

GraphQL and Go Best Friends?



About Me!

- I have a big dog! 🐶
- Java, JavaScript and now Go Developer
- I have seen SOAP, EJB, RMI, Corba, REST, gRPC, Websocket and GraphQL APIs... 😅
- I work at Manifold! Please take some stickers
- First time talking about GraphQL

@seriousben



One marketplace. Every ecosystem.

manifold

Javascript (Frontend) - Remote

Go (Backend) - Remote

manifold.co/careers

Announcing Marketplace-as-a-Service for developer platforms

Improve adoption and user experience with an easy-to-integrate marketplace of developer tools and services.

[Read announcement](#)

What is this talk about?

What is this talk about?

- Why GraphQL?

What is this talk about?

- Why GraphQL?
- What is GraphQL?

What is this talk about?

- Why GraphQL?
- What is GraphQL?
- How to GraphQL in go?

What is this talk about?

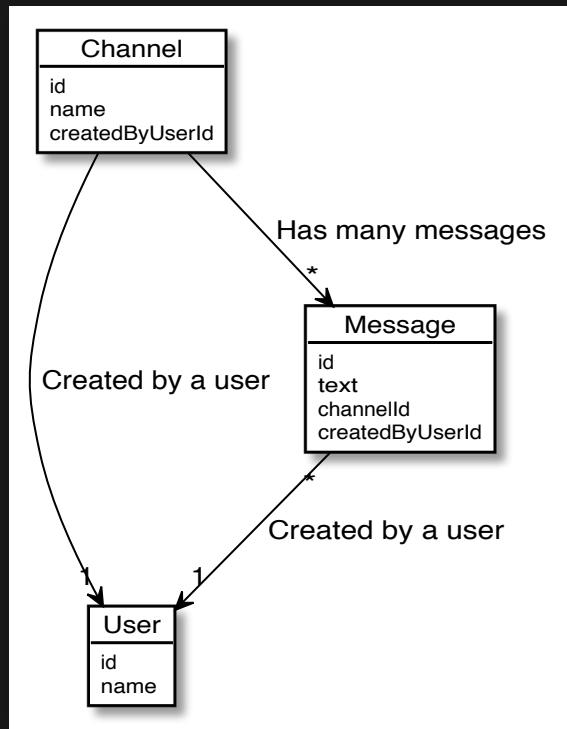
- Why GraphQL?
- What is GraphQL?
- How to GraphQL in go?

Shaking some preconceptions

You don't have to be RESTful 😊

Our Example Application

A better and much faster messaging App



Why GraphQL?

Before getting into GraphQL let's look at some REST API best practices.

RESTfulness: Granularity

```
GET /api/v1/messages
```

```
[ {  
    "id": "msg-id",  
    "text": "msg-content"  
} ]
```

RESTfulness: Leverage HTTP verbs

```
POST /api/v1/messages
```

```
GET /api/v1/messages/:id
```

```
PATCH /api/v1/messages/:id
```

```
PUT /api/v1/messages/:id
```

```
DELETE /api/v1/messages/:id
```

RESTfulness: URLs are resources not actions

```
PATCH /api/v1/messages/:id
```

Not:

```
POST /api/v1/update-message
```

RESTfulness: Resources are represented by an unique URL and a format.

```
GET /api/v1/messages/:id  
Accept: application/json
```

```
{  
  "id": "id",  
  "text": "msg-content"  
}
```

RESTfulness: Discoverability

Hypermedia As The Engine Of Application State (HATEOAS)

```
GET /api/v1/channels/:id
```

```
{
  "id": "id",
  "name": "channel name",
  "createdByUserId": "user-id",
  "links": {
    "self": {
      "href": "/api/v1/channels/$id"
    },
    "messages": {
      "href": "/api/v1/channels/id/messages",
      "totalCount": 100,
    },
    "creator": {
      "href": "/api/v1/users/user-id"
    }
}
```

RESTfulness: OpenAPI Documentation

OpenAPI Specification

defines a **standard**, language-agnostic interface to RESTful APIs which allows both humans and computers to discover and understand the capabilities of the service.

Our API is RESTful and follows all
the best practices



But real life requirements...

*On the App landing page, we need the channels and their messages
-- An amazing product manager*

But real life requirements...

*On the App landing page, we need the channels and their messages
-- An amazing product manager*

- Channel 1
 - Message 1.1
 - Message 1.2
 - ...
- Channel 2
 - Message 2.1
 - Message 2.2
- ...

