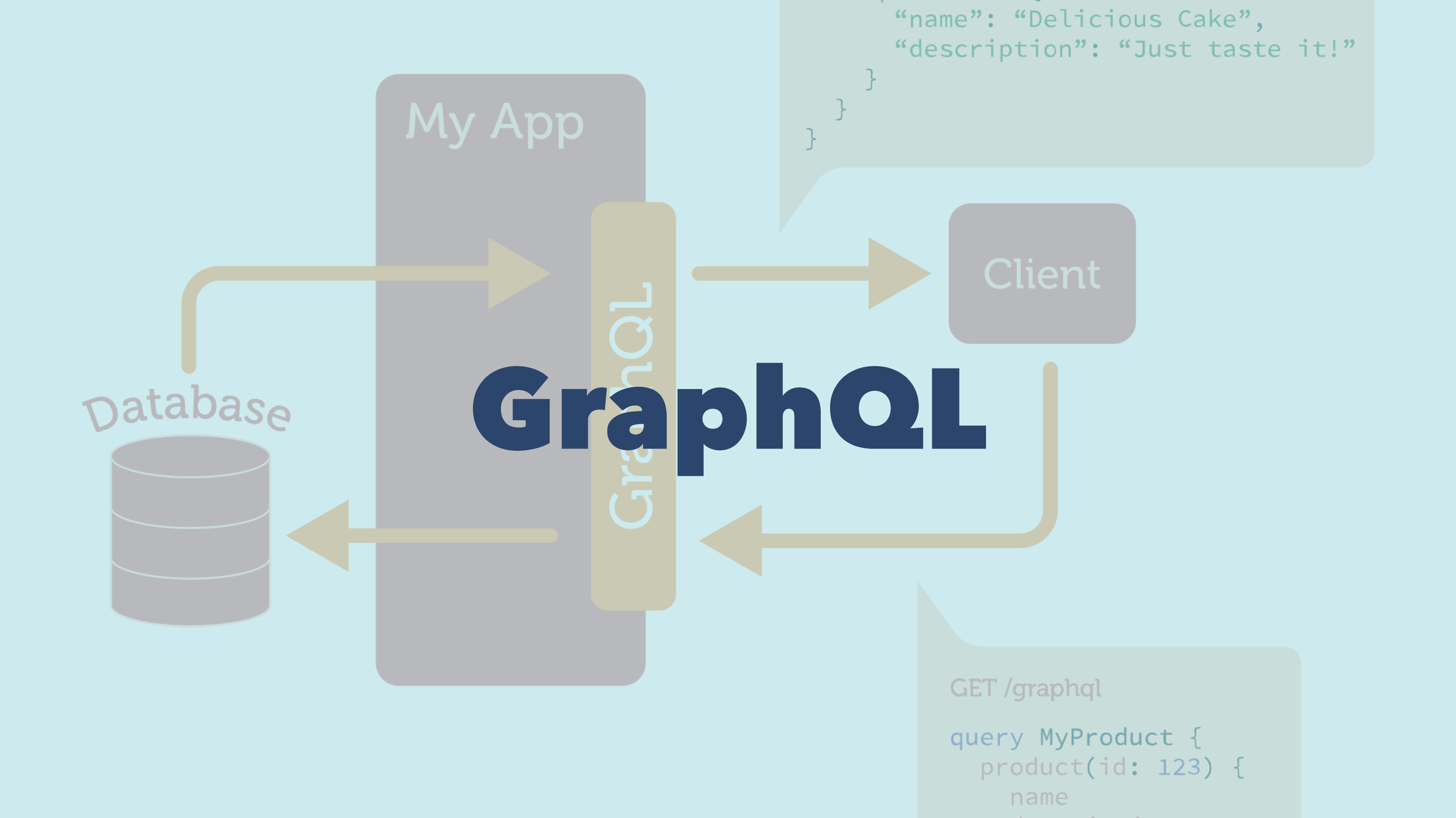
REST is *hard*

Cargo Cults

Common Problems

Overfetching
Underfetching

Internal APIs have strong contracts between client and server


"name": "Delicious Cake",
"description": "Just taste it!"
}
}
}
My App
GraphQL
Client
Database
GraphQL
GET /graphql
query MyProduct {
product(id: 123) {
name

What is GraphQL?

