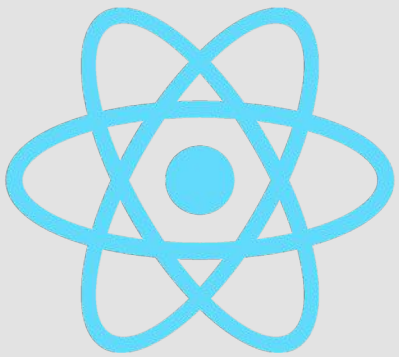
Create flexible React applications using GraphQL APIs

Maurice de Beijer - @mauricedb

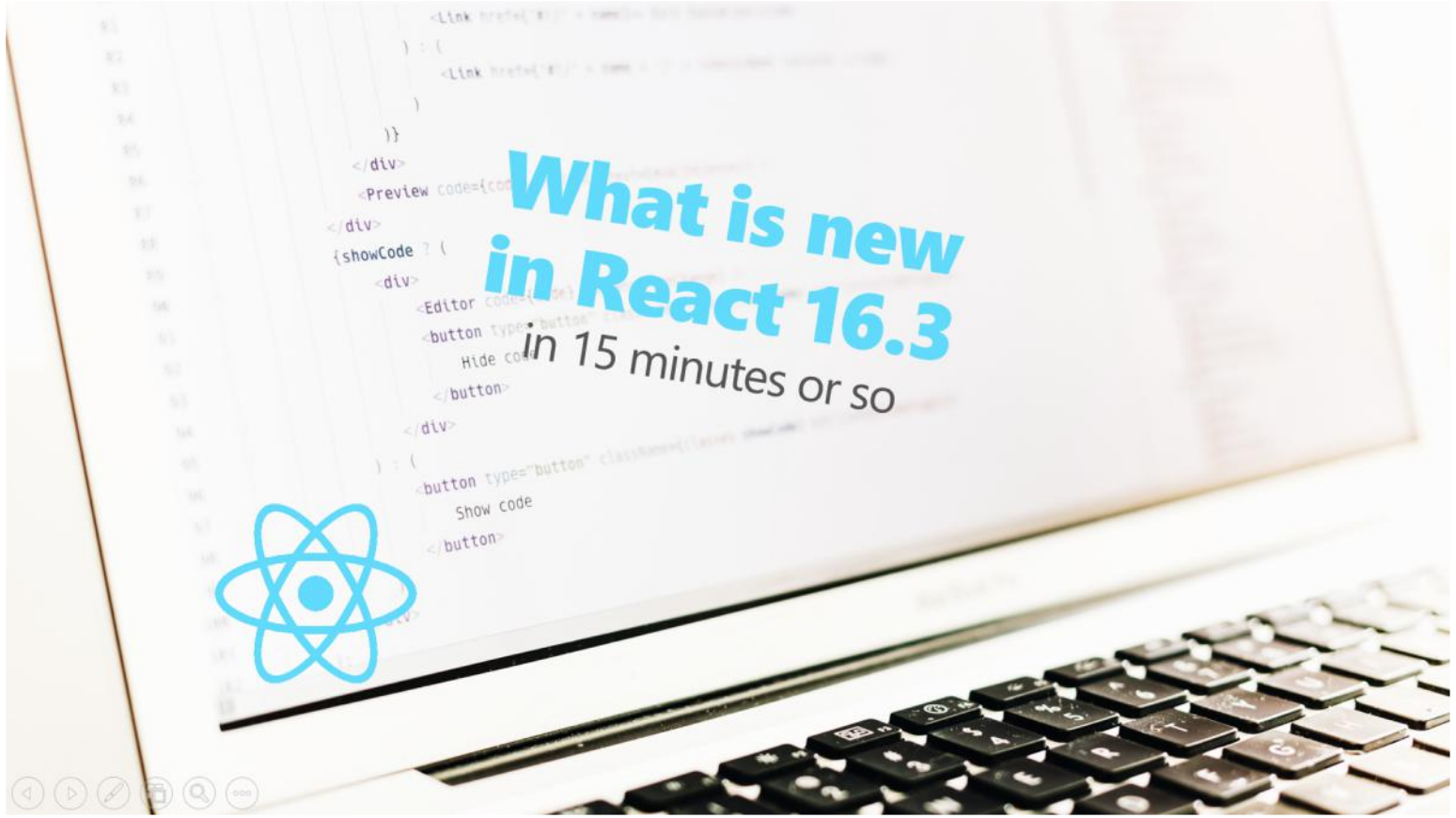


Topics

