GraphQL in 2020

< geekle >

Photo by Nicolas Gras

Who am I

Charly POLY

Who am I

Charly POLY

Lead Sr. Software Engineer at Double

Who am I

Charly POLY

Lead Sr. Software Engineer at Double

Big fan and user of GraphQL since 4 years

GraphQL in 1’

1. Schema

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

type Query {
  project(name: String!): Project
}
```

GraphQL in 1’

1. Schema

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

type Query {
  project(name: String!): Project
}
```

```
const resolvers = {
  Query: {
    project: (parent, args, context, info) => (
       projects.find(p => p.name === args.name)
    ),
  },
};
```

2. Resolvers

GraphQL in 1’

1. Schema

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

type Query {
  project(name: String!): Project
}
```

```
const resolvers = {
  Query: {
    project: (parent, args, context, info) => (
       projects.find(p => p.name === args.name)
    ),
  },
};
```

2. Resolvers

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

3. Query

GraphQL in 1’

1. Schema

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

type Query {
  project(name: String!): Project
}
```

```
const resolvers = {
  Query: {
    project: (parent, args, context, info) => (
       projects.find(p => p.name === args.name)
    ),
  },
};
```

2. Resolvers

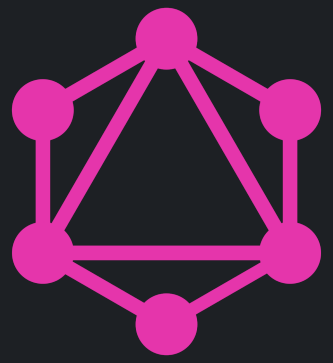
```
{
  project(name: "GraphQL") {
    tagline
  }
}
```

3. Query

4. JSON data

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

GraphQL in 2015

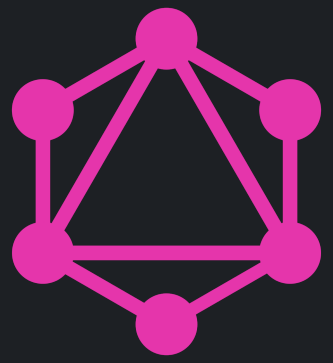


initial public release

server: graphql@0.0.2 (+ express)

client: relay@0.1.0

GraphQL in 2015

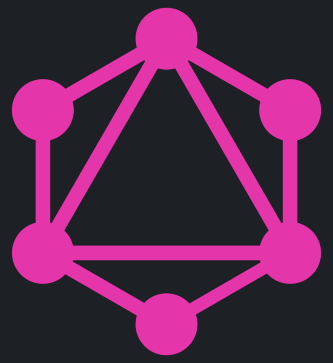


initial public release

server: graphql@0.0.2 (+ express)

```
import {
  graphql,
  GraphQLSchema,
  GraphQLObjectType,
  GraphQLString,
} from 'graphql';

var schema = new GraphQLSchema({
  query: new GraphQLObjectType({
    name: 'RootQueryType',
    fields: {
      hello: {
        type: GraphQLString,
        resolve() {
          return 'world';
        },
      },
    },
  }),
});
```

GraphQL in 2016



Ecosystem awakening

graphql-tools

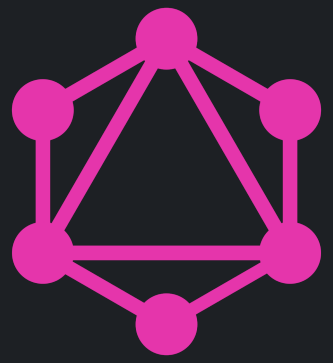
apollo-server

apollo-client (+ react-apollo)

graphiql

Libraries to boost development

GraphQL in 2016



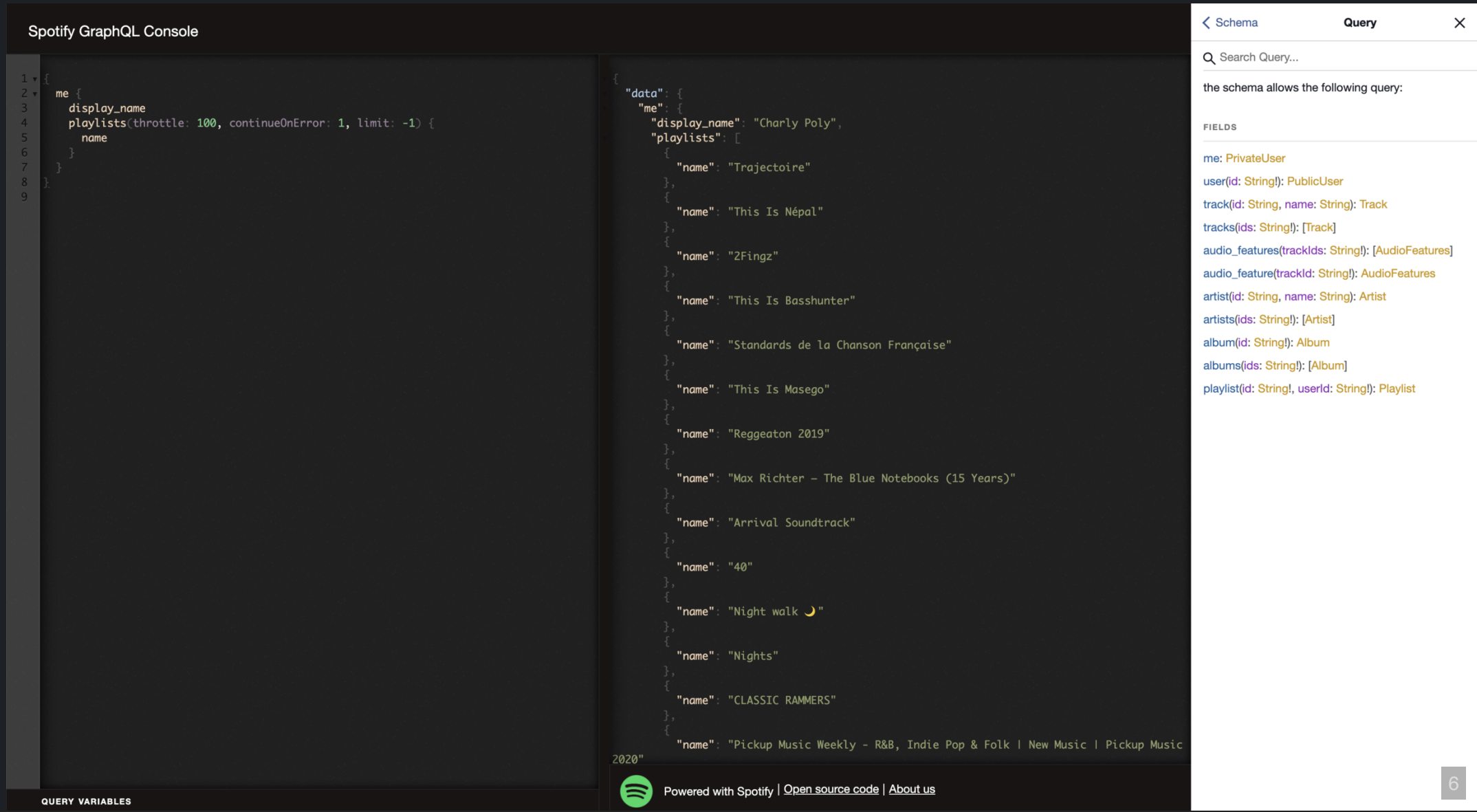
Ecosystem awakening

graphql-tools

```
1  import { makeExecutableSchema } from '@graphql-tools
2
3
4  const resolvers = /*...*/
5
6  const typeDefs = `
7  # ...
8
9  schema {
10   query: Query
11   mutation: Mutation
12 }
13 `;
14
15 const executableSchema = makeExecutableSchema({
16   typeDefs,
17   resolvers,
18 });
```

GraphQL in 2016 - intermission

GraphQL in 2020



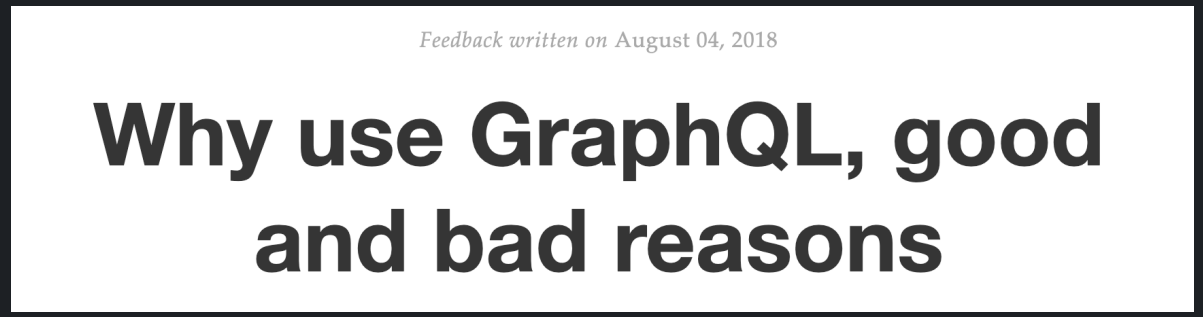
GraphQL in 2018 - intermission

GraphQL in 2018 - intermission

Feedback written on August 04, 2018

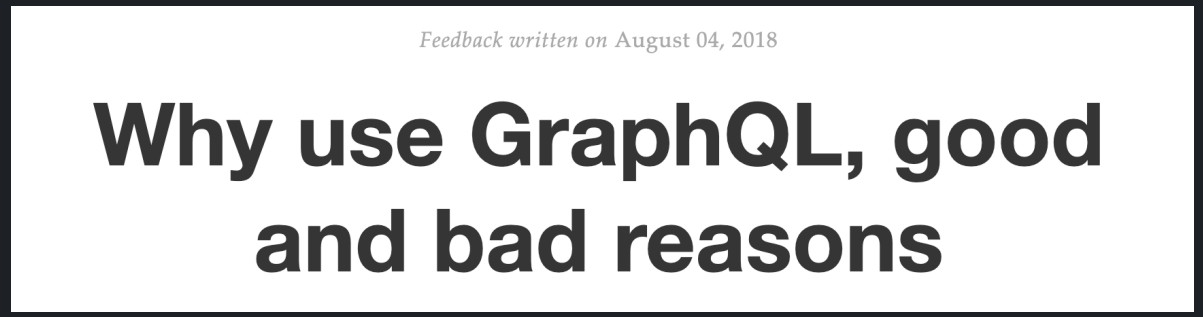
Why use GraphQL, good and bad reasons

GraphQL in 2018 - intermission



- **Build smooth user-experiences**
  "Ask for what you want", optimistic UIs

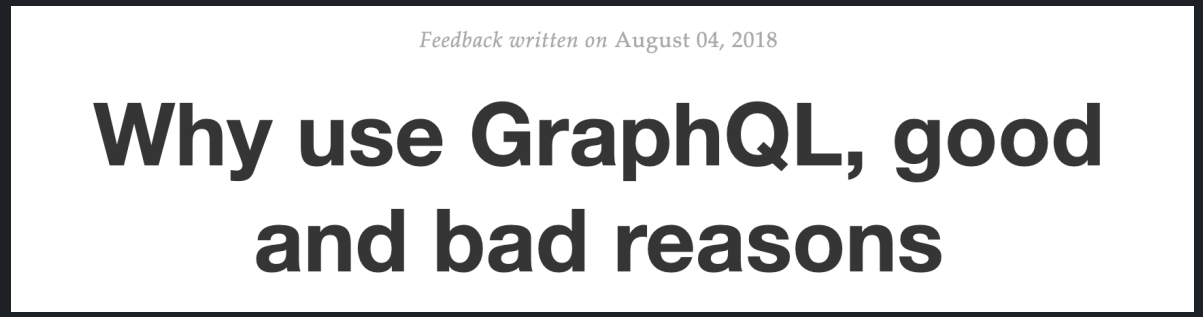
GraphQL in 2018 - intermission



- **Build smooth user-experiences**
  "Ask for what you want", optimistic UIs

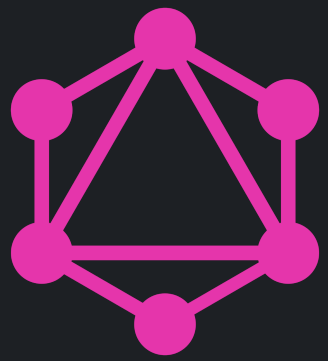
- **Solve data-complexity issue on front-end side**
  Apollo cache, typed mutations, DDD APIs