- Language for defining schemas, types & queries
- Developed by Facebook in 2012
- Used to improve mobile app performance
- Serves **300 billion+** requests per day

Open Source

- Open sourced in **July 2015**
- Specification
  - **facebook.github.io/graphql**
- Reference Implementation
  - **github.com/graphql/graphql-js**
- Relay released in **August 2015**

GraphQL Server Implementations

- JavaScript reference
- Ruby / Python / PHP
- Java / Scala (Sangria)
- .NET
- Elixir / Go / Haskell / OCaml
- etc...

GraphQL Misconceptions

- Not really about "graphs"
- A *specification* for client/server interaction
- Language independent
- Assumes nothing about:
  - transport
  - message protocol
  - data store

Exhibit A: REST API

Fetch user name and friends names with 2 requests

```
GET /users/1
GET /users/1/friends
```

or a single request

```
GET /users/1/friends?include=user.name,friend.name
```

Exhibit B: GraphQL API

```
{
  user(id: 1) {
    name
    friends(first: 1) {
      name
    }
  }
}
```

```
"data": {
  "user": {
    "name": "Josh",
    "friends": [{
      "name": "James"
    }]
  }
}
```

Better Mental Model

Strongly typed

Single Endpoint

Single Unambiguous Query

Consumer Driven Contracts

Less Versioning

Self Documenting

Performance

Let's build our first Schema

Query and Mutation Roots

```
type Query {
  me: User
  user(id: Int): User
}

type Mutation {
  createPost(title: String!): Post
  createComment(message: String!): Comment
}
```

Object Types and Enums

```
type User {
  name: String
  profilePicture(size: Int = 50): ProfilePicture
  friends(first: Int, orderBy: FriendOrder): [User]
}

enum FriendOrder { FIRST_NAME, LAST_NAME, IMPORTANCE }
```

More Types

```
type ProfilePicture {
  width: Int
  height: Int
  url: String
}

type Event {
  name: String
  attendees(first: Int): [User]
}
```

Simplest Query

```
{
  me {
    name
  }
}
```

```
{
  "data": {
    "me": {
      "name": "Josh Price"
    }
  }
}
```

Deeply Nested Query

```
{
  me {
    name
    profilePicture {
      url
    }
    friends(first: 1) {
      name
    }
    events(first: 1) {
      name
      attendees(first: 1) {
        name
      }
    }
  }
}
```

```
{
  "data": {
    "name": "Josh Price",
    "profilePicture": {
      "url": "http://cdn/josh_50x50.png"
    },
    "friends": [{
      "name": "James Sadler"
    }],
    "events": [{
      "name": "Afterparty!",
      "attendees": [{
        "name": "Jenny Savage"
      }]
    }]
  }
}
```

How do we fetch data?

Resolvers

- Your own functions
- Could use in memory data
- Call any data store
- Proxy REST APIs
- Call existing services
- GraphQL doesn't care
- Resolver "Middleware" is possible (Auth, Logging, etc)

User Type with JS Resolvers

```
new GraphQLObject({
  type: "User",
  fields: {
    name(user) {
      return user.name
    },
    profilePicture(user, {size}) {
      return getProfilePicForUser(user, size);
    },
    friends(user) {
      return user.friendIDs.map(id => getUser(id));
    }
  }
});
```

Mutations Modify Data

```
mutation {
  acceptFriendRequest(userId: 345124) {
    user {
      friends { count }
    }
  }
}
```

```
mutation {
  rsvpToEvent(eventId: 134624, status: ATTENDING) {
    event {
      invitees { count }
      attendees { count }
    }
  }
}
```

