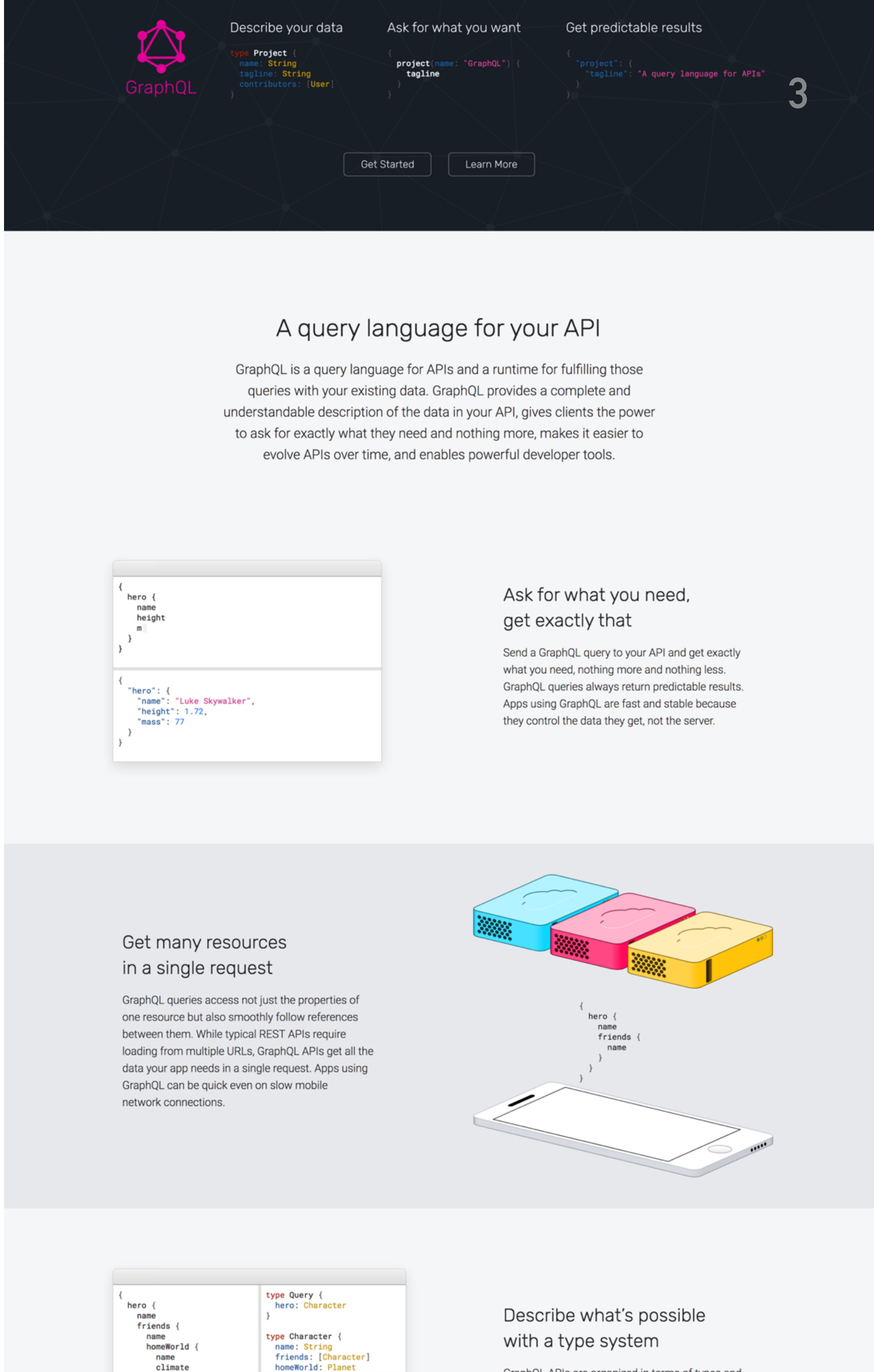
MARTIN BROCHHAUS

USING APOLLO WITH REACTJS AND GRAPHQL

WHOAMI

- Martin Brochhaus
- CTO of The Artling
- Founder of Bitlab Studio
- @mbrochh
- martin@theartling.com
- martin.brochhaus@bitlabstudio.com