GraphQL in 2018 - intermission



- **Build smooth user-experiences**
  "Ask for what you want", optimistic UIs
- **Solve data-complexity issue on front-end side**
  Apollo cache, typed mutations, DDD APIs
- **Microservices orchestration**
  Apollo schema stitching ➡️ Apollo Federation

GraphQL in 2018



Ecosystem on the rise

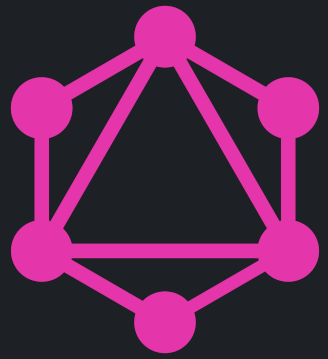
Libraries: Apollo, Relay

Concepts: Schema stitching, client cache, optimistic UI patterns

Services: AWS App Sync, Apollo Optics, Prisma

Faster and more robust development

GraphQL in 2018



First public GraphQL APIs

GraphQL in 2019

State of JS 2019

GraphQL in 2020?

GraphQL in 2020?

GraphQL modules

GraphQL Code Generator

Apollo Platform

Relay

DGraph

Apollo Federation

graphqless

Hasura

Hasura

GraphQL Engine

GraphQL Mesh

Apollo State management

GraphCMS

urql

AWS AppSync

Prisma

TypeGraphQL

Gatsby

GraphQL in 2020?

GraphQL modules

GraphQL Code Generator

Apollo Platform

Relay

DGraph

Apollo Federation

graphqless

Hasura

Hasura

GraphQL Engine

GraphQL Mesh

Apollo State management

GraphCMS

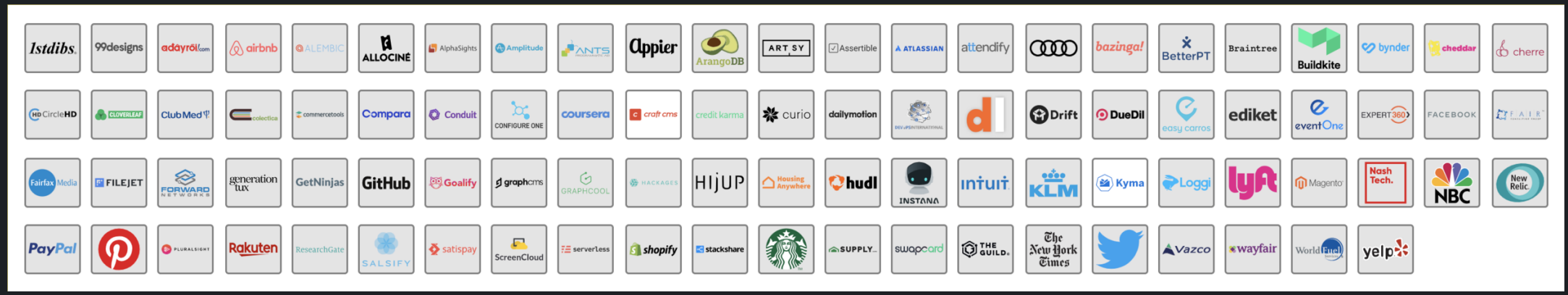
urql

AWS AppSync

Prisma

TypeGraphQL

Gatsby



GraphQL in 2020

> *GraphQL is much more than an efficient way of fetching data from the client side*

GraphQL in 2020

> *GraphQL is much more than an efficient way of fetching data from the client side*

GraphQL in 2020

> *GraphQL is much more than an efficient way of fetching data from the client side*

1. The GraphQL innovations on front-end side

GraphQL in 2020

> *GraphQL is much more than an efficient way of fetching data from the client side*

1. The GraphQL innovations on front-end side

2. GraphQL bridging the gaps on server-side

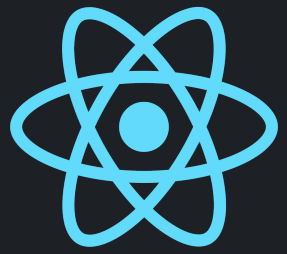
The GraphQL innovations on front-end side

GraphQL as state management

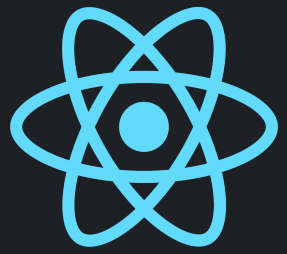
Photo by Taelynn Christopher

GraphQL as state management

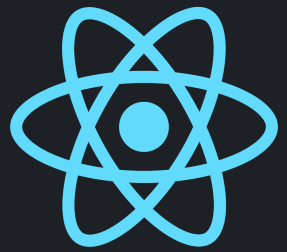
GraphQL as state management



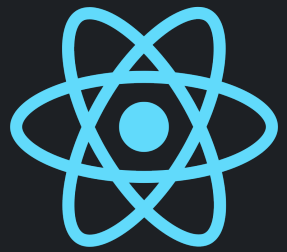
GraphQL as state management



GraphQL as state management



GraphQL as state management



Recoil

GraphQL as state management

> *GraphQL is a "data query and manipulation language for APIs"*

GraphQL as state management

> *GraphQL is a "data query and manipulation language ~~for APIs~~"*

APOLLO

GraphQL as state management

Apollo Client mission

GraphQL as state management

Apollo Client mission

- **Apollo Client 1 & 2**: build the best GraphQL Client

GraphQL as state management

Apollo Client mission

- **Apollo Client 1 & 2**: build the best GraphQL Client
- **Apollo Client 3**: a library for interacting with a client-side data graph

GraphQL as state management

Apollo Client mission

- **Apollo Client 1 & 2**: build the best GraphQL Client
- **Apollo Client 3**: a library for interacting with a client-side data graph

Local state management with GraphQL

GraphQL as state management

Simple local state example

GraphQL as state management

Example: darkMode settings

GraphQL as state management

Example: darkMode settings

```
const { data } = useQuery(`
  query isDarkMode {
    darkMode @client
  }
`)
```

GraphQL as state management

Example: darkMode settings

```
const { data } = useQuery(`
  query isDarkMode {
    darkMode @client
  }
`)
```

```
import { TypePolicies, makeVar } from '@apollo/client';


export const DARK_MODE_KEY = 'dark-mode'

export const darkMode = makeVar<boolean>(localStorage.getItem(DARK_MODE_KEY) || false)


export const resolvers: TypePolicies = {
  Query: {
    fields: {
      darkMode: {
        read: () => darkMode(),
      },
    },
  },
};


export default resolvers
```

GraphQL as state management

Example: darkMode settings

GraphQL as state management

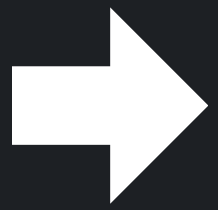
Example: darkMode settings

```
1  darkMode(true)
```

GraphQL as state management

Example: darkMode settings

```
darkMode(true)
```

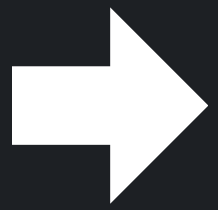


```
import { TypePolicies, makeVar } from '@apollo/client';


export const DARK_MODE_KEY = 'dark-mode'

export const darkMode = makeVar<boolean>(localStorage.getI


export const resolvers: TypePolicies = {
  Query: {
    fields: {
      darkMode: {
        read: () => darkMode(),
      },
    },
  },
};


export default resolvers
```

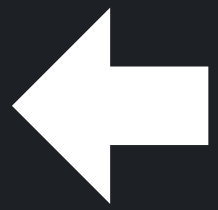
GraphQL as state management

Example: darkMode settings

```
1  darkMode(true)
```



```
1  import { TypePolicies, makeVar } from '@apollo/client';
2
3
4  export const DARK_MODE_KEY = 'dark-mode'
5
6  export const darkMode = makeVar<boolean>(localStorage.getI
7
8
9  export const resolvers: TypePolicies = {
10   Query: {
11     fields: {
12       darkMode: {
13         read: () => darkMode(),
14       },
15     },
16   },
17 };
18
19
20 export default resolvers
```



