GraphQL

The new way people do data loading.

MIT 6.148 Jan 18, 2017

Sashko Stubailo
@stubailo
Open Source Lead, Meteor
Back in my day…

6.470    6.148
Question time

Web development experience
A BRIEF HISTORY OF WEB APPS

First way people did it: HTML from a web server to a browser
Starting 2012: Loading data from an API server to a single-page app, for better performance

- Better when you have complex UI interactions
- Better performance
- Easy to enforce security at API layer
- Easy to build a mobile app
A BRIEF HISTORY OF APPS

- JavaScript UI
- Native iOS app

Loading data from an API server to multiple clients in different environments
A BRIEF HISTORY OF APPS

Splitting into microservices, now you have multiple APIs and also multiple clients.
A BRIEF HISTORY OF APPS

Add an API gateway, now there’s a development bottleneck
Current most common practice: REST API

Read  
GET /todo-lists/5

Create  
POST /todo-lists

Update  
PATCH /todo-lists/5

• What are you going to get back from the server?
• Can you predict it based on the request?
• What if you need a todo list, and details about the items inside?
Fundamental tension between efficiently satisfying the data needs of specific clients, and keeping the number of endpoints manageable on the server.
THE FUTURE OF APPS

The solution: GraphQL as the translation layer between frontend and backend
Let clients specify their own data needs against the capabilities exposed by the server.
What does GraphQL look like?

```
// Schema

type Post {
  id: Int!
  title: String
  author: Author
  votes: Int
}

// Query

query {
  post(id: 5) {
    title
    votes
    author {
      id
      firstName
      lastName
    }
  }
}

// Result

{ "data": {
  "post": {
    "title": "GraphQL rocks",
    "votes": 5,
    "author": {
      "id": 17,
      "firstName": "Sashko",
      "lastName": "Stubailo"
    }
  }
}
```

Schema
Defines capabilities

Query
Defines requirements

Result
Shape guaranteed to match query
```json
{
  hero {
    name
    friends {
      name
      homeWorld {
        name
        climate
      }
    }
    species {
      name
      lifespan
      origin {
        name
      }
    }
  }
}

type Query {
  hero: Character
}

type Character {
  name: String
  friends: [Character]
  homeWorld: Planet
  species: Species
}

type Planet {
  name: String
  climate: String
}

type Species {
  name: String
  lifespan: Int
  origin: Planet
}
```
Not just a tool: An open source spec

SERVERS
- Node.js
- Ruby
- Python
- Scala
- Java
- Erlang

CLIENTS
- Plain HTTP request
- React
- Angular
- Vue
- Polymer
- Native iOS/Android

TOOLS
- API explorer (GraphiQL)
- Editor autocomplete
- Query validation
- Mocking
- Performance analysis
- Code generation

...and more
Think carefully about API needs

User Interface

Hardcode special endpoints

Backend

WITH REST AND OTHERS
WITH GRAPHQL

User Interface

Ask for data

Get the data

Backend
A better API experience
WORKING WITH GRAPHQL
Implementing a server means implementing the schema

```plaintext
{  
  hero {  
    name  
    friends {  
      name  
      homeWorld {  
        name  
        climate  
      }  
      species {  
        name  
        lifespan  
        origin {  
          name  
        }  
      }  
    }  
  }  
}  

type Query {  
  hero: Character  
}

type Character {  
  name: String  
  friends: [Character]  
  homeWorld: Planet  
  species: Species  
}

type Planet {  
  name: String  
  climate: String  
}

type Species {  
  name: String  
  lifespan: Int  
  origin: Planet  
}
```
Demo: Local app
Demo: GitHub API
WHO IS USING THIS?
Everyone is checking this out

See some more at graphql.org/users
GraphQL Summit

See 13 videos at graphql.com/summit
GraphQL APIs are inherently self-documenting
Query autocompletion in your editor

We’re collaborating with Facebook and others on a new language service to power GraphQL features across editors and IDEs.

jimkyndemeyer/js-graphql-intellij-plugin
Typed code generation: Swift, Java, TS, Flow

```typescript
query HeroName($episode: Episode) {
  hero(episode: $episode) {
    ...DescribeHero
  }
}

fragment DescribeHero on Character {
  name
  appearsIn
}

export type Episode =

export interface HeroNameQueryVariables {
  episode: Episode | null;
}

export interface HeroNameQuery {
  hero: DescribeHeroFragment;
}

export interface DescribeHeroFragment {
  name: string;
  appearsIn: Array<Episode | null>;
}
```

The above query combined with schema information outputs the type definitions on the right.
Easy to bind data to your UI with Apollo

// The `graphql` wrapper executes a GraphQL query and makes the results available on the `data` prop of the wrapped component (PostList here)

```javascript
export default graphql graphql(
  query allPosts {
    posts {
      id
      title
      votes
      author {
        id
        firstName
        lastName
      }
    }
  }
)(PostList);
```

Visit dev.apollodata.com for more code snippets.
Another benefit of using a sophisticated client is integrated tooling to understand your app.

Chrome dev tools for Apollo
Learn more

This tech is going to be part of at least one of your future jobs or internships.

- The main site about the specification: graphql.org
- Our website for GraphQL things: dev.apol lodata.com
- Our blog about GraphQL: dev-blog.apol lodata.com
- Try a tutorial: learnapollo.com
- Feel free to email me: sashko@meteor.com

If you want to get into GraphQL, open source, or developer tools in general we are hiring and also looking for contributors to work with. We’re having our first-ever contributor week starting Jan 31!

Please reach out to me directly!