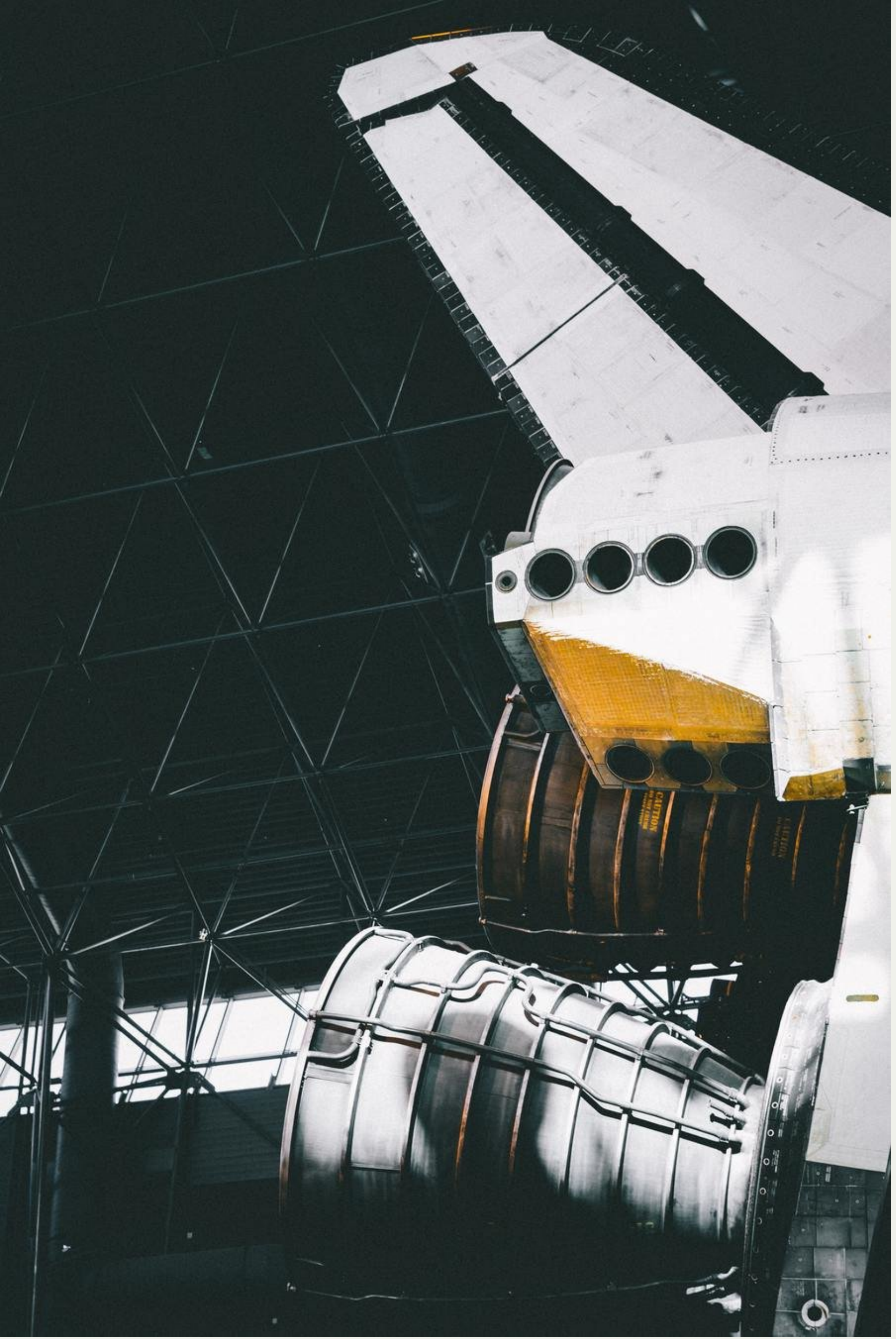


GraphQL fantastic four

photo by Meriç Dağlı

Who am I

Charly POLY

Who am I

Charly POLY

- Sr. Software Engineer at Double
- Former Tech lead at Algolia

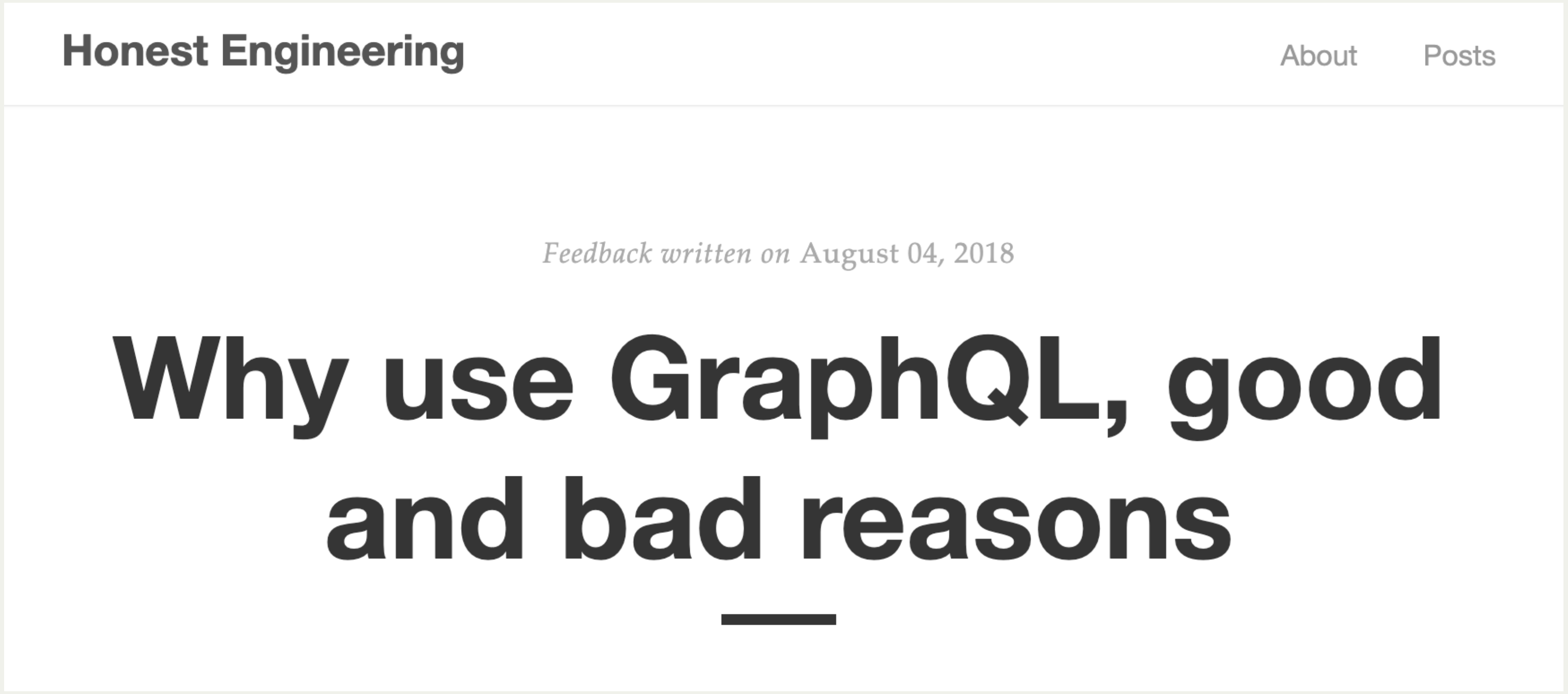
Who am I

Charly POLY

- Sr. Software Engineer at Double
- Former Tech lead at Algolia

- Started using GraphQL 4 years ago

On GraphQL



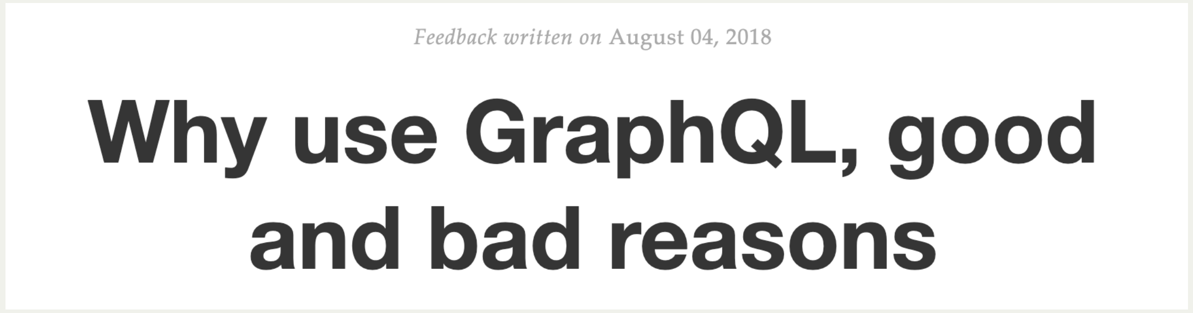