updates
useQuery() hook

GraphQL as state management

Example: darkMode settings

GraphQL as state management

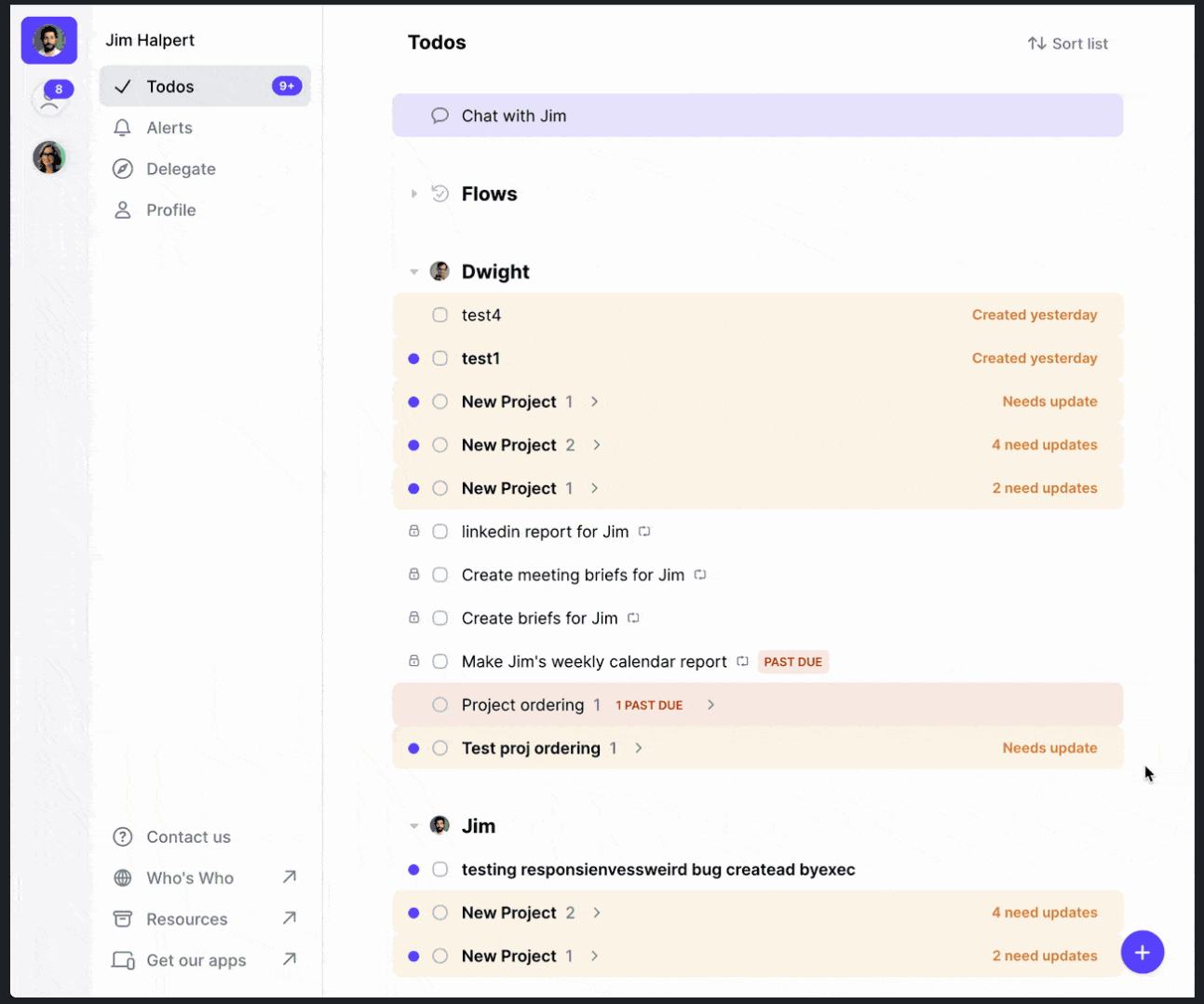
Example: darkMode settings

```
1 import { darkMode, DARK_MODE_KEY } from './resolvers'
2
3 const useUpdateDarkMode = () => useCallback(
4   (enabled: boolean) => {
5     localStorage.setItem(DARK_MODE_KEY, enabled)
6     darkMode(enabled)
7   },
8   [],
9 )
```