Yes.

I can do this,

Easy!

-- The naive me

Our Javascript hat on!

```
const channels = await fetchJSON('/api/v1/channels');
const msgPromises = channels.map(async function(channel) {
  // These links are so amazing!
  const messages = await fetchJSON(channel.links.messages.href);
  channel.messages = messages;
});
await Promise.all(msgPromises);
console.log(channels[0]);
```

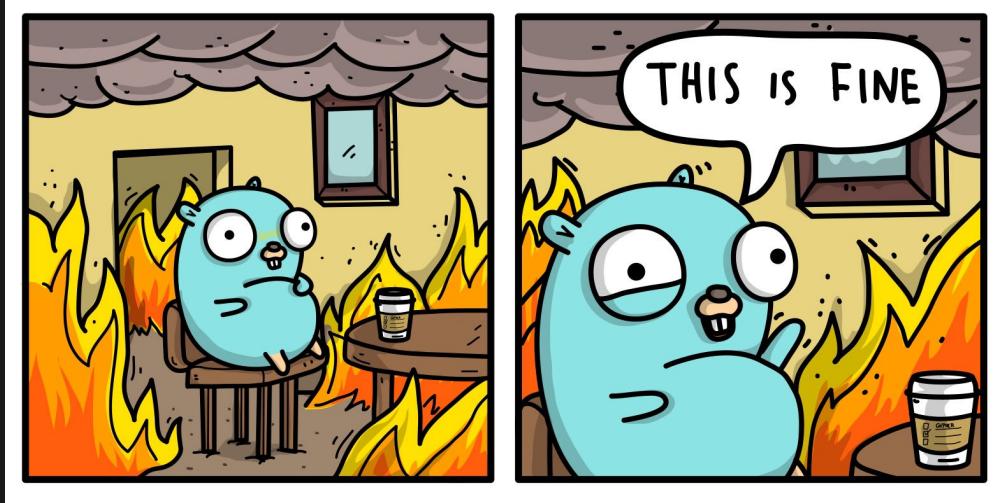
```
{
  "id": "channel-id",
  "name": "channel name",
  "messages": [ {
    "id": "message-id",
    "text": "content",
    "links": ...
  }],
  "links": ...
}
```

Our Javascript hat on!

```
const channels = await fetchJSON('/api/v1/channels');
const msgPromises = channels.map(async function(channel) {
  // These links are so amazing!
  const messages = await fetchJSON(channel.links.messages.href);
  channel.messages = messages;
});
await Promise.all(msgPromises);
console.log(channels[0]);
```

```
{
  "id": "channel-id",
  "name": "channel name",
  "messages": [ {
    "id": "message-id",
    "text": "content",
    "links": ...
  }],
  "links": ...
}
```

Great Success! 





*Our bigger customers are complaining
about performance.*

*Also we need to display the user of
each of those messages.*

-- An amazing product manager

The N+1 Problem

100 channels = 1 + 100 requests to get messages.

1. [1] GET /api/v1/channels ~100ms

2. [N] foreach Channels:

GET /api/v1/channels/:id/messages

~100ms

n=10 => 1.1s

n=100 => 10.1s

The N+1 Problem

100 channels = 1 + 100 requests to get messages.

1. [1] GET /api/v1/channels ~100ms

2. [N] foreach Channels:

GET /api/v1/channels/:id/messages

~100ms

$n=10 \Rightarrow 1.1s$

$n=100 \Rightarrow 10.1s$

Let's get the author of each messages

1 (channels) + 100 (messages of channels) + X (author of messages)

Back to the drawing board



Before putting everything in the Channel resource...
What are other people doing?

Hypertext Application Language

HAL

```
GET /api/v1/channels/:id
Accept: application/hal+json
```

```
1  {
2      "id": "id",
3      "name": "channel name",
4      "createdByUserId": "user-id",
5      "_embedded": {
6          "messages": [ {
7              "id": "msg-id",
8              "text": "some-text",
9              "_embedded": {
10                  "author": {
11                      "id": "user-id",
12                      "name": "user name",
13                      "_links": { ... }
14                  }
15              },
16          }
17      ],
18      "_links": { ... }
19  }
```