Honest Engineering

About Posts

Feedback written on August 04, 2018

Why use GraphQL, good and bad reasons

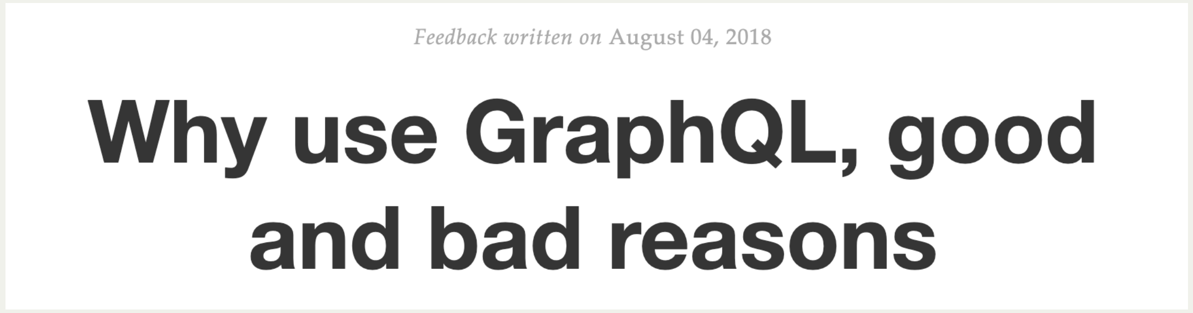
On GraphQL



Feedback written on August 04, 2018

Why use GraphQL, good and bad reasons

On GraphQL

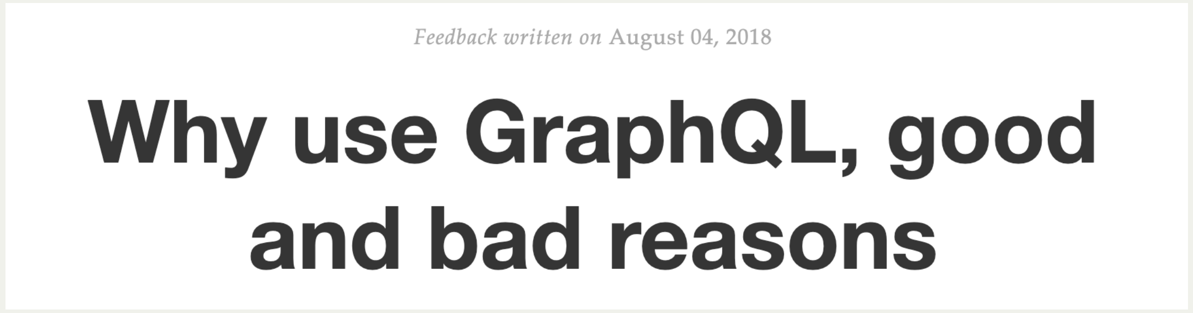


Feedback written on August 04, 2018

Why use GraphQL, good and bad reasons