Setup GraphQL Express

```
import { Schema } from './schema.js';
import graphqlHTTP from 'express-graphql';
import express from 'express';

const app = express();

app.get('/', function(req, res) {
  res.redirect('/graphql');
});

app.use('/graphql', graphqlHTTP({ schema: Schema, graphiql: true }));

app.listen(3000);
```

Relay

- Each view component declares query fragment
- Relay batches all data req'ts for render tree
- Sends single query
- Handles caching using global IDs
- Relies on schema conventions for metadata

Client-Side Alternatives

- ApolloStack Client
  - React + Native
  - Angular 2
  - Redux support
- Lokka
  - Simple

Gotchas

- Arbitrary Queries
  - Could be slow if deeply nested (friends of friends...)
  - Complexity analysis
  - Query depth
- Batching resolvers
  - Data Loader (JS)
  - GraphQL Batch (Ruby)

When to use?

- Use for internal APIs first, or limited external use
- Improve mobile (and desktop performance)
- Github has exposed their API externally
- Be careful exposing this to the world!
- Don't allow arbitrary queries from unknown clients

GraphQL Ecosystem Evolving Quickly

GraphQL Backend as a Service

- reindex.io
- graph.cool
- scaphold.io

Future - GraphQL Spec

- Push: Apps should reflect current state of world
- Subscriptions + Reactive Backend + RethinkDB
- Defer
- Stream
- Live queries
- GraphQL CATS

Subscriptions

```
subscription {
  createCommentSubscribe(storyId: $id) {
    comment {
      ...FBCommentFragment
    }
  }
}
```

Warning!

Experimental

Defer Directive

```
{
  feed {
    stories {
      author { name }
      title
      comments @defer {
        author { name }
        comment
      }
    }
  }
}
```

```
{
  "feed": {
    "stories": [{
      "author": { "name": "Lee Byron" },
      "title": "GraphQL is the Future"
    }, {
      "author": { "name": "Josh Price" },
      "title": "REST is old school"
    }]
  }
}
```

Defer - Comments arrive

```
{
  feed {
    stories {
      author { name }
      title
      comments @defer {
        author { name }
        comment
      }
    }
  }
}
```

```
{
  "path": ["feed", "stories", 0, "comment"],
  "data": [{
    "author": { "name": "Joe Bloggs" },
    "comment": "That blew my mind!"
  }, {
    "author": { "name": "Jenny Savage" },
    "comment": "I love it"
  }]
}
```

Stream Directive

```
{
  feed {
    stories @stream {
      author { name }
      title
      comments @defer {
        author { name }
        comment
      }
    }
  }
}
```

```
{
  "feed": {
    "stories": []
  }
}
```

Stream - First Story

```
{
  feed {
    stories @stream {
      author { name }
      title
      comments @defer {
        author { name }
        comment
      }
    }
  }
}
```

```
{
  "path": ["feed", "stories", 0],
  "data": [{
    "author": { "name": "Joe Bloggs" },
    "title": "That blew my mind!"
  }]
}
```

Live Directive

```
{
  feed {
    stories {
      author { name }
      title
      likeCount @live
    }
  }
}
```

```
{
  "feed": {
    "stories": [{
      "author": { "name": "Lee Byron" },
      "title": "GraphQL is the Future",
      "likeCount": 9
    }]
  }
}
```

Live - Likes update on backend

```
{
  feed {
    stories {
      author { name }
      title
      likeCount @live
    }
  }
}
```

```
{
  "path": ["feed", "stories", 0, "likeCount"],
  "data": 10
}
```

Resources

- **graphql.org**
- Github
  - **graphql/graphql-js**
  - **graphql/express-graphql**
- Steve Luscher talk **Zero to GraphQL**
- Awesome GraphQL (**chentsulin/awesome-graphql**)

Questions?

Thanks!