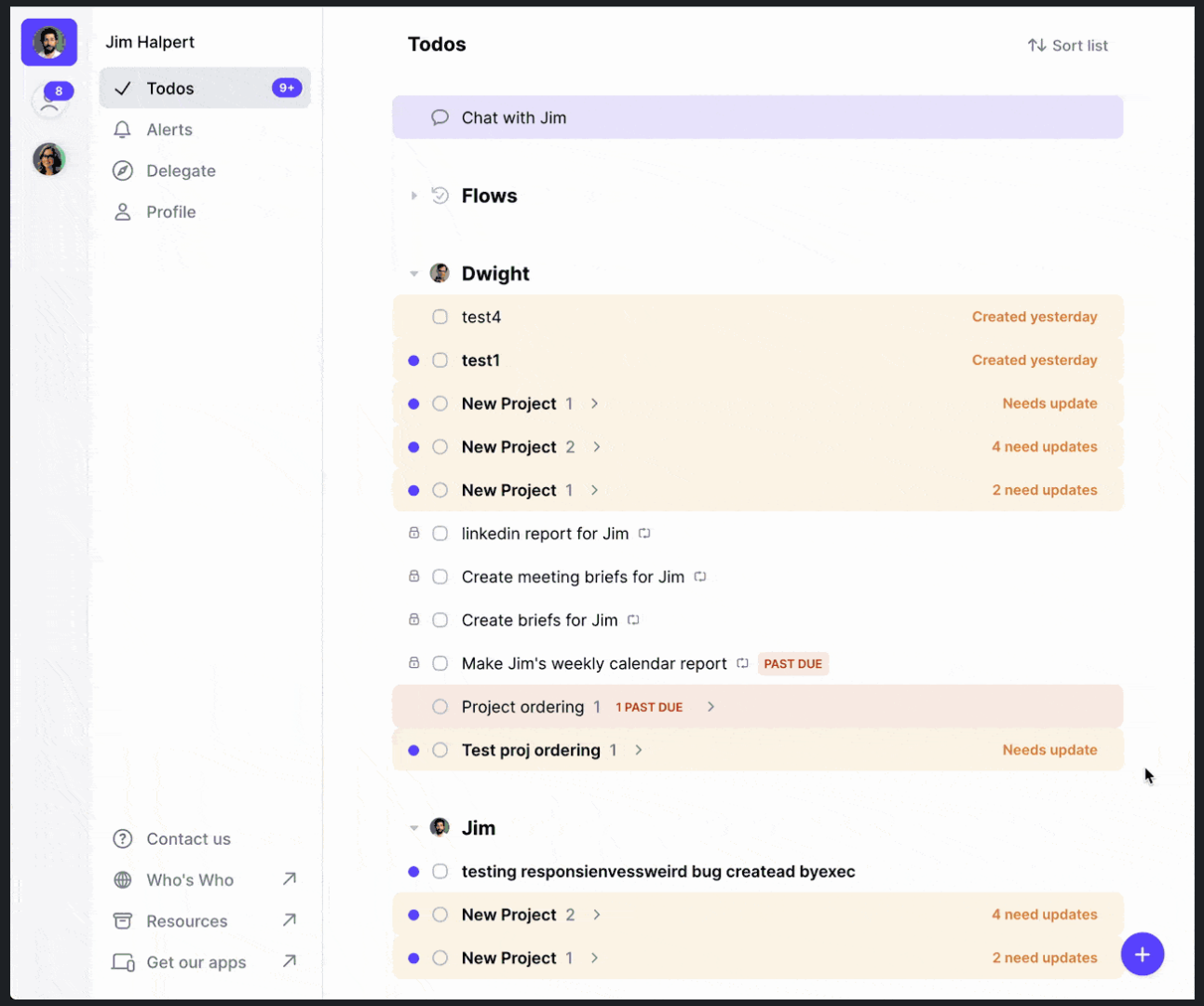
GraphQL as state management

More advanced local state example

GraphQL as state management

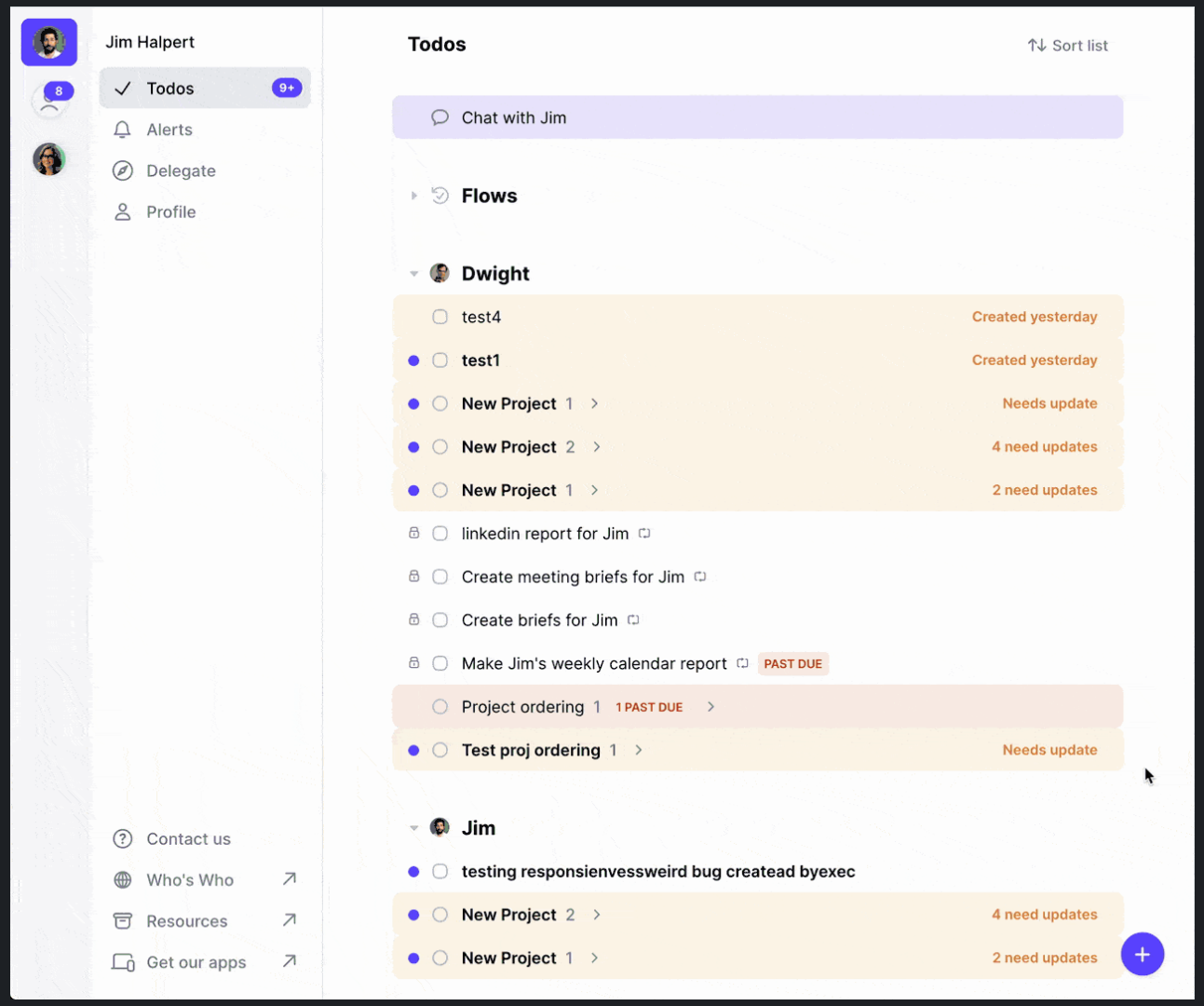


GraphQL as state management



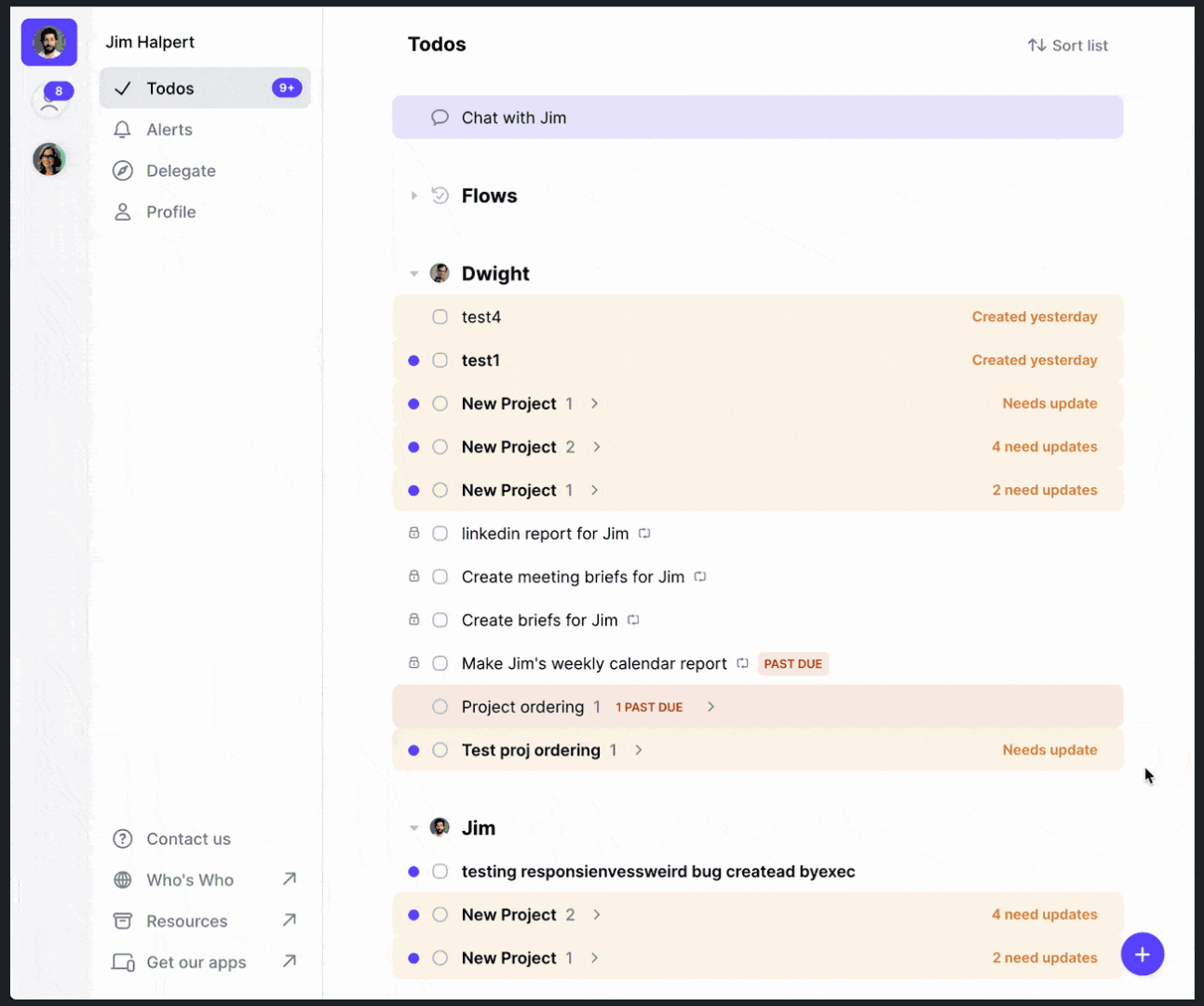
- TodoList items are remote (GQL API)

GraphQL as state management



- TodoList items are remote (GQL API)
- The newly created Todo is local

GraphQL as state management



- TodoList items are remote (GQL API)
- The newly created Todo is local
- Totally transparent at component- level

```
1 const { data, error } = useMainTopicListQuery()
2
3 // data contains remote + local todos
```

GraphQL as state management

Adding the TodoDraft to the list Query

GraphQL as state management

1. Adding the TodoDraft to the list Query

```
const { data } = useQuery(`
  query getMainTopicList($completedAfter: DateTime!) {
    activeTopics: topicsList(first: 200, view: ACTIVE) {
      hasAfter
      items {
        ...TopicListItem
      }
    }
  }
`)
```

GraphQL as state management

1. Adding the TodoDraft to the list Query

```
const { data } = useQuery(`
  query getMainTopicList($completedAfter: DateTime!) {
    activeTopics: topicsList(first: 200, view: ACTIVE) {
      hasAfter
      items {
        ...TopicListItem
      }
    }
  }
`)
```

```
const typePolicies: TypePolicies = {
  Query: {
    fields: {
      topicsList: {
        keyArgs: ['view', 'first'],
        // ...
        read: (existing, { args }) => {
          const draft = todoDrafts()
          return draft ?
            {
              ...existing,
              items: [
                draft,
                ...(existing ? existing.items : []),
              ],
            }:
            existing
        },
      },
    },
  },
}
```

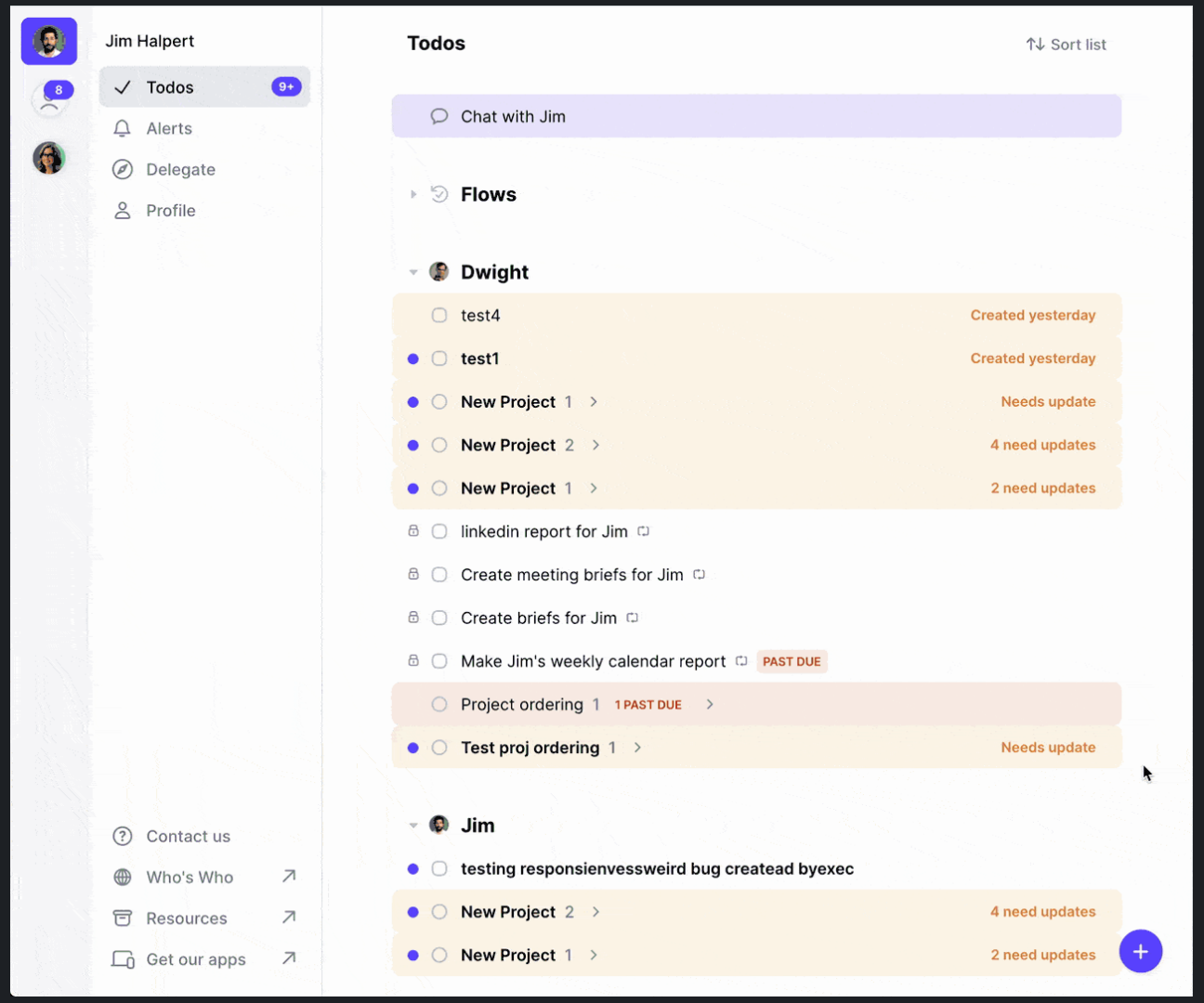
GraphQL as state management

2. Todo `isDraft` local property

```
fragment TodoListItem on Todo {
  id
  reference
  title
  # ...
  isDraft @client
  # ...
}
```

```
export const resolvers: TypePolicies = {
  // we augment the `Todo` type
  Todo: {
    fields: {
      isDraft: {
        read: (existing = false, { readField }) => isDraft(readField('id')),
      },
    },
  },
};
```

GraphQL as state management



GraphQL as state management

Full local state management capabilities

GraphQL as state management

Full local state management capabilities

- **State**: managed by ApolloCache, along side with APIs data

GraphQL as state management

Full local state management capabilities

- **State**: managed by ApolloCache, along side with APIs data
- **Computed values**: TypePolicies

GraphQL as state management

Full local state management capabilities

- **State**: managed by ApolloCache, along side with APIs data
- **Computed values**: TypePolicies
- **Actions**: reactive variables or client.writeQuery()

GraphQL as state management

Full local state management capabilities

- **State**: managed by ApolloCache, along side with APIs data
- **Computed values**: TypePolicies
- **Actions**: reactive variables or client.writeQuery()
- **Reactions**: Apollo ObservableQuery + reactive variables

GraphQL as state management

Full local state management capabilities

- **State**: managed by ApolloCache, along side with APIs data
- **Computed values**: TypePolicies
- **Actions**: reactive variables or client.writeQuery()
- **Reactions**: Apollo ObservableQuery + reactive variables
- **Tools:** Apollo Client Dev tools

The GraphQL innovations on front-end side

GraphQL Full-stack type safety

Photo by Taelynn Christopher

GraphQL Full-stack type safety

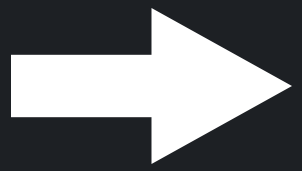
“ GraphQL introspection is a core
- most underrated - feature of the language

GraphQL Full-stack type safety


```
{
  __schema {
    types {
      name
      kind
      fields {
        name
        type {
          name
          ofType {
            name
          }
        }
      }
    }
  }
}
```

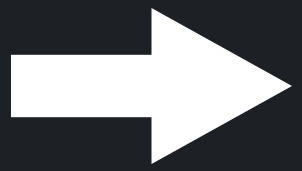
GraphQL Full-stack type safety


```
{
  __schema {
    types {
      name
      kind
      fields {
        name
        type {
          name
          ofType {
            name
          }
        }
      }
    }
  }
}
```