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

On GraphQL



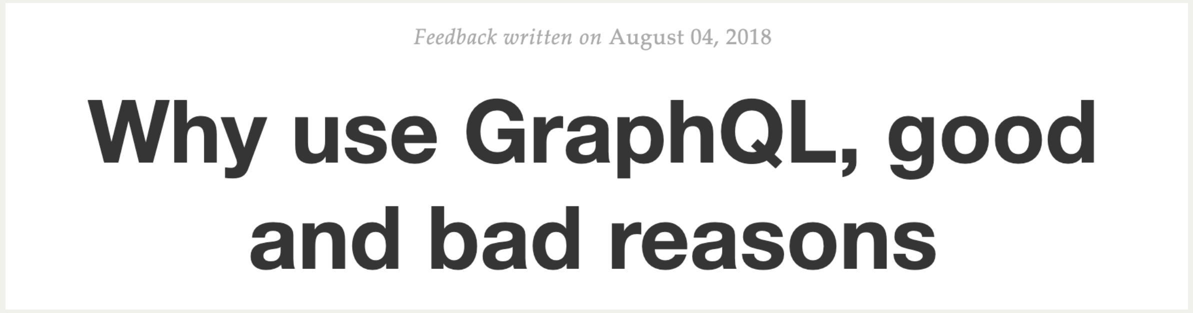
Feedback written on August 04, 2018

Why use GraphQL, good and bad reasons

- **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

On GraphQL


Feedback written on August 04, 2018

Why use GraphQL, good and bad reasons

- **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

On GraphQL