- graphql.org
- ***"A query language for your API"***
- The landing page gives a very good overview over what GraphQL is
- Let's have a quick look at each section…



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"
  }
}
```

A query language for your API

GraphQL is a query language for APIs and a runtime for fulfilling those queries with your existing data. GraphQL provides a complete and understandable description of the data in your API, gives clients the power to ask for exactly what they need and nothing more, makes it easier to evolve APIs over time, and enables powerful developer tools.

```
{
  hero {
    name
    height
    m
  }
}
```

```
{
  "hero": {
    "name": "Luke Skywalker",
    "height": 1.72,
    "mass": 77
  }
}
```

Ask for what you need, get exactly that

Send a GraphQL query to your API and get exactly what you need, nothing more and nothing less. GraphQL queries always return predictable results. Apps using GraphQL are fast and stable because they control the data they get, not the server.

Get many resources in a single request

GraphQL queries access not just the properties of one resource but also smoothly follow references between them. While typical REST APIs require loading from multiple URLs, GraphQL APIs get all the data your app needs in a single request. Apps using GraphQL can be quick even on slow mobile network connections.

```
{
  hero {
    name
    friends {
      name
    }
  }
}
```

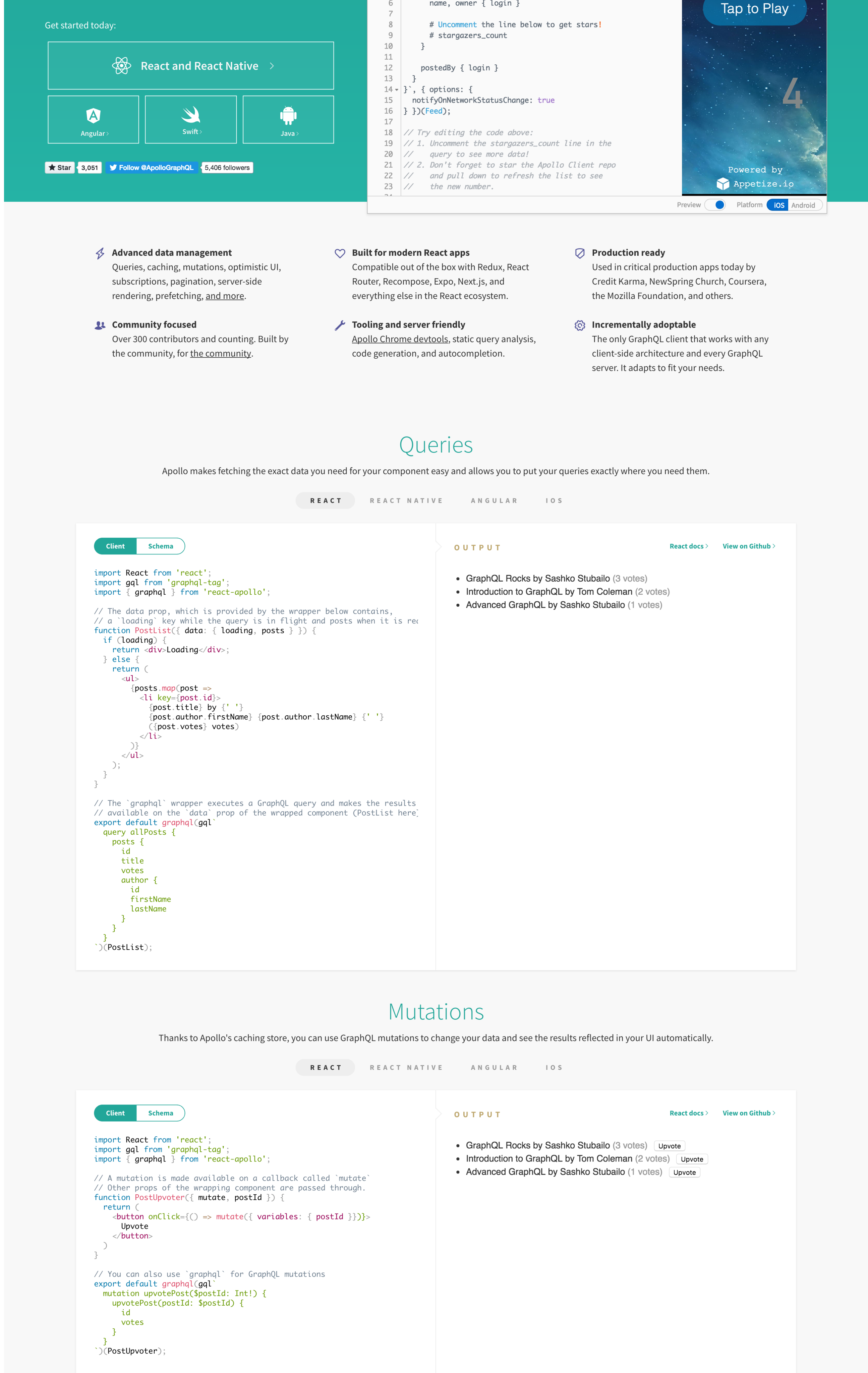
```
{
  hero {
    name
    friends {
      name
      homeWorld {
        name
        climate
```