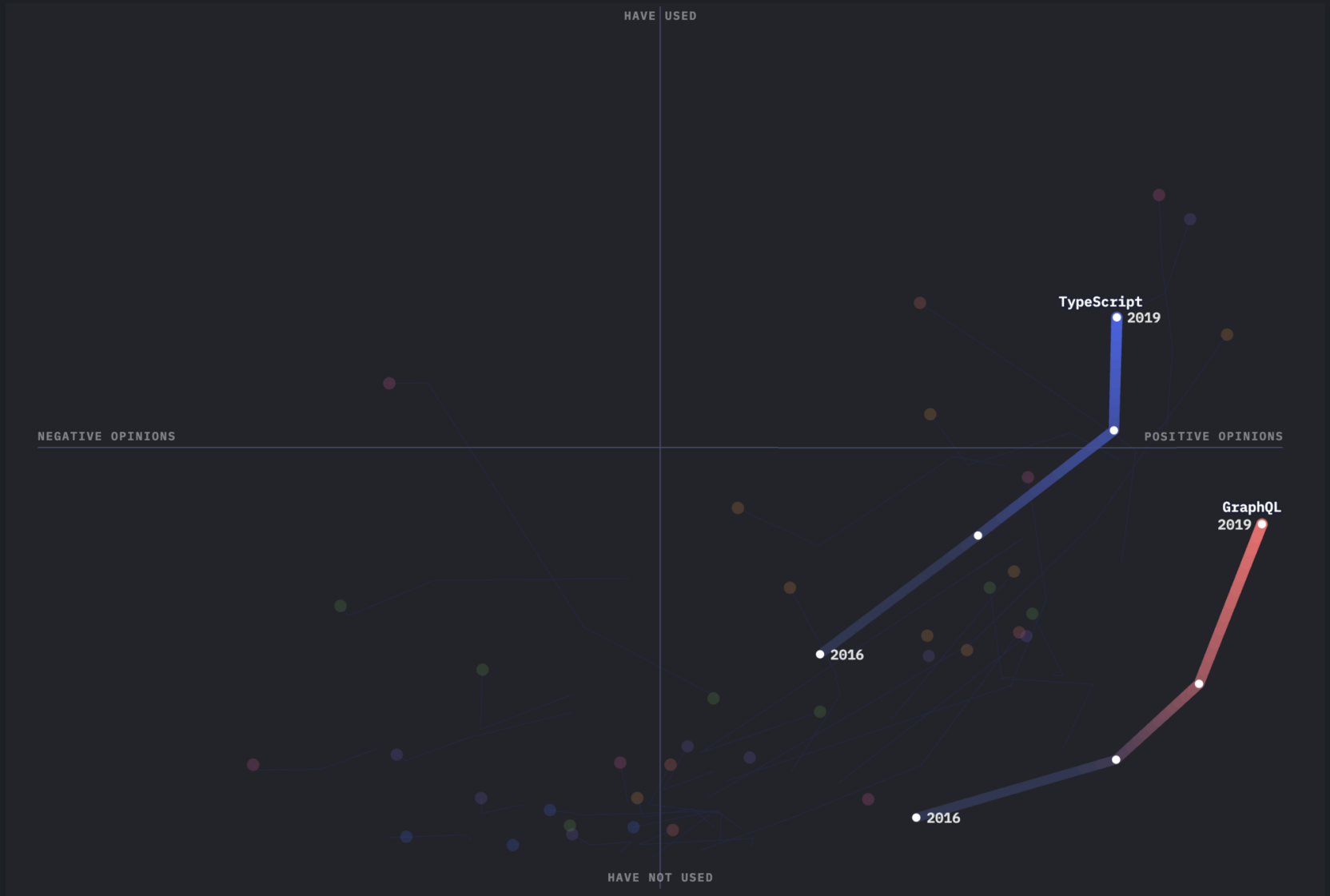
GraphQL Full-stack type safety


```graphql
{
  __schema {
    types {
      name
      kind
      fields {
        name
        type {
          name
          ofType {
            name
          }
        }
      }
    }
  }
}
```



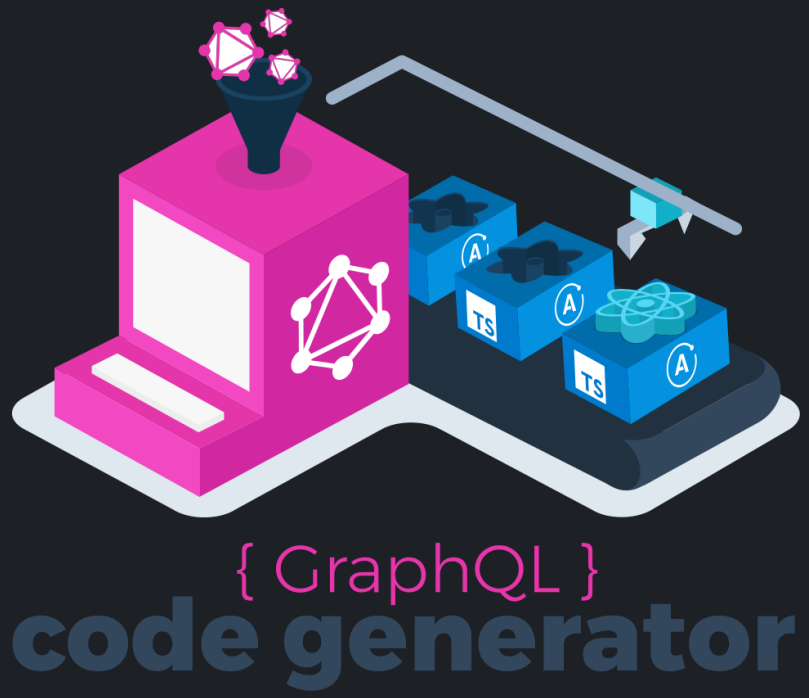
```json
{
  "data": {
    "__schema": {
      "types": [
        {
          "name": "Query",
          /* ... */
        },
        {
          "name": "PrivateUser",
          "kind": "OBJECT",
          "fields": [
            {
              "name": "id",
              "type": {
                "name": "String",
                "ofType": null
              }
            },
            {
              "name": "birthday",
              "type": {
                "name": "String",
                "ofType": null
              }
            },
            {
```

GraphQL Full-stack type safety



State of JS 2019

GraphQL Full-stack type safety

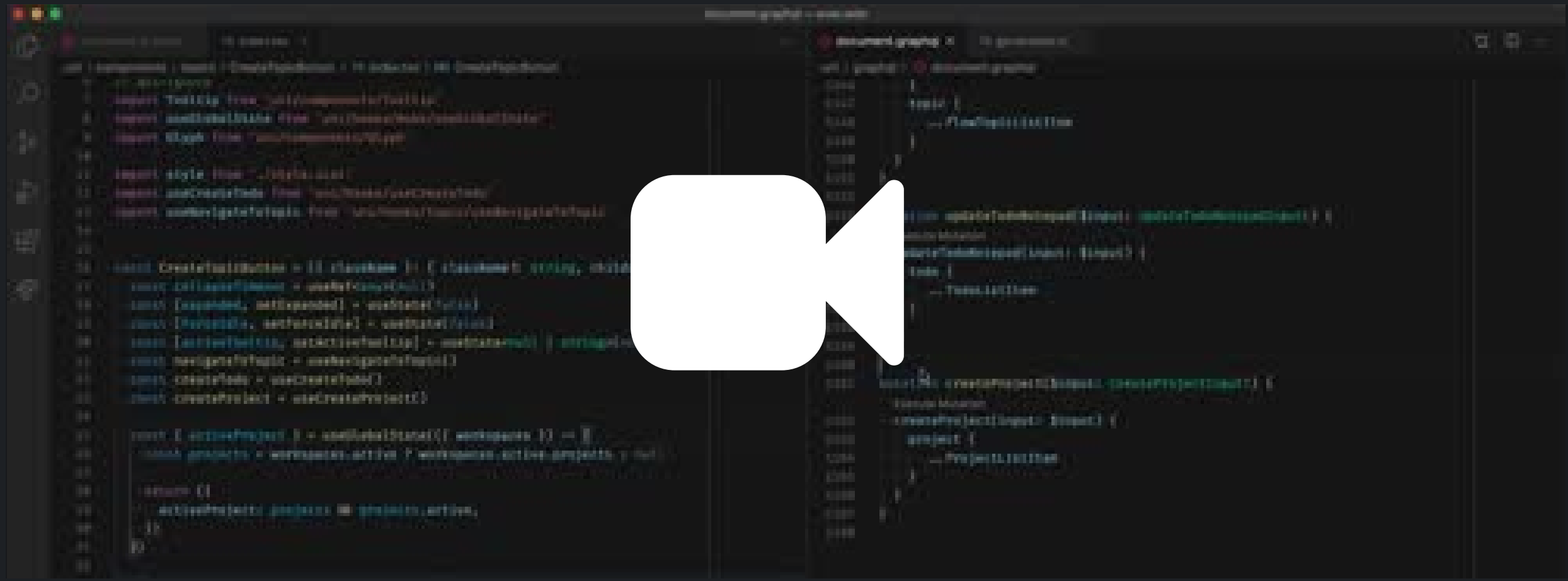


Given a GraphQL Schema, generates:

- TypeScript types definition
- React Apollo hooks definition
- urql components
- Types for Flow, Java, Kotlin

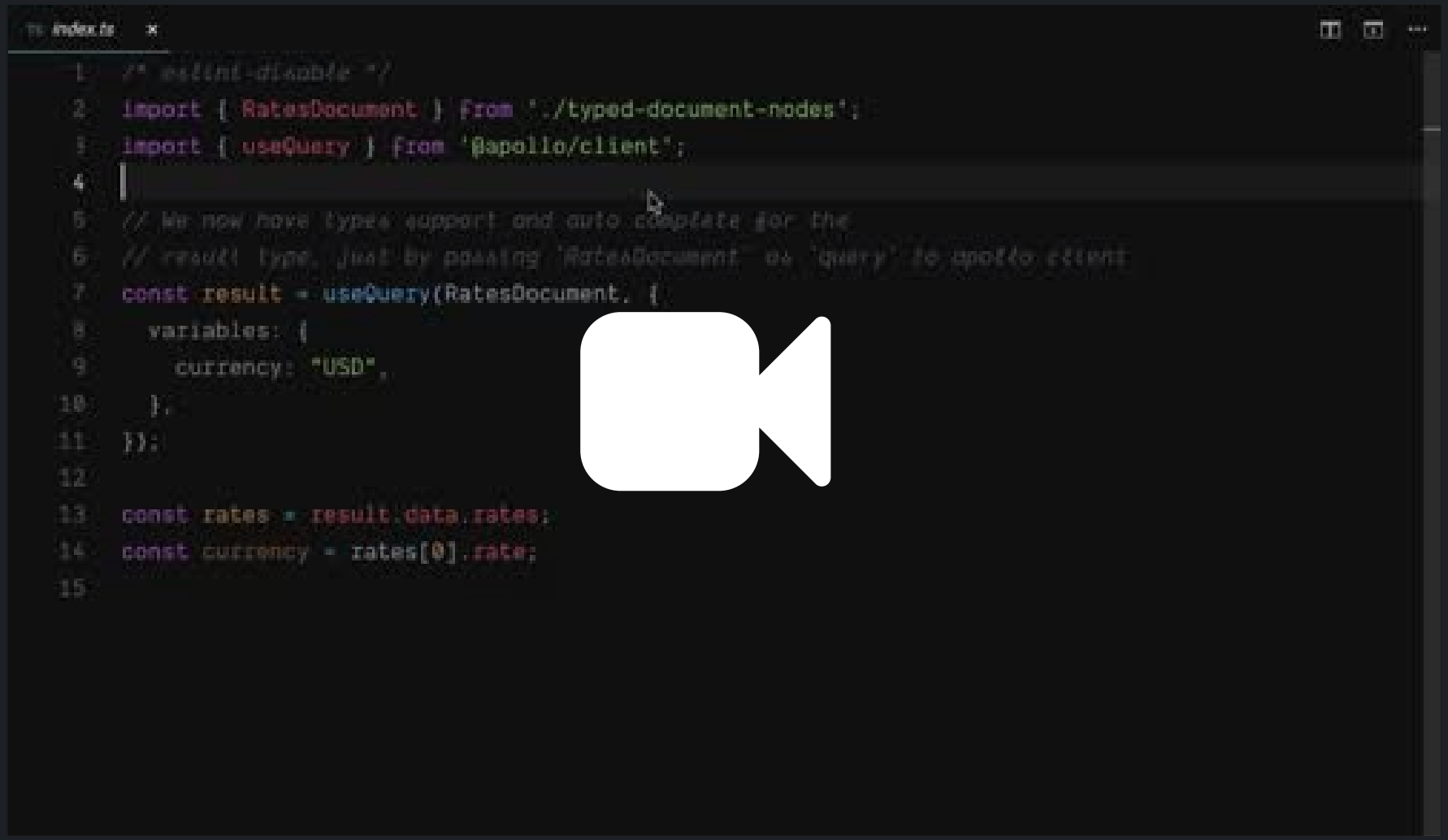
GraphQL Full-stack type safety

Typed React hooks generations



GraphQL Full-stack type safety

`TypedDocumentNode` for simplicity



GraphQL Full-stack type safety

> *We now have 'cheap to maintain'*
> *best-in-class TypeScript types in our front-end*