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

GraphQL as application state management

photo by jesse orrico

GraphQL as application state management

GraphQL as application state management

GraphQL as application state management

GraphQL as application state management

GraphQL as application state management

GraphQL as application state management

1. GraphQL is a "data query and manipulation language for APIs"

GraphQL as application state management

1. GraphQL is a "data query and manipulation language for APIs"

2. What if your state behave like a local API?

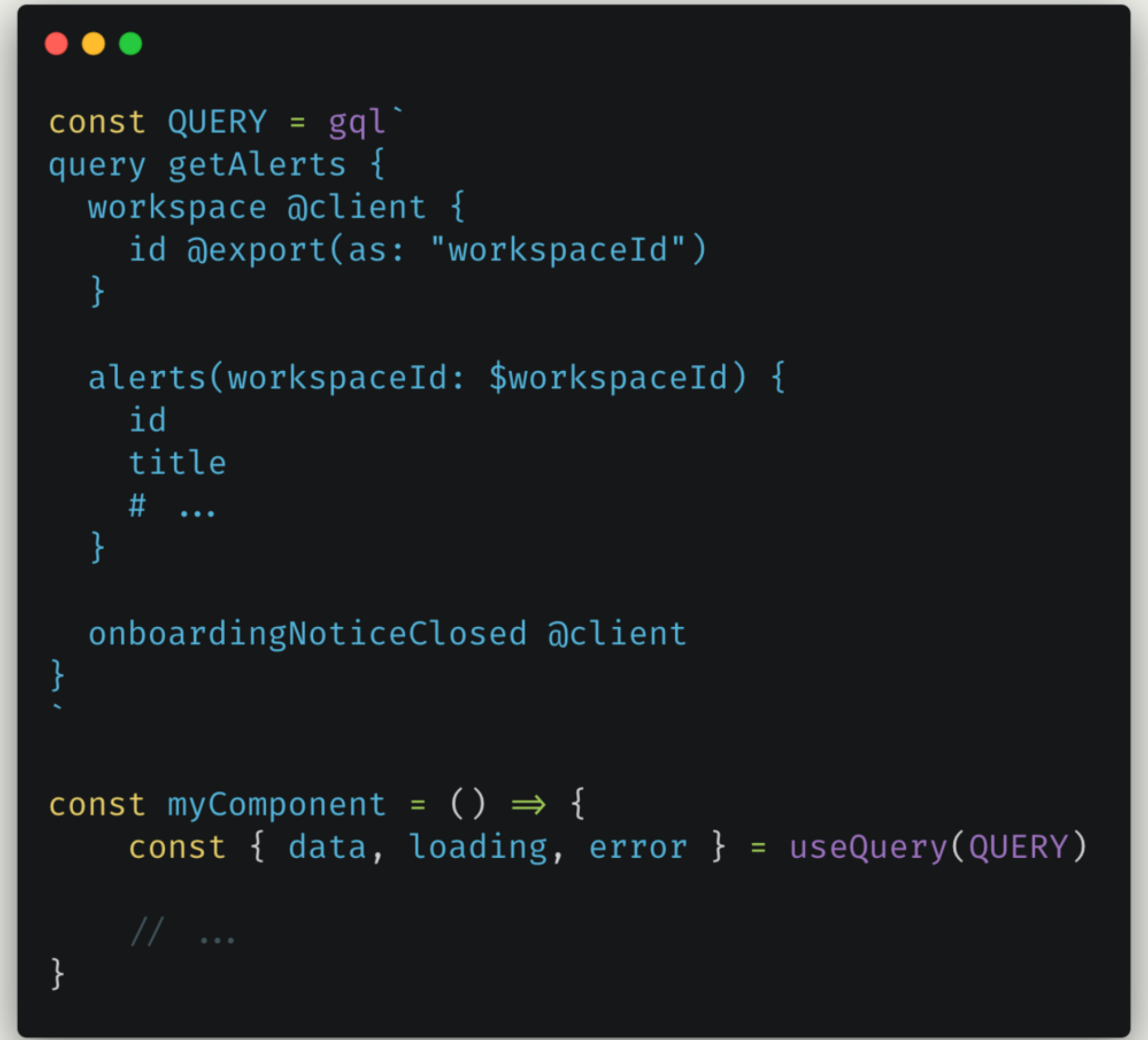
GraphQL as application state management

1. GraphQL is a "data query and manipulation language for APIs"

2. What if your state behave like a local API?

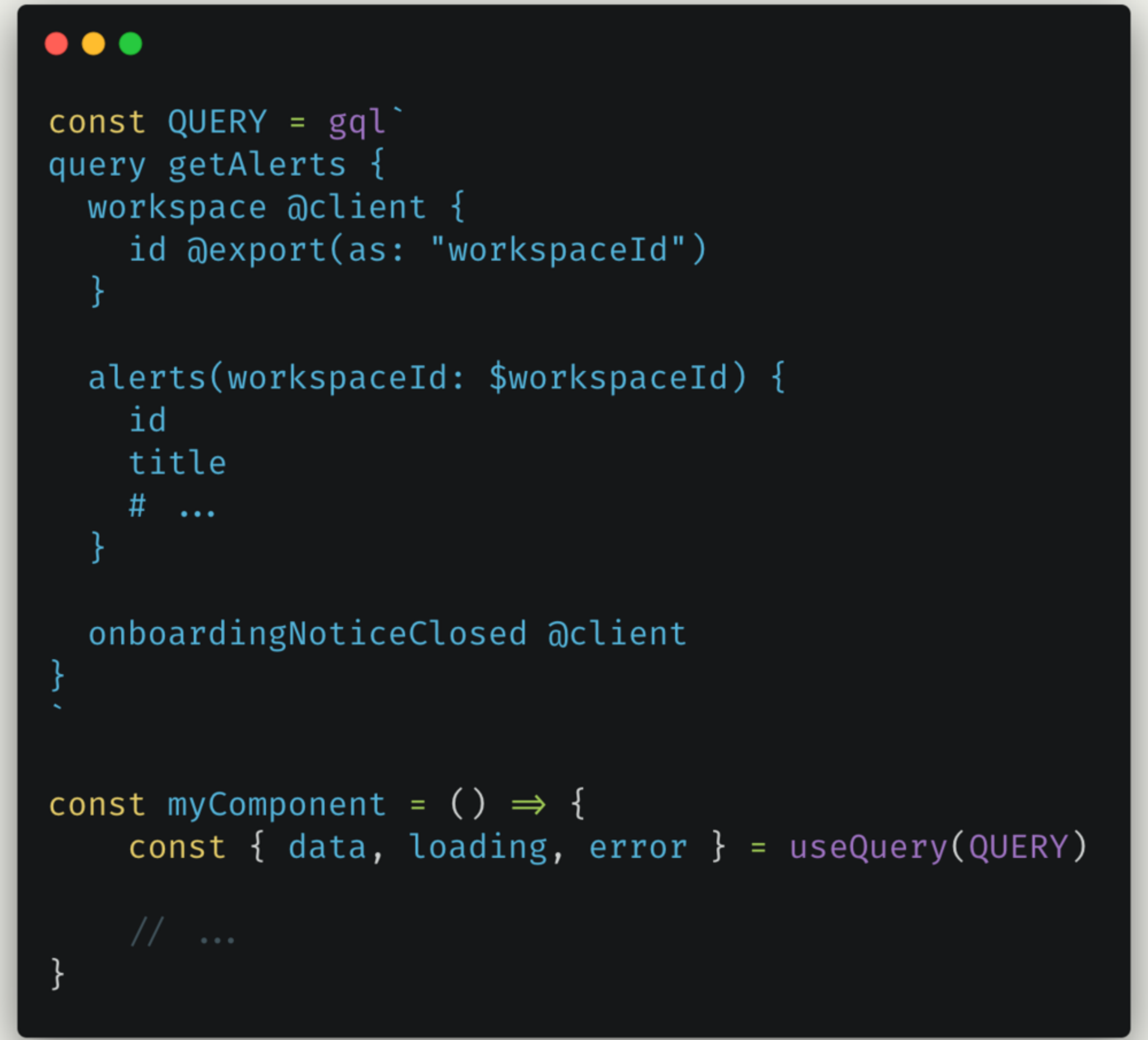
3. ➡️ Apollo GraphQL Local state management

GraphQL as application state management



```
const QUERY = gql`
query getAlerts {
  workspace @client {
    id @export(as: "workspaceId")
  }

  alerts(workspaceId: $workspaceId) {
    id
    title
    # ...
  }

  onboardingNoticeClosed @client
}
`

const myComponent = () => {
    const { data, loading, error } = useQuery(QUERY)

    // ...
}
```