JSON API

Like HAL but with a flat structure and standardized URLs.

```
GET /api/v1/channels  
  ?include=messages  
  &include[messages]=author  
  &fields[channels]=name  
  &fields[messages]=text  
  &fields[messages][author]=name
```

JSON API - Payload

```
{  
  "data": [ {  
    "type": "channel",  
    "id": "1",  
    "attributes": {  
      "name": "channel name"  
    },  
    "relationships": {  
      "messages": {  
        "data": [ {  
          "id": "msg-id", "type": "message",  
        }, {  
          "id": "msg-id2", "type": "message",  
        }, {  
          "id": "msg-id3", "type": "message",  
        }  
      }  
    }  
  }]  
}
```

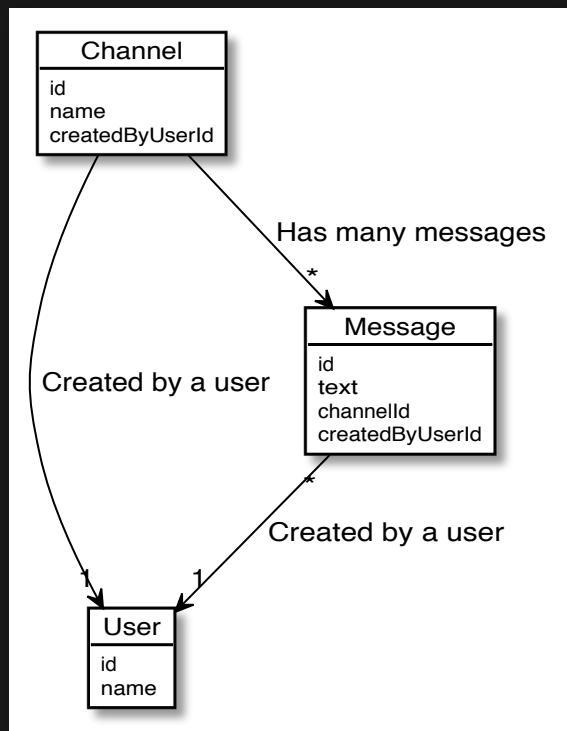
Imagine some links in there as well :)

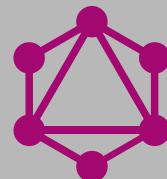
Let's Rewind



Our Example Application

A better and much faster messaging App

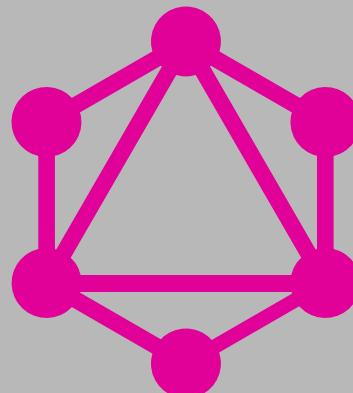




GraphQL

Describe your data

```
type Project {  
  name: String  
  tagline: String  
  contributors: [User]  
}
```



Ask for what you want

```
{  
  project(name: "GraphQL") {  
    tagline  
  }  
}
```