GraphQL (Beyond type safety intermission)

GraphQL (Beyond type safety intermission)

Better front-end developer experience

- GraphQL without queries
- GraphQL forms

GraphQL (Beyond type safety intermission)

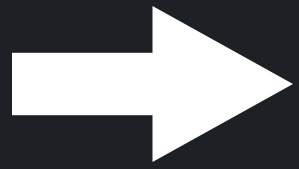
GraphQL without queries: gqless

```
const User = graphql(({ user }: { user: User }) =
  <div>
    <h2>{user.name}</h2>
    <img src={user.avatarUrl({ size: 100 })} />
  </div>
))

const App = graphql(() => (
  <div>
    {query.users.map(user => (
      <User key={user.id} user={user} />
    ))}
  </div>
))
```

GraphQL (Beyond type safety intermission)

GraphQL without queries: gqless

```
const User = graphql(({ user }: { user: User }) =
  <div>
    <h2>{user.name}</h2>
    <img src={user.avatarUrl({ size: 100 })} />
  </div>
))

const App = graphql(() => (
  <div>
    {query.users.map(user => (
      <User key={user.id} user={user} />
    ))}
  </div>
))
```



```
query App {
  users {
    id
    name
    avatarUrl(size: 100)
  }
}
```

+ TypeScript support

GraphQL (Beyond type safety intermission)

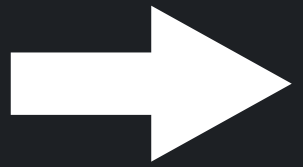
GraphQL forms: Frontier

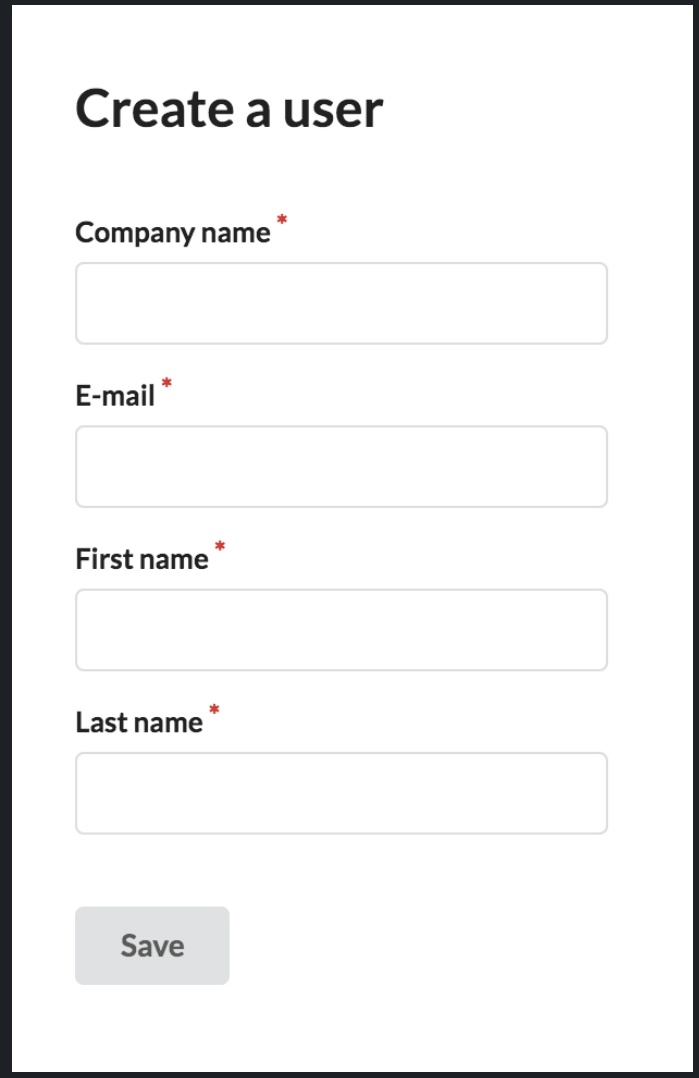
```
import gql from "graphql-tag";
import { Frontier } from "frontier-forms";
import { myApplicationKit } from "./uiKit";
import { client } from "./apollo-client";

const mutation = gql`
    mutation($user: User!) {
      createUser(user: $user) { id }
    }
`;

<Frontier
  client={client}
  mutation={mutation}
  uiKit={myApplicationKit}
/>
```

GraphQL (Beyond type safety intermission)

GraphQL forms: Frontier

```
import gql from "graphql-tag";
import { Frontier } from "frontier-forms";
import { myApplicationKit } from "./uiKit";
import { client } from "./apollo-client";

const mutation = gql`
    mutation($user: User!) {
      createUser(user: $user) { id }
    }
`;

<Frontier
  client={client}
  mutation={mutation}
  uiKit={myApplicationKit}
/>
```





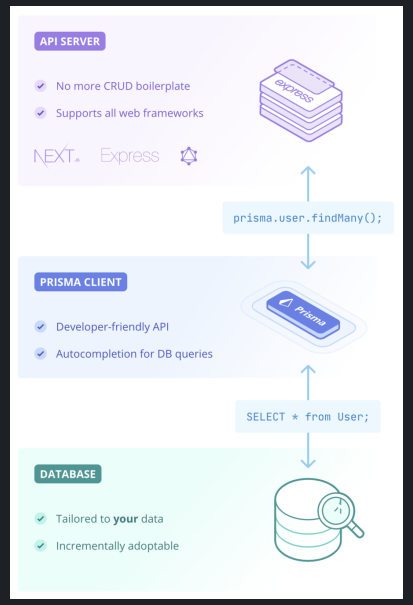
GraphQL bridging the gaps on server-side

Photo by Alexander Andrews

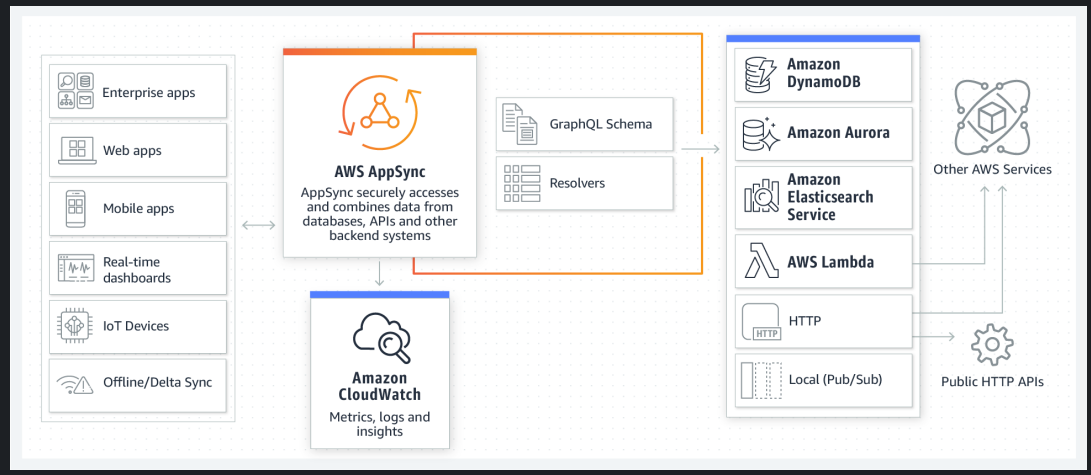
GraphQL bridging the gaps on server-side

Apollo

Prisma



AWS AppSync

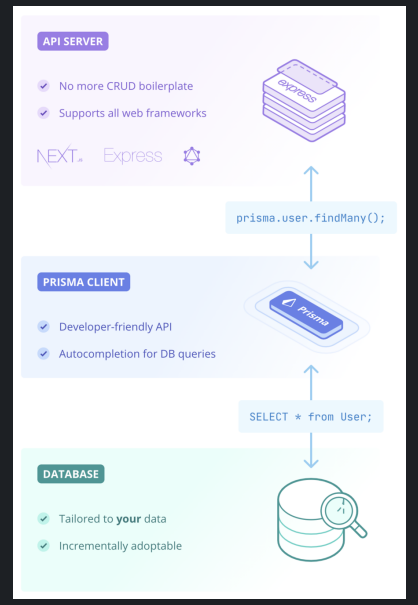


GraphQL bridging the gaps on server-side

Apollo

- What about more complex GraphQL APIs without a from-scratch development?