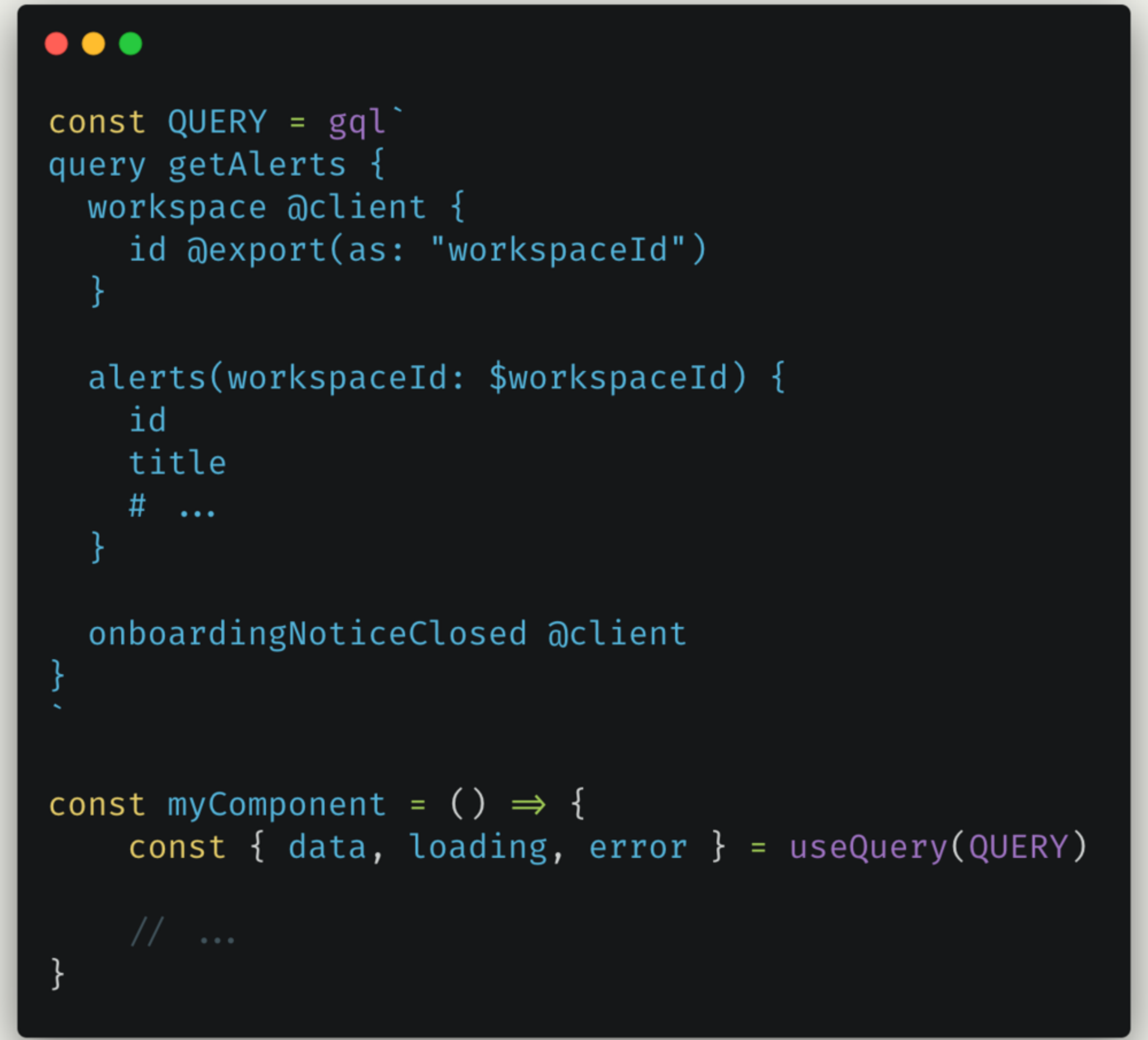
GraphQL as application state management



```
const QUERY = gql`
query getAlerts {
  workspace @client {
    id @export(as: "workspaceId")
  }

  alerts(workspaceId: $workspaceId) {
    id
    title
    # ...
  }

  onboardingNoticeClosed @client
}
`

const myComponent = () ⇒ {
    const { data, loading, error } = useQuery(QUERY)

    // ...
}
```