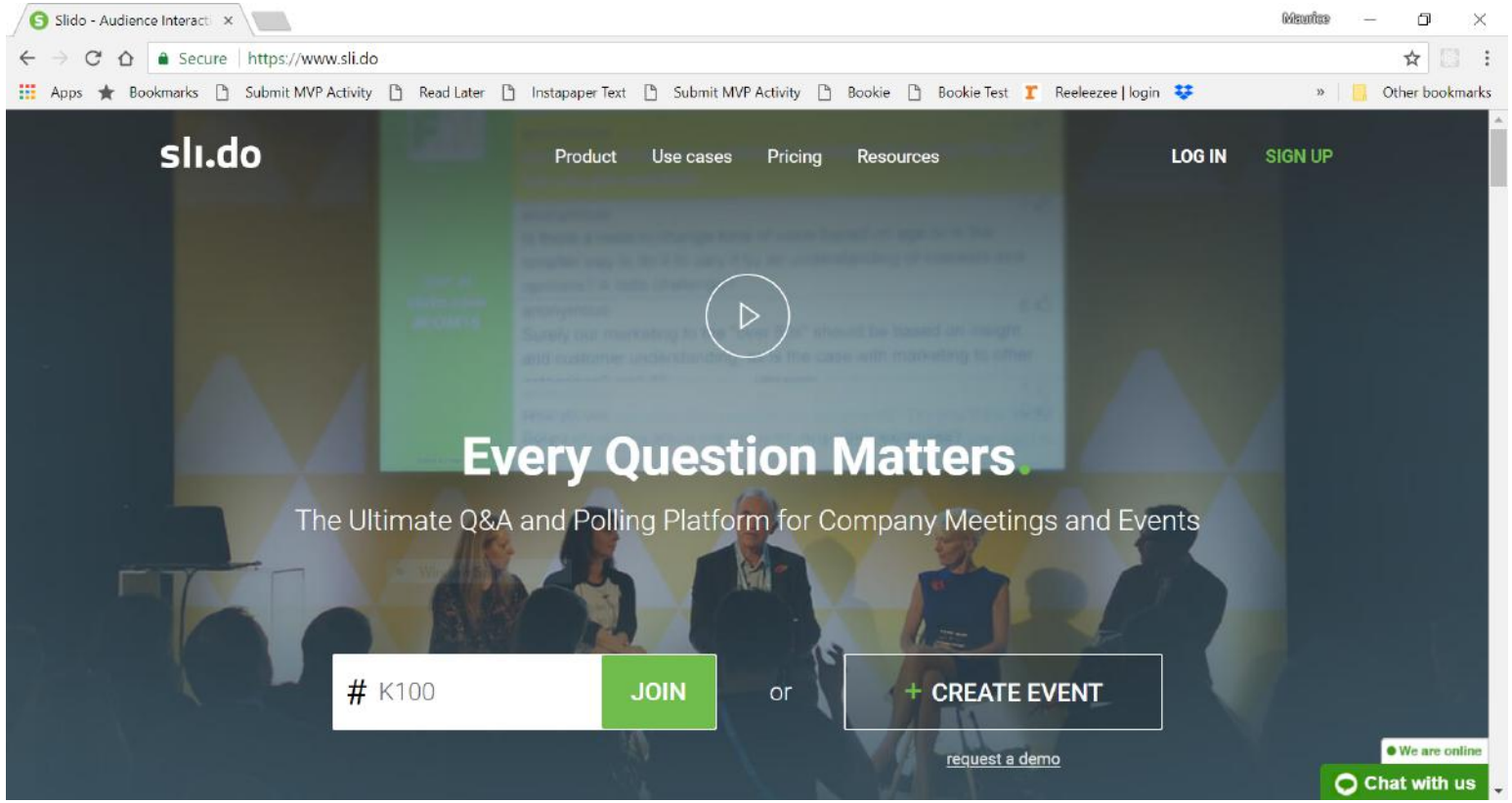
- What is **GraphQL** and why use it?
- Using GraphQL from a **React** application
- **Optimizing** GraphQL queries
- **Mutating** data