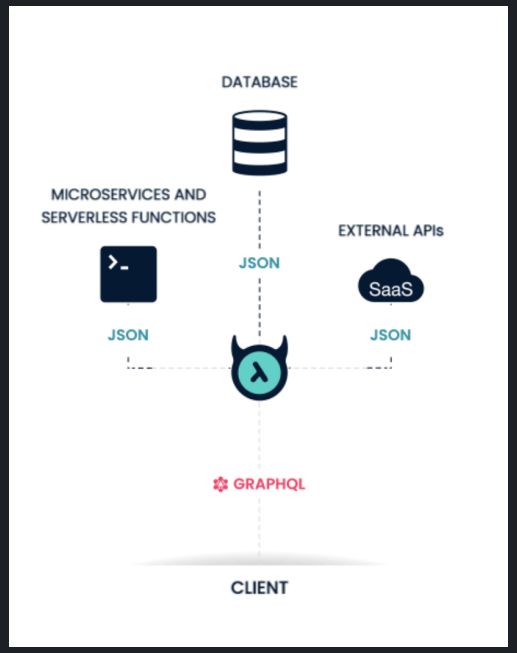
Prisma



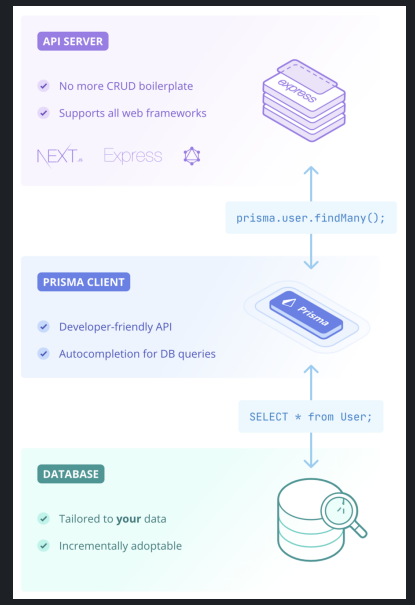
GraphQL bridging the gaps on server-side

Apollo

Hasura



Prisma



GraphQL bridging the gaps on server-side

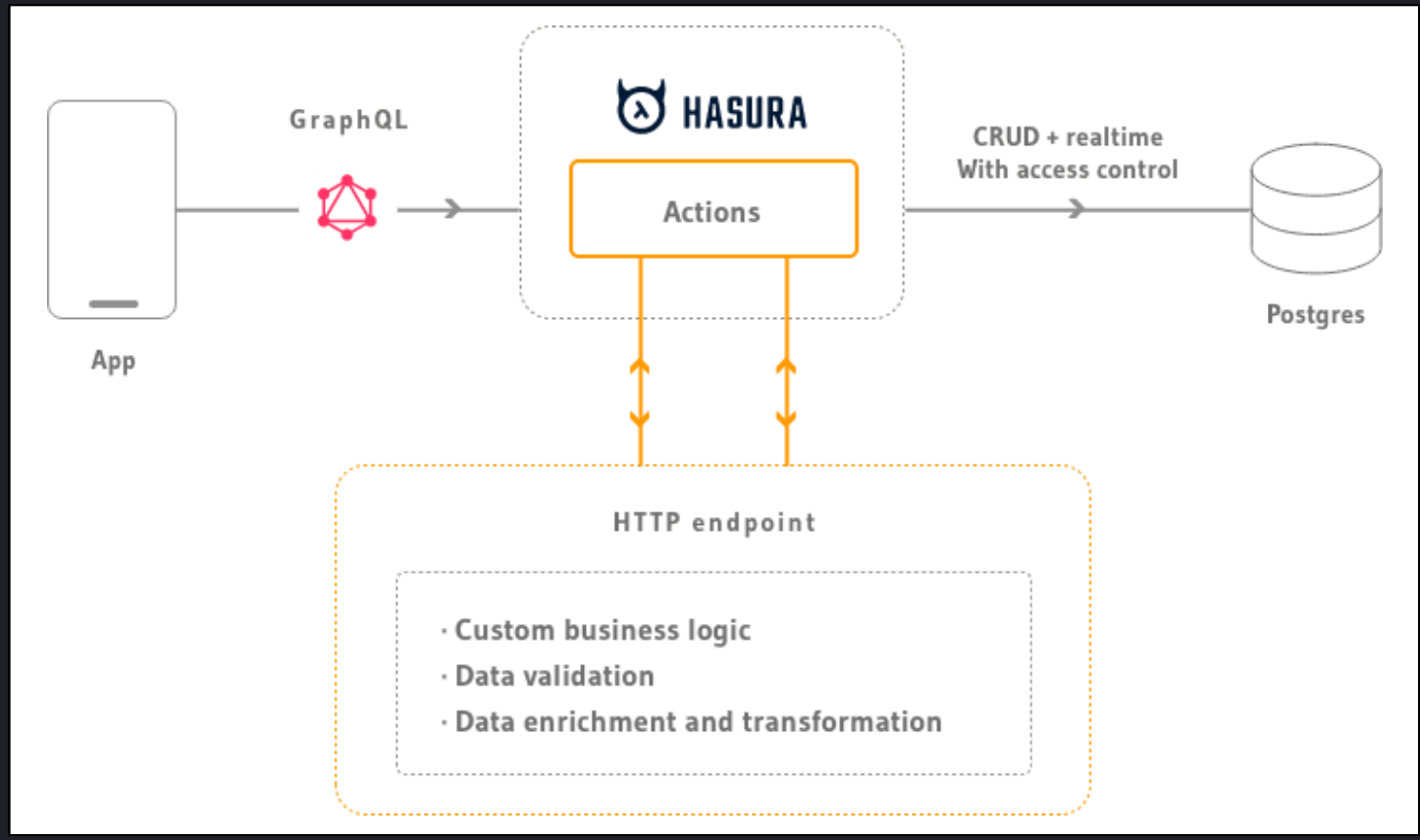
Hasura

Hasura translate GraphQL AST to SQL AST, providing blazing fast execution with minimum configuration

GraphQL bridging the gaps on server-side

Hasura

Custom logic support via Actions



GraphQL bridging the gaps on server-side

Hasura

GraphQL bridging the gaps on server-side

Hasura

- ACL support

GraphQL bridging the gaps on server-side

Hasura

- ACL support
- Authentication / Authorization (JWT)

GraphQL bridging the gaps on server-side

Hasura

- ACL support
- Authentication / Authorization (JWT)
- Remote schemas support (schema stitching)

GraphQL bridging the gaps on server-side

Hasura

- ACL support
- Authentication / Authorization (JWT)
- Remote schemas support (schema stitching)
- Subscriptions support

GraphQL bridging the gaps on server-side

Hasura

- ACL support
- Authentication / Authorization (JWT)
- Remote schemas support (schema stitching)
- Subscriptions support
- Actions for custom logic
- Trigger webhooks on database events

GraphQL bridging the gaps on server-side

Hasura

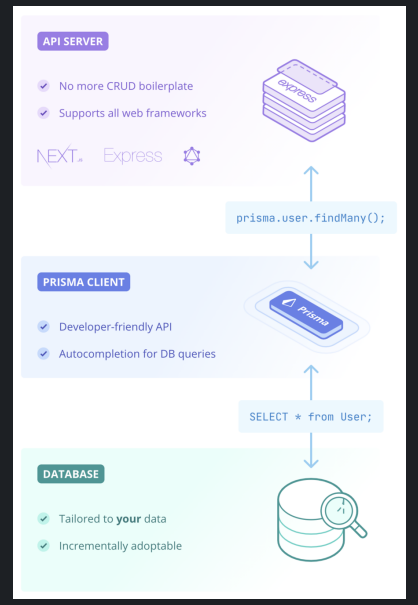
- ACL support
- Authentication / Authorization (JWT)
- Remote schemas support (schema stitching)
- Subscriptions support
- Actions for custom logic
- Trigger webhooks on database events
- Bonus: one-click install on most cloud providers

GraphQL bridging the gaps on server-side

Apollo

- What about old REST APIs and external services?

Prisma

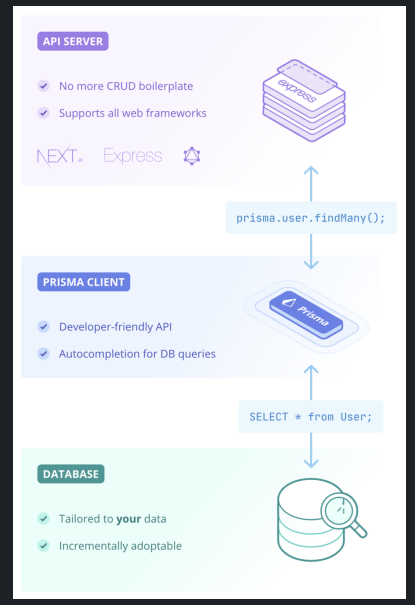


GraphQL bridging the gaps on server-side

Apollo

GraphQL Mesh

Prisma



GraphQL bridging the gaps on server-side

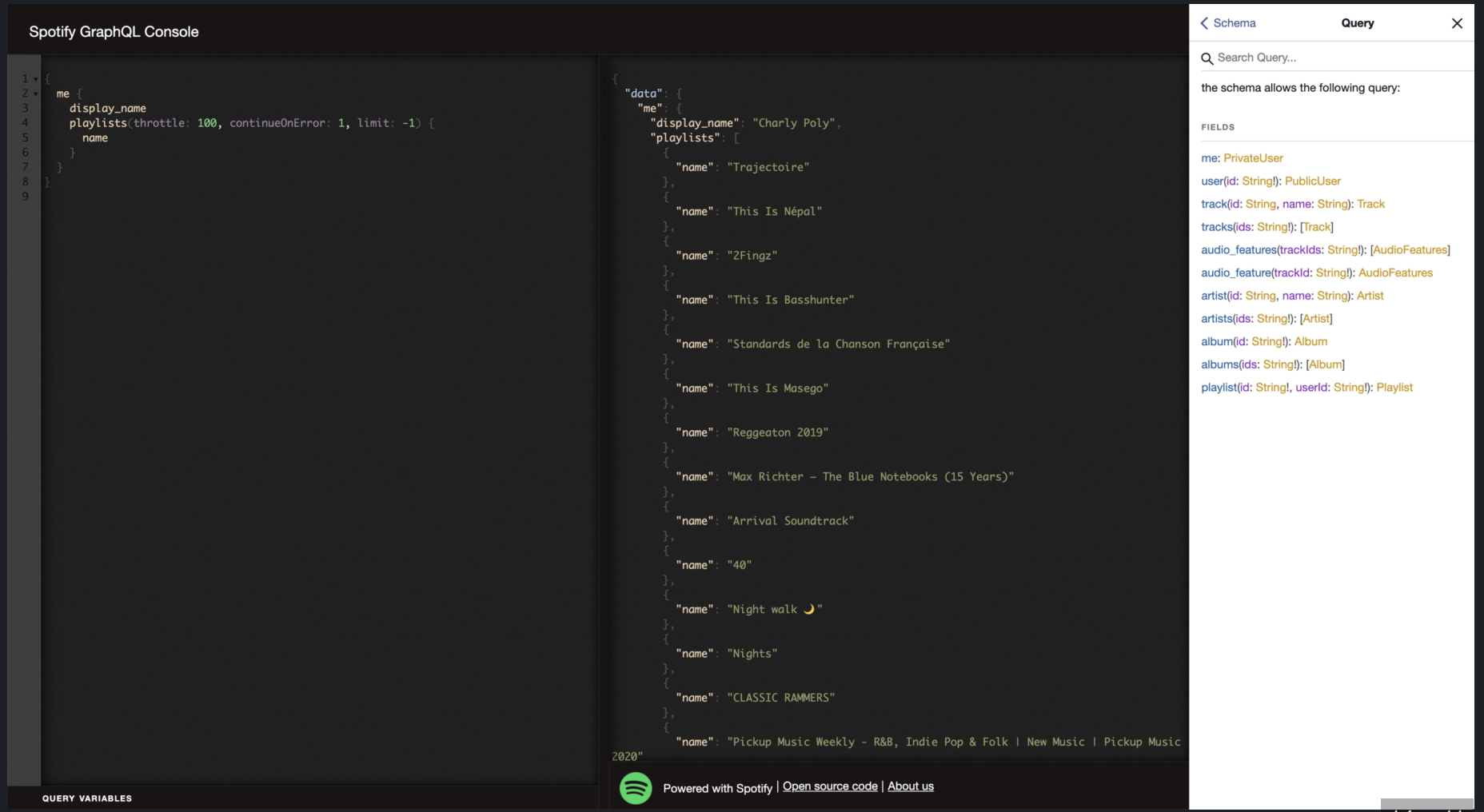
GraphQL Mesh

“ GraphQL Mesh leverages API definition standards to transform existing API to GraphQL APIs