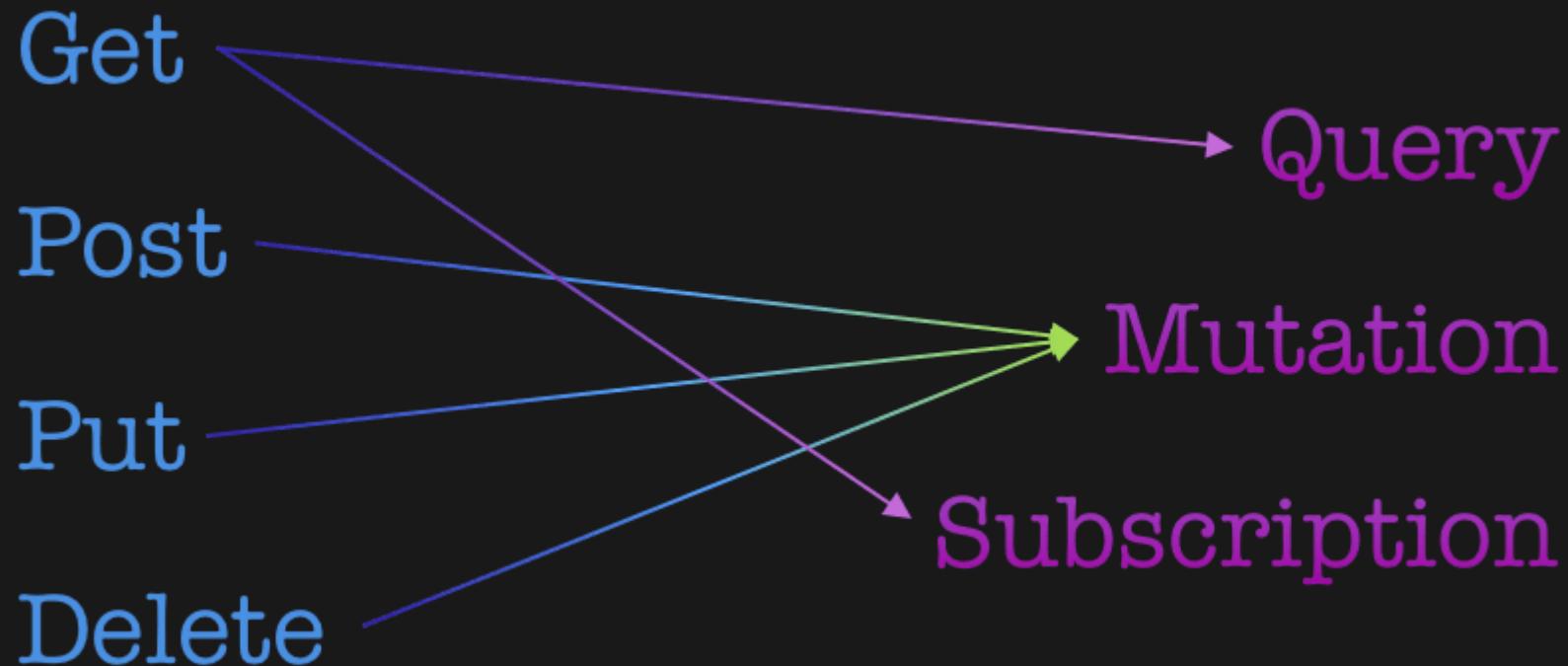
Get predictable results

```
{  
  "project": {  
    "tagline": "A query language for APIs"  
  }  
}
```

GraphQL Features

- Has a Schema
- Allows clients to only fetch what they need (fields and relationships)
- Allows fetching complex and deep relationships in one request
- Is not only a Query Language
- Generates documentation
- Allows generating "type-safe" client code