- Maurice de Beijer
- The Problem Solver
- Microsoft MVP
- Freelance developer/instructor
- Twitter: @mauricedb
- Web: http://www.TheProblemSolver.nl
- E-mail: maurice.de.beijer@gmail.com

Skillshare



Questions


Slido - Audience Interact
Secure
https://www.sli.do
Apps
Bookmarks
Submit MVP Activity
Read Later
Instapaper Text
Submit MVP Activity
Bookie
Bookie Test
Reeleezee | login
Other bookmarks
sli.do
Product
Use cases
Pricing
Resources
LOG IN
SIGN UP
Every Question Matters.
The Ultimate Q&A and Polling Platform for Company Meetings and Events
K100
JOIN
or
+ CREATE EVENT
request a demo
We are online
Chat with us

What is GraphQL?

What is GraphQL?

GraphQL is a query language for APIs

Written at Facebook

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"
  }
}
```

Get Started Learn More

A query language for your API

GraphQL is a query language for APIs and a runtime for fulfilling those

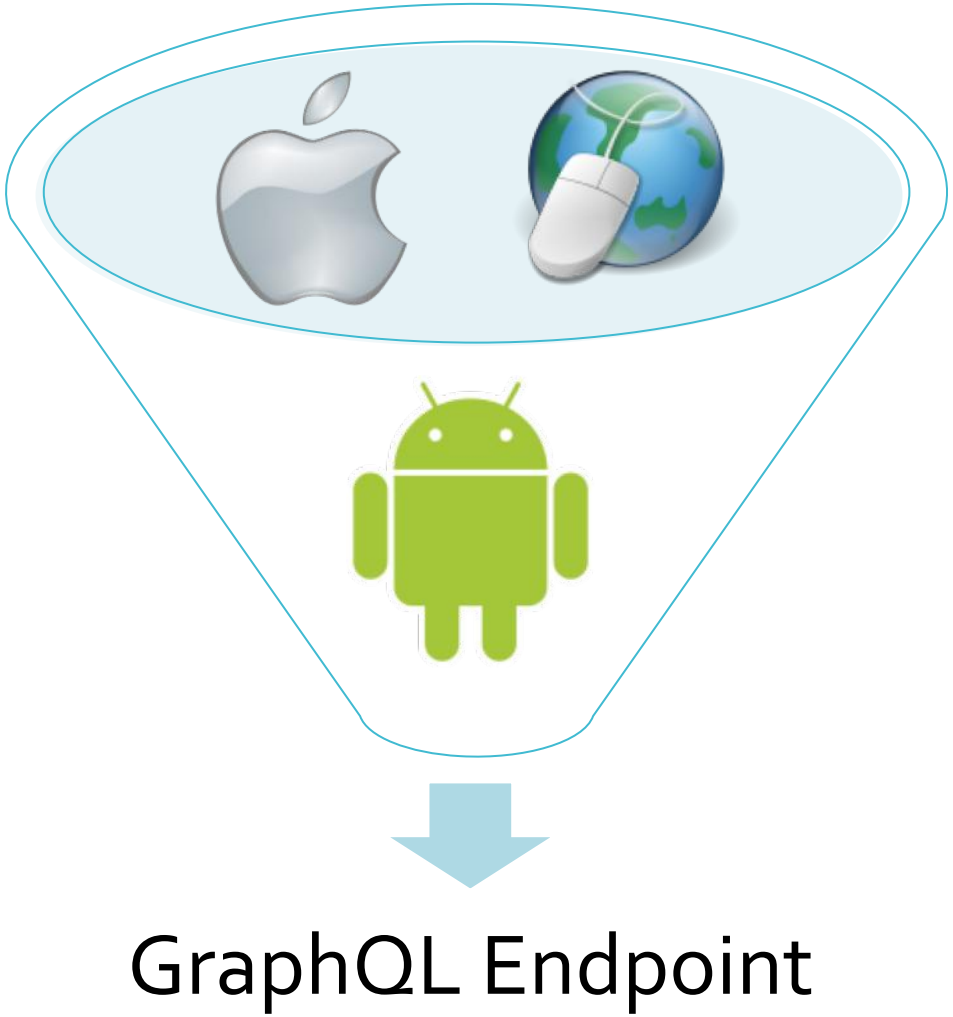
What is GraphQL?

- A way to **query hierarchical data** over HTTP
- The server provides a **data definition**
  - The client validates queries again this definition
- The **client query** defines shape of data loaded
  - Both shape and number of records
- Data sets as well as fields can be **parameterized**

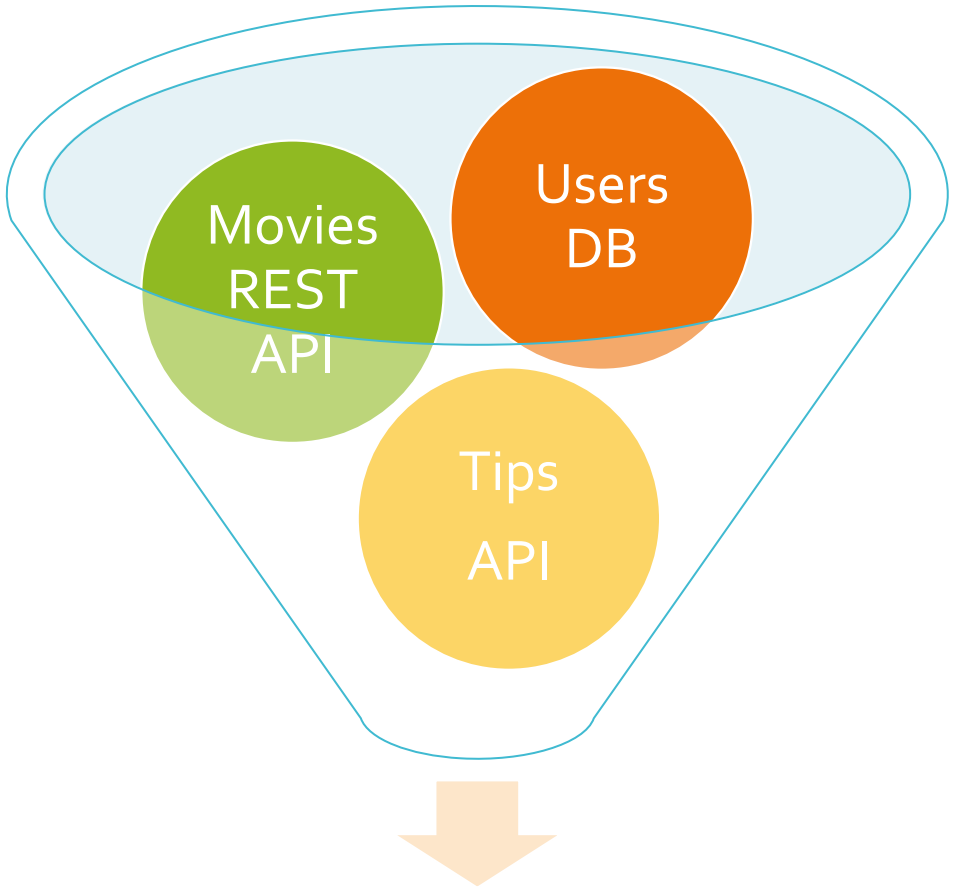
Why use GraphQL?

- Only fetch data you **really** need
  - Different clients need different result shapes
- **Consolidate** different API's
- **Playground** to test queries
  - Backed by strongly typed schema