```
type Query {
  hero: Character
}

type Character {
  name: String
  friends: [Character]
  homeWorld: Planet
```

Describe what's possible with a type system

- dev.apollodata.com
- ***"The flexible, production ready GraphQL client for React and native apps."***
- Frontend-agnostic
- Uses redux under the hood
- Can be added to your frontend without affecting anything else



```
      name, owner { login }

      # Uncomment the line below to get stars!
      # stargazers_count
    }

    postedBy { login }
  }
}`, { options: {
  notifyOnNetworkStatusChange: true
} })(Feed);

// Try editing the code above:
// 1. Uncomment the stargazers_count line in the
//    query to see more data!
// 2. Don't forget to star the Apollo Client repo
//    and pull down to refresh the list to see
//    the new number.
```

Advanced data management
Queries, caching, mutations, optimistic UI, subscriptions, pagination, server-side rendering, prefetching, and more.

Built for modern React apps
Compatible out of the box with Redux, React Router, Recompose, Expo, Next.js, and everything else in the React ecosystem.

Production ready
Used in critical production apps today by Credit Karma, NewSpring Church, Coursera, the Mozilla Foundation, and others.

Community focused
Over 300 contributors and counting. Built by the community, for the community.

Tooling and server friendly
Apollo Chrome devtools, static query analysis, code generation, and autocompletion.

Incrementally adoptable
The only GraphQL client that works with any client-side architecture and every GraphQL server. It adapts to fit your needs.

Queries

Apollo makes fetching the exact data you need for your component easy and allows you to put your queries exactly where you need them.

```
import React from 'react';
import gql from 'graphql-tag';
import { graphql } from 'react-apollo';

// The data prop, which is provided by the wrapper below contains,
// a `loading` key while the query is in flight and posts when it is rea
function PostList({ data: { loading, posts } }) {
  if (loading) {
    return <div>Loading</div>;
  } else {
    return (
      <ul>
        {posts.map(post =>
          <li key={post.id}>
            {post.title} by {' '}
            {post.author.firstName} {post.author.lastName} {' '}
            ({post.votes} votes)
          </li>
        )}
      </ul>
    );
  }
}

// The `graphql` wrapper executes a GraphQL query and makes the results
// available on the `data` prop of the wrapped component (PostList here)
export default graphql(gql`
  query allPosts {
    posts {
      id
      title
      votes
      author {
        id
        firstName
        lastName
      }
    }
  }
`)(PostList);
```

- GraphQL Rocks by Sashko Stubailo (3 votes)
- Introduction to GraphQL by Tom Coleman (2 votes)
- Advanced GraphQL by Sashko Stubailo (1 votes)

Mutations

Thanks to Apollo's caching store, you can use GraphQL mutations to change your data and see the results reflected in your UI automatically.