GraphQL bridging the gaps on server-side

GraphQL Mesh

Spotify GraphQL in 5'



GraphQL bridging the gaps on server-side

GraphQL Mesh

Spotify GraphQL in 5'

GraphQL bridging the gaps on server-side

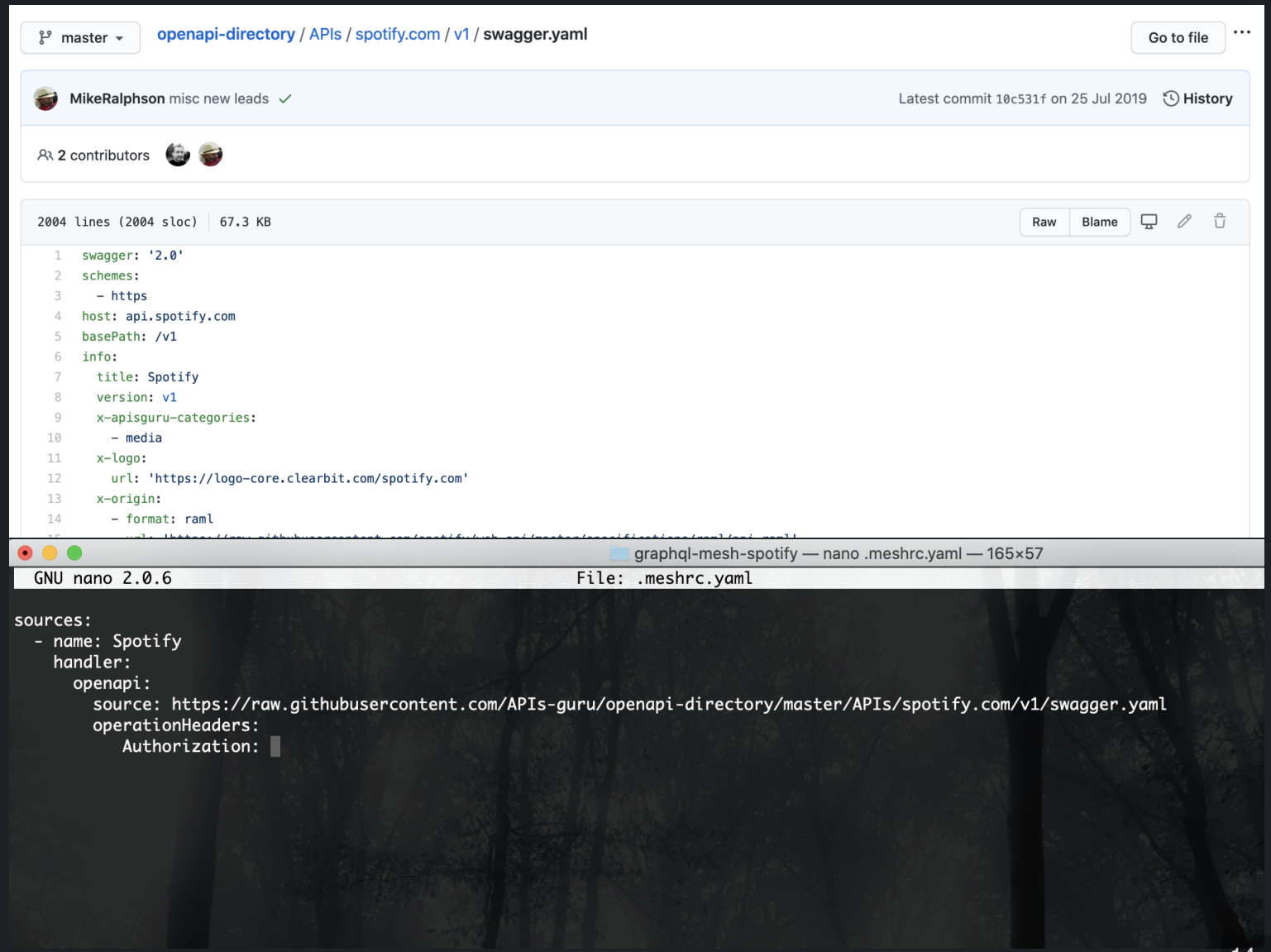
GraphQL Mesh

Spotify GraphQL in 5'

GraphQL bridging the gaps on server-side

GraphQL Mesh

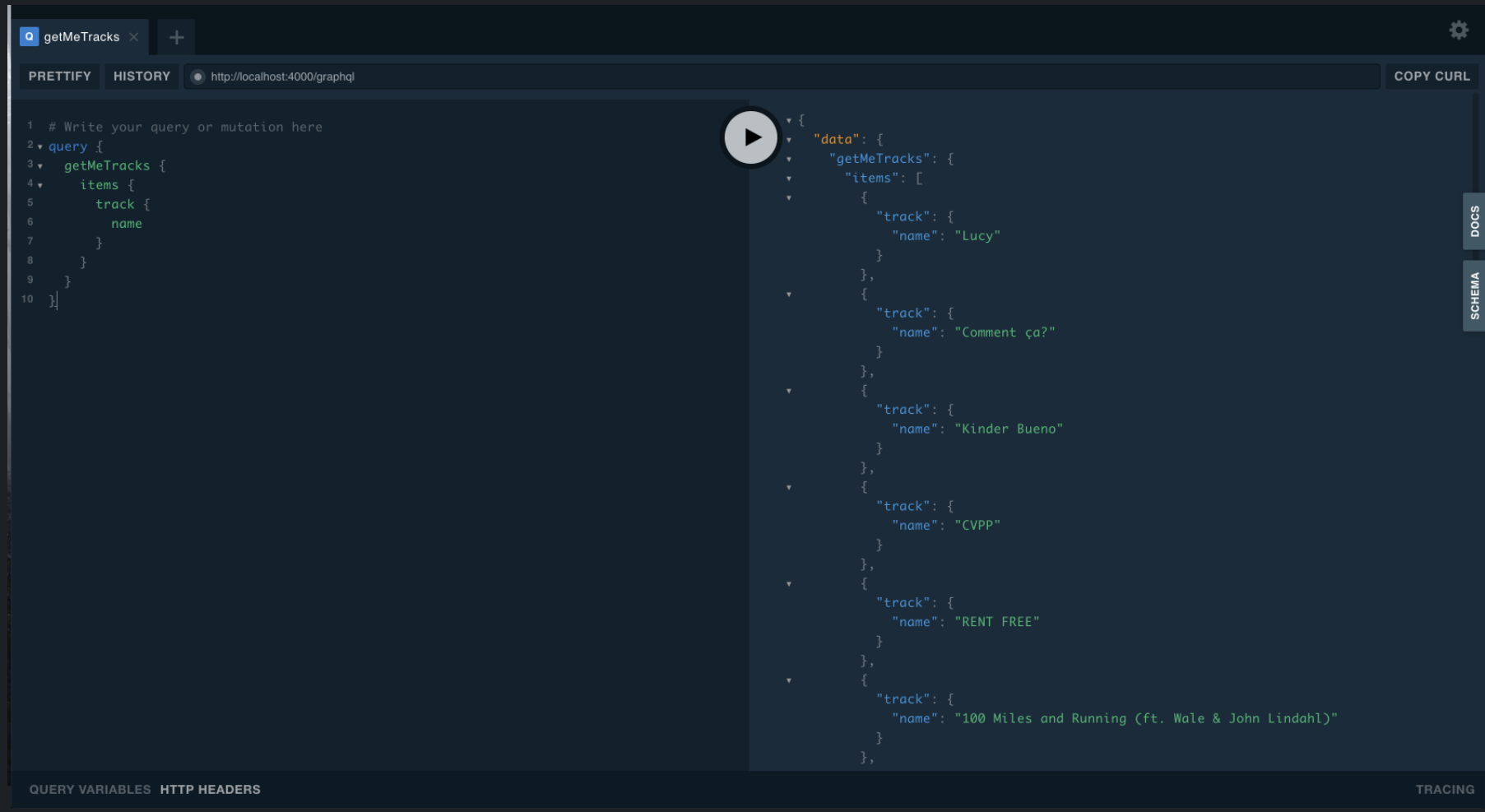
Spotify GraphQL in 5'



GraphQL bridging the gaps on server-side

GraphQL Mesh

Spotify GraphQL in 5'



GraphQL bridging the gaps on server-side

GraphQL Mesh

GraphQL bridging the gaps on server-side

GraphQL Mesh

- GraphQL proxy over your data

GraphQL bridging the gaps on server-side

GraphQL Mesh

- GraphQL proxy over your data
- Schema transformation

GraphQL bridging the gaps on server-side

GraphQL Mesh

- GraphQL proxy over your data
- Schema transformation
- Resolvers composition

GraphQL bridging the gaps on server-side

GraphQL Mesh

- GraphQL proxy over your data
- Schema transformation
- Resolvers composition
- Caching

GraphQL bridging the gaps on server-side

GraphQL Mesh

- GraphQL proxy over your data
- Schema transformation
- Resolvers composition
- Caching
- SDK generation

GraphQL bridging the gaps on server-side

GraphQL Mesh

- GraphQL proxy over your data
- Schema transformation
- Resolvers composition
- Caching
- SDK generation

GraphQL bridging the gaps on server-side

GraphQL Mesh

- GraphQL proxy over your data
- Schema transformation
- Resolvers composition
- Caching
- SDK generation

> ***GraphQL Mesh doesn't aim to create [...] public GraphQL schema [...], you should consider implementing another layer that exposes your public data [...].***

GraphQL bridging the gaps on server-side

GraphQL API building in 2020

GraphQL bridging the gaps on server-side

GraphQL API building in 2020

- Apollo, Prisma, and AWS AppSync are not unique solutions

GraphQL bridging the gaps on server-side

GraphQL API building in 2020

- Apollo, Prisma, and AWS AppSync are not unique solutions
- Hasura helps you build performant and powerful API

GraphQL bridging the gaps on server-side

GraphQL API building in 2020

- Apollo, Prisma, and AWS AppSync are not unique solutions
- Hasura helps you build performant and powerful API
- GraphQL Mesh brings you universal GraphQL

GraphQL in 2020 ⚡

Wrap-up

GraphQL in 2020 ⚡

Beyond a ”Facebook & Apollo technology”

- The Guild
- Hasura
- Dgraph
- and many more individual contributors!

GraphQL in 2020 ⚡

Beyond fetching data from the client-side

- Better front-end developer experience (Full-stack type safety, ...)
- Easier complex GraphQL API building (Hasura)
- Fast GraphQL Database (Dgraph, Hasura)
- GraphQL as a universal data language (state, GraphQL Mesh)

GraphQL: Looking forward

GraphQL in 2021?

- Subscriptions / Async evolutions
  - Apollo 3, Hasura
  - @defer, @live
- Smarter client-side cache?
- Better mobile clients ecosystem?
- more public APIs?

Thank you!

slides on noti.st/charlypoly

@whereischarly

@wittydeveloper

Photo by Nicolas Gras