- Clear **separation** between **queries** and **mutations**
  - CQRS for the win ☺
- And many more advantages
  - Subscriptions for data changes
  - Deprecate old fields that are not used

Clients have different requirements



Consolidate different API's



GraphQL versus Rest

Rest

- **Server decides** what you get
- **No standard** schema
  - Several flavors
- **No playground** to test
- **Caching** using HTTP

GraphQL

- Server decides what is **available**
- Client decides what to **load**
- **Strongly typed** schema
- Interactive **playground**
  - To test queries

Using GraphQL

From a React application

GraphQL Clients

Relay

A JavaScript framework for building data-driven React applications

GraphQL over REST.

Build a universal GraphQL API on top of your existing REST APIs, so you can ship new application features fast without waiting on backend changes.

TRY IT SUPPORT + TRAINING

Getting started with Apollo

- Just **three NPM packages** needed:
  - apollo-boost
  - react-apollo
  - graphql

apollo-boost

public

Readme | 7 Dependencies | 7 Dependents | 5 Versions

apollo-boost 🚀

The fastest, easiest way to get started with Apollo Client!

Apollo Boost is a zero-config way to start using Apollo Client. It includes some sensible defaults, such as our recommended `InMemoryCache` and `HttpLink`, which come configured for you with our recommended settings.

Quick start

First, install `apollo-boost`. If you don't have `graphql` & `react-apollo@beta` already in your project, please install those too.

install