Queries • Mutations • Subscriptions

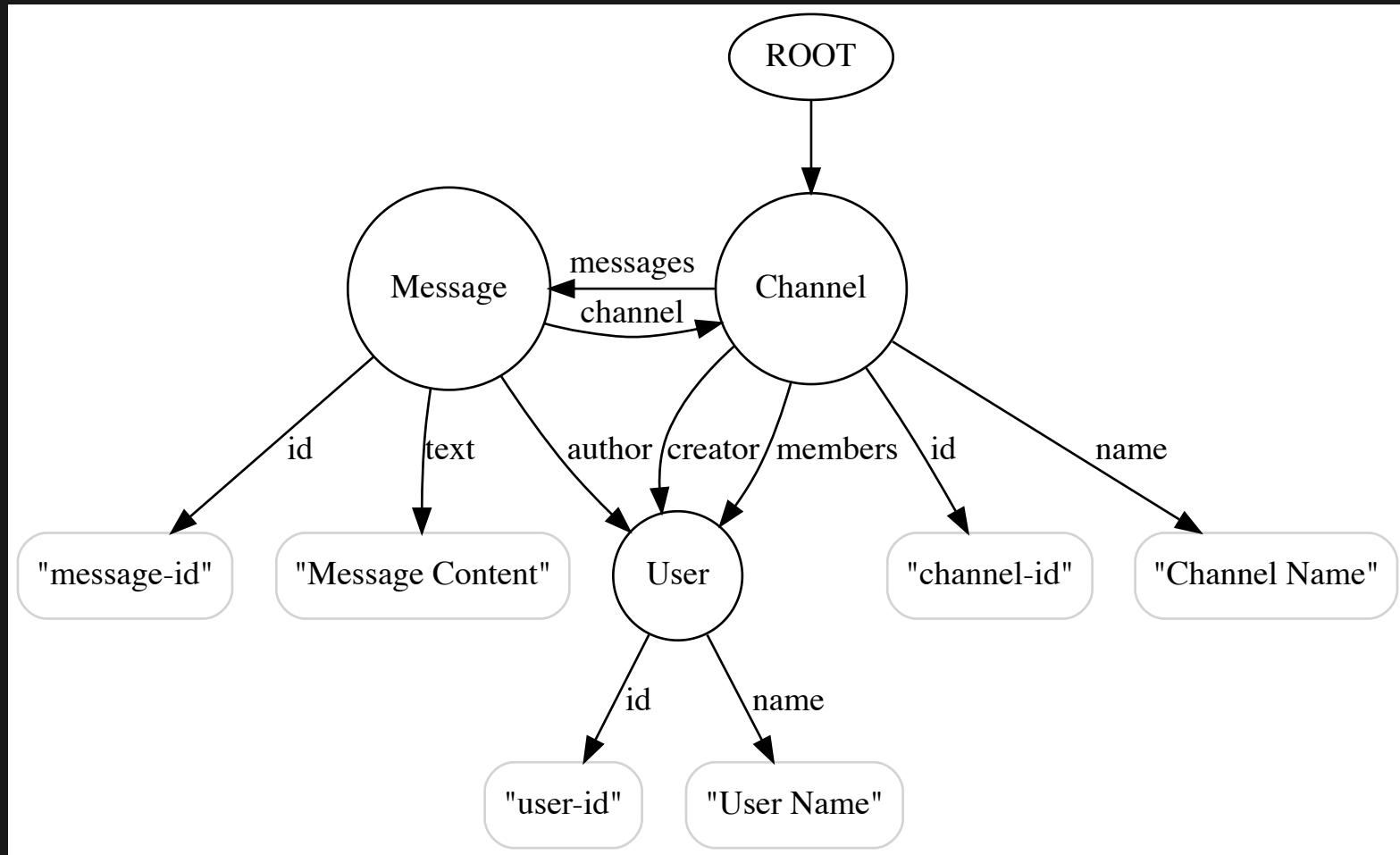


Schema

```
type User {  
    id: ID!  
    name: String!  
}  
  
type Channel {  
    id: ID!  
    name: String!  
    createdBy: User!  
    createdBy: ID!  
    messages: [Message!]!  
}  
  
type Message {  
    id: ID!  
    text: String!  
    createdBy: User!  
    createdBy: ID!  
    channel: Channel!  
    channelId: ID!  
}
```

```
type Query {  
    channels: [Channel!]!  
}  
  
input NewMessage {  
    channelId: ID!  
    text: String!  
}  
  
type Mutation {  
    createMessage(input: NewMessage!): Me
```

What is this about a Graph?



Query

```
query getChannelsForFrontpage{  
  channels {  
    name  
    createdBy {  
      name  
    }  
    messages {  
      text  
      createdBy {  
        name  
      }  
      channel {  
        name  
      }  
    }  
  }  
}
```

```
{  
  "data": {  
    "channels": [  
      {  
        "name": "Nader, Zemlak and Maye",  
        "createdBy": {  
          "name": "Ophelia Ebert"  
        },  
        "messages": [  
          {  
            "text": "You can't synthesi",  
            "createdBy": {  
              "name": "Bettye Kshlerin'"  
            },  
            "channel": {  
              "name": "Nader, Zemlak ar"  
            }  
          }  
        ]  
      }  
    ]  
  }  
}
```


Mutation

```
mutation createMsg($newMsg: NewMessage!){  
  createMessage(input: $newMsg) {  
    id  
    text  
    channel {  
      name  
      createdBy {  
        name  
      }  
    }  
  }  
}
```

```
# Variables  
{  
  "newMsg": {  
    "channelId": 3,  
    "text": "testing"  
  }  
}
```

```
{  
  "data": {  
    "createMessage": {  
      "id": "10",  
      "text": "testing",  
      "channel": {  
        "name": "Ullrich-McCullough",  
        "createdBy": {  
          "name": "Ophelia Ebert"  
        }  
      }  
    }  
  }  
}
```

Let's experiment



GraphQL Playground

Let's build our own server

Want more?

Here is what I haven't talked about

- GraphQL ❤️ Microservices
(Ask me about Manifold's Architecture)
- Subscriptions: GraphQL primitive allowing easy integration with websocket, rabbitmq, any pubsub.
- Pagination (ask me about it)
- Advanced schema (unions, interfaces, schema directives, ...)
- Advanced querying with fragments and directives
- Client code generation
- So much more

The End!