```
import React from 'react';
import gql from 'graphql-tag';
import { graphql } from 'react-apollo';

// A mutation is made available on a callback called `mutate`
// Other props of the wrapping component are passed through.
function PostUpvoter({ mutate, postId }) {
  return (
    <button onClick={() => mutate({ variables: { postId }})}>
      Upvote
    </button>
  )
}

// You can also use `graphql` for GraphQL mutations
export default graphql(gql`
  mutation upvotePost($postId: Int!) {
    upvotePost(postId: $postId) {
      id
      votes
    }
  }
`)(PostUpvoter);
```

- GraphQL Rocks by Sashko Stubailo (3 votes) Upvote
- Introduction to GraphQL by Tom Coleman (2 votes) Upvote
- Advanced GraphQL by Sashko Stubailo (1 votes) Upvote

First, let's get a local GraphQL backend server

I like using Django, so I prepared one that you can simply clone and run…

CLONE REPOSITORY

```
cd ~/Projects/
git clone git@github.com:mbrochh/react-apollo-graphql-demo.git
cd react—apollo—graphql-demo/backend
mkvirtualenv react-apollo-graphql-demo
pip install -r requirements.txt
./manage.py runserver 0.0.0.0:8000
```

- After these steps, you should be able to browse to “**0.0.0.0:8000/graphiql**”

Next, let's create a new project for our ReactJS frontend

You can start using Apollo today! It will not affect your existing code and you can migrate gradually, one component at a time

CREATE A NEW REACTJS PROJECT

- Install “**create-react-app**”
- Create a new ReactJS application

```
npm instal -g create-react-app
cd ~/Projects/react-apollo-graphql-demo
create-react-app frontend
```

INSTALL REACT-ROUTER AND APOLLO

```
cd ~/Projects/react-apollo-graphql-demo/frontend
yarn add react-router-dom
yarn add react-apollo
```

PREPARE APP.JS FILE: STEP 1/3

- Add necessary imports

```
import React, { Component } from 'react'
import {
  ApolloClient,
  ApolloProvider,
  createBatchingNetworkInterface,
} from 'react-apollo'
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom'
import HomeView from './views/HomeView'
import CreateView from './views/CreateView'
import DetailView from './views/DetailView'
```

PREPARE APP.JS FILE: STEP 2/3

- Setup the Apollo client

```
const networkInterface = createBatchingNetworkInterface({
  uri: ‘http://0.0.0.0:8000/gql/',
  batchInterval: 10,
  opts: {
    credentials: 'same-origin',
  },
})

const client = new ApolloClient({
  networkInterface: networkInterface,
})
```

PREPARE APP.JS FILE: STEP 3/3

- Setup your Routes

```
class App extends Component {
  render() {
    return (
      <ApolloProvider client={client}>
        <Router>
          <div>
            <Route exact path="/" component={HomeView} />
            <Switch>
              <Route exact path="/messages/create/" component={CreateView} />
              <Route exact path="/messages/:id/" component={DetailView} />
            </Switch>
          </div>
        </Router>
      </ApolloProvider>
    )
  }
}

export default App
```

CREATE THE THREE MISSING VIEWS

- Create folder “**views**” and files “**HomeView.js**”, “**CreateView.js**” and “**DetailView.js**”

```
import React from 'react'

export default class HomeView extends React.Component {
  render() {
    return <div>Home</div>
  }
}
```

- The code in all three files is the same, just change the class name and the text in the render function.

TRY IT OUT IN THE BROWSER!

- Terminal 1: Run "**./manage.py runserver 0.0.0.0:8000**"
- Terminal 2: Run "**yarn start**"
- Browse to "**/**", "**/messages/create/**" and "**/messages/1/**"

How to query data?

15 slides and still no GraphQL? Let's get going!

ADD A QUERY TO YOUR HOMEVIEW COMPONENT

- Import “**gql**” and “**graphql**” and create the query