```
> npm i apollo-boost
```

last 7 days

12,626

version	license
0.1.4	MIT

open issues	pull requests
269	28

The ApolloClient class

- Use the class from **apollo-boost**!
  - Comes preconfigured with sensible defaults

```
import ApolloClient from 'apollo-boost';

const client = new ApolloClient({
  uri: 'http://localhost:4000/graphql'
});
```

react-apollo

public

Readme | 5 Dependencies | 265 Dependents | 135 Versions

React Apollo allows you to fetch data from your GraphQL server and use it in building complex and reactive UIs using the React framework. React Apollo may be used in any context that React may be used. In the browser, in React Native, or in Node.js when you want to do server-side rendering.

React Apollo unlike many other tools in the React ecosystem requires *no* complex build setup to get up and running. As long as you have a GraphQL server you can get started building out your application with React immediately. React Apollo works out of the box with both `create-react-app` and React Native with a single install and with no extra hassle configuring Babel or other JavaScript tools.

React Apollo is:

1. **Incrementally adoptable**, so that you can drop it into an existing JavaScript app and start using GraphQL for just part of your UI.
2. **Universally compatible**, so that Apollo works with any build setup, any GraphQL server, and any GraphQL schema.
3. **Simple to get started with**, you can start loading data right away and learn about advanced features later.

install

```
> npm i react-apollo
```

last 7 days

128,431

version	license
2.1.3	MIT

open issues	pull requests
176	9

The ApolloProvider component

- **Provides** the rest of the application access to the ApolloClient

```
import { ApolloProvider } from 'react-apollo';

ReactDOM.render(
  <ApolloProvider client={client}>
    <App />
  </ApolloProvider>,
  document.getElementById('root')
);
```

graphql

public

Readme | 1 Dependencies | 1,518 Dependents | 83 Versions

GraphQL.js

The JavaScript reference implementation for GraphQL, a query language for APIs created by Facebook.

npm package 0.13.2 | build passing | coverage 99%

See more complete documentation at http://graphql.org/ and http://graphql.org/graphql-js/.

Looking for help? Find resources from the community.

Getting Started

An overview of GraphQL in general is available in the README for the Specification for GraphQL. That overview describes a simple set of GraphQL examples that exist as tests in this repository. A good way

install

```
> npm i graphql
```

last 7 days

514,860

version	license
0.13.2	MIT

open issues	pull requests
67	21

The GQL query

- Define the **data requirements**
- Add **parameters** as needed

```
const query = gql`
  query movies {
    topRatedMovies {
      id
      title
      overview
      poster_path(size: w154)
      vote_average
      vote_count
    }
  }
`;
```

The Query component

- **Executes the query** using the provided ApolloClient
- Uses the **render props** syntax with the QueryResult
  - There is also a **graphql HOC** available

```
const MoviesContainer = () => (
  <div>
    <h2>Top rated movies</h2>
    <Query query={query}>
      {result => <MoviesPresentation {...result} />}
    </Query>
  </div>
);
```

The QueryResult

```
const MoviesPresentation = ({ loading, error, data }) => {
  if (loading) return <div>Loading...</div>;
  if (error) return <div>{error.message}</div>;

  return (
    <div>
      {data.topRatedMovies.map(movie => (
        <MovieListItem key={movie.id} movie={movie} />
      ))}
    </div>
  );
};
```

Defining query fragments

- Each component defines **its own data requirements**

```
export const movieFragment = gql`
  fragment movieFragment on Movie {
    id
    ...movieListItemFragment
  }
  ${movieListItemFragment}
`;
```

Using query fragments

- These are combined into one **single query**

```
const query = gql`
  query {
    topRatedMovies {
      ...movieFragment
    }
  }
  ${movieFragment}
`;
```

Batching HTTP Requests

- **Multiple queries** result in multiple HTTP request
- Batch these into a single request using the **BatchHttpLink**
- Use the **ApolloClient** from **apollo-client**
  - The ApolloClient from apollo-boost doesn't support this (yet)

```
import { ApolloClient } from 'apollo-client';
import { BatchHttpLink } from 'apollo-link-batch-http';
import { InMemoryCache } from 'apollo-cache-inmemory';

const client = new ApolloClient({
  link: new BatchHttpLink({
    uri: 'http://localhost:4000/graphql'
  }),
  cache: new InMemoryCache()
});
```

Mutating data

- Any **updates** can be send to the server using **mutations**
  - Calls a function on the server with the specified data

The GQL mutation

- Defines the **data to send** to the server
  - And the data we want to receive on success

```
const rateMovie = gql`
  mutation rateMovie($id: Int!, $vote: Int!) {
    rateMovie(id: $id, vote: $vote) {
      id
      vote_average
      vote_count
    }
  }
`;
```

The Mutation component

- **Executes the mutation** using the provided ApolloClient
- Uses the **render props** syntax with the function to mutate
  - There is also an **graphql HOC** available
- Use the **update function** to update caches