```
# File: HomeView.js
import { gql, graphql } from 'react-apollo'

const query = gql`{
  allMessages {
    edges {
      node {
        id
        message
      }
    }
  }
}`
```

USE THE QUERY IN YOUR COMPONENT

- Wrap the component in the “**graphql**” decorator, use the data via “**this.props.data**”

```
# File: HomeView.js
class HomeView extends React.Component {
  render() {
    let { data } = this.props
    if (data.loading) { return <div>Loading...</div> }
    return (
      <div>
        {data.allMessages.edges.map((item, index) => (
          <p key={item.node.id}>
            <Link to={`/messages/${item.node.id}/`}>
              {item.node.message}
            </Link>
          </p>
        ))}
      </div>
    )
  }
}

HomeView = graphql(query)(HomeView)
export default HomeView
```

How to query data with variables?

What if I don’t want a list of all items, but one specific item?

ADD A QUERY TO YOUR DETAILVIEW COMPONENT

- Import “**gql**” and “**graphql**” and create the query

```
# File: DetailView.js

import { gql, graphql } from 'react-apollo'

const query = gql`
query DetailView($id: ID!) {
  message(id: $id) {
    id,
    message
    creationDate,
  }
}
`
```

USE THE QUERY-DATA IN YOUR COMPONENT

- Use the data via “**this.props.data**”, like we did in “**HomeView.js**”

```
# File: DetailView.js

class DetailView extends React.Component {
  render() {
    let { data } = this.props
    if (data.loading) { return <div>Loading...</div> }
    return (
      <div>
        <h1>Message: {data.message.id}</h1>
        <p>{data.message.creationDate}</p>
        <p>{data.message.message}</p>
      </div>
    )
  }
}
```

USE THE URL-PART AS A VARIABLE IN THE QUERY

- Wrap the component in the “**graphql**” decorator

```
# File: DetailView.js

const queryOptions = {
  options: props => ({
    variables: {
      id: props.match.params.id,
    },
  }),
}

DetailView = graphql(query, queryOptions)(DetailView)
export default DetailView
```

- “**props.match.params**” is available because of react-router

How to write data?

In GraphQL-land, we call that a “Mutation”.

ADD A MUTATION TO YOUR CREATEVIEW COMPONENT

- As always: Import “**gql**” and “**graphql**” and create the mutation

```
# File: CreateView.js

import { gql, graphql } from 'react-apollo'

const mutation = gql`
mutation CreateView($message: String!) {
  createMessage(message: $message) {
    formErrors,
    message {
      id,
    }
  }
}
`
```

ADD A SUBMIT HANDLER TO YOUR COMPONENT

```
# File: CreateView.js

class CreateView extends React.Component {
  handleSubmit(e) {
    e.preventDefault()
    let formData = new FormData(this.form)
    this.props
      .mutate({ variables: { message: formData.get('message') } })
      .then(res => {
        if (res.data.createMessage.formErrors === null) {
          window.location.replace(`/`)
        } else {
          console.log(res.data.createMessage.formErrors)
        }
      })
      .catch(err => {
        console.log('Network error!')
      })
  }
```

UPDATE THE MARKUP OF YOUR COMPONENT

```
# File: CreateView.js

  render() {
    return (
      <div>
        <h1>Create</h1>
        <form
          ref={ref => (this.form = ref)}
          onSubmit={e => this.handleSubmit(e)}
        >
          <textarea name="message" />
          <button type="submit">Submit</button>
        </form>
      </div>
    )
  }
```

WRAP YOUR COMPONENT IN THE GRAPHQL DECORATOR

- Wrap your component in the “**graphql**” decorator

```
# File: CreateView.js

CreateView = graphql(mutation)(CreateView)
export default CreateView
```

THANK YOU FOR LISTENING!

Martin Brochhaus
CTO of The Artling
@mbrochh