```
export default ({ movie }) => (
  <div>
    <h3>{movie.title}</h3>
    <Mutation mutation={rateMovie} update={updateAfterMutate}>
      {rateMovie => <RateUp movie={movie} rateMovie={rateMovie} />}
    </Mutation>
  </div>
);
```

Executing the mutation

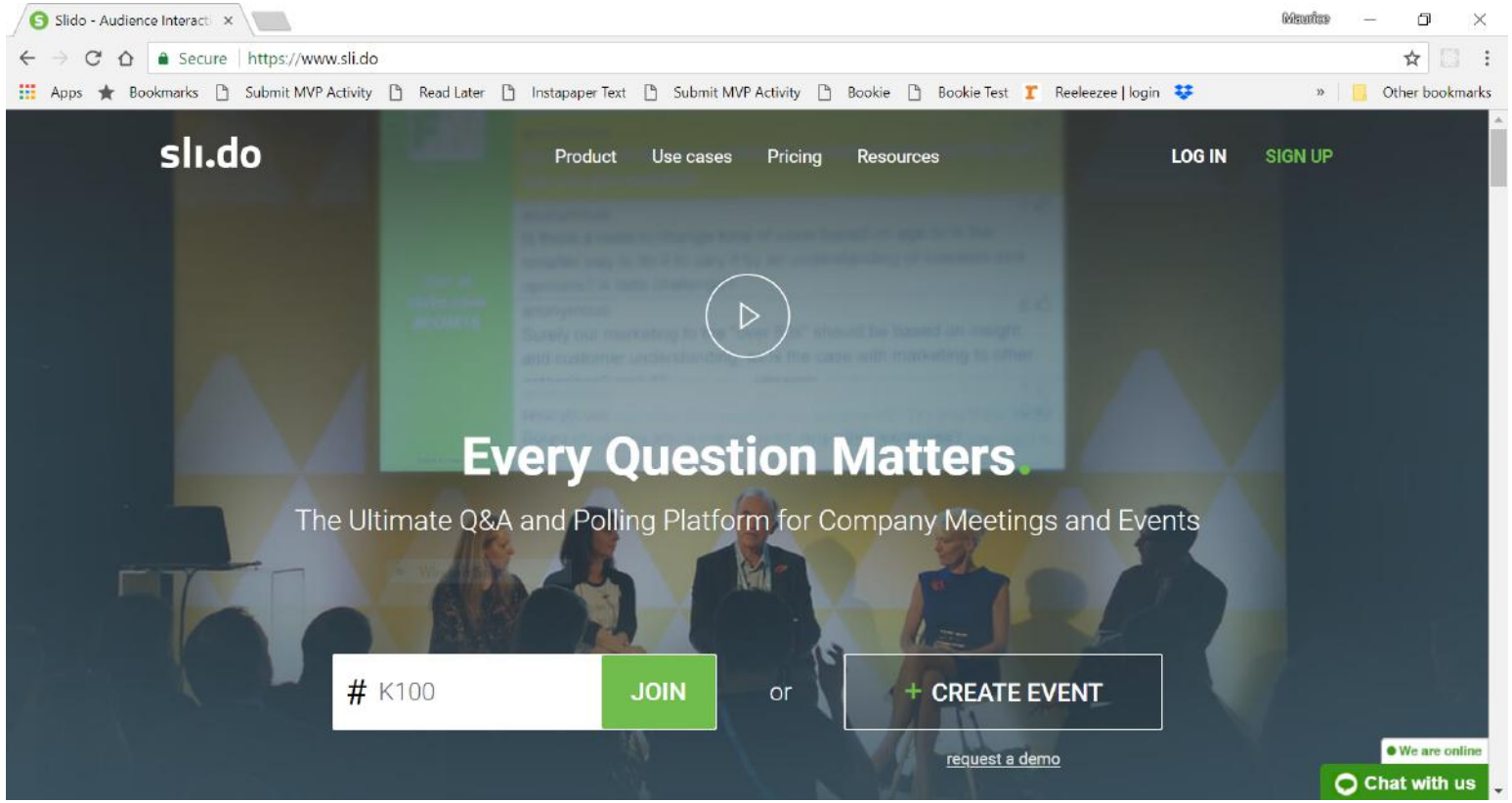
- Provide the **mutation parameters** using the **variables object**

```
const RateUp = ({ movie, rateMovie }) => (
  <button
    onClick={() =>
      rateMovie({
        variables: { id: movie.id, vote: 10 }
      })
    }
  >👍</button>
);
```

Conclusion

- GraphQL is a **great way** to query API's over HTTP
  - The server determines what is possible
  - The client controls what is loaded
- Queries are validated against a **type system**
- Updates can be done using **mutations**

Questions


Slido - Audience Interact
Secure
https://www.sli.do
Apps
Bookmarks
Submit MVP Activity
Read Later
Instapaper Text
Submit MVP Activity
Bookie
Bookie Test
Reeleezee | login
Other bookmarks
sli.do
Product
Use cases
Pricing
Resources
LOG IN
SIGN UP
Every Question Matters.
The Ultimate Q&A and Polling Platform for Company Meetings and Events
K100
JOIN
or
+ CREATE EVENT
request a demo
We are online
Chat with us

Maurice de Beijer

@mauricedb