- **@client** directive for local state

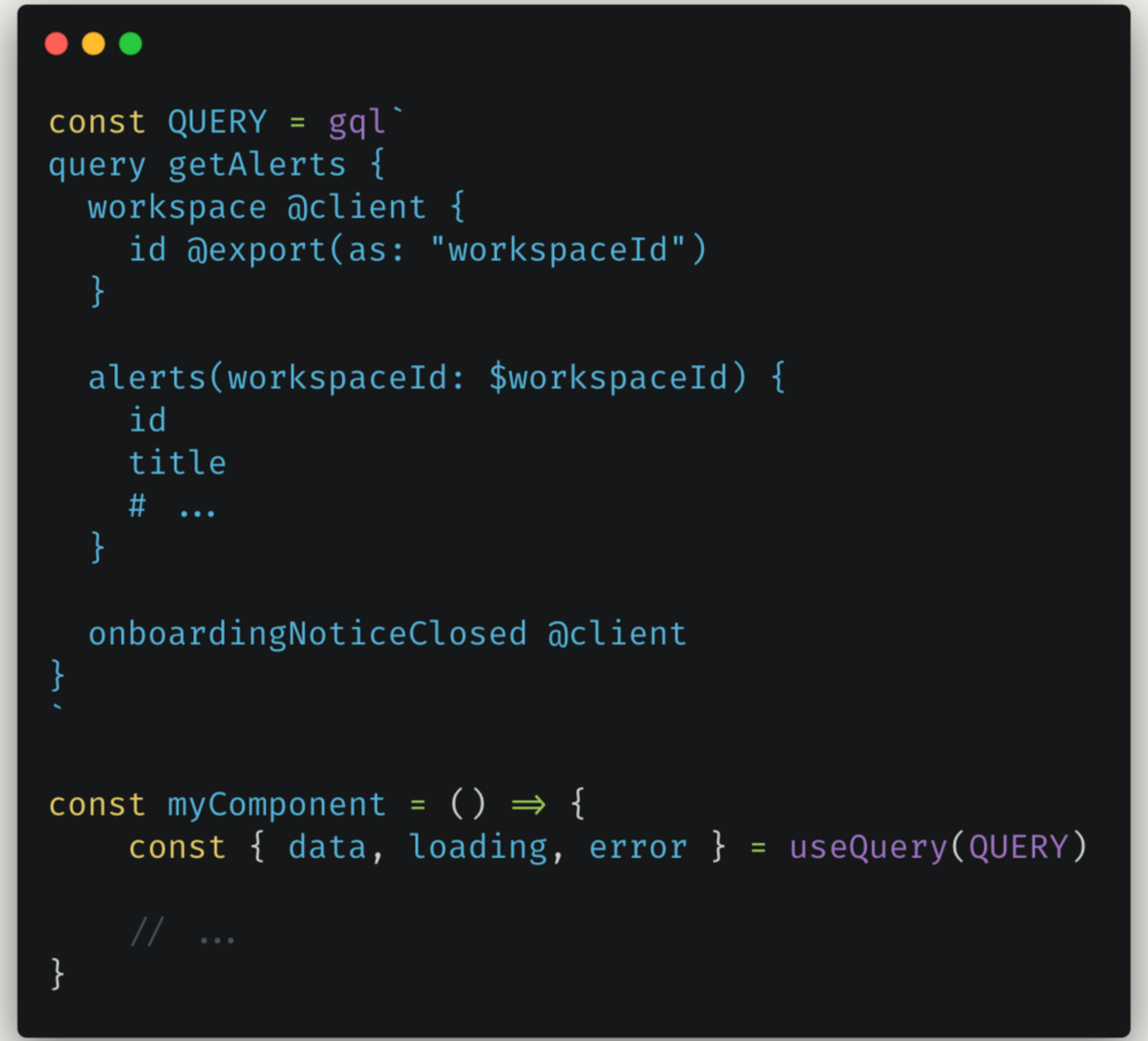
GraphQL as application state management



```
const QUERY = gql`
query getAlerts {
  workspace @client {
    id @export(as: "workspaceId")
  }

  alerts(workspaceId: $workspaceId) {
    id
    title
    # ...
  }

  onboardingNoticeClosed @client
}
`

const myComponent = () => {
    const { data, loading, error } = useQuery(QUERY)

    // ...
}
```

- **@client** directive for local state
- One language and hooks set for all data

GraphQL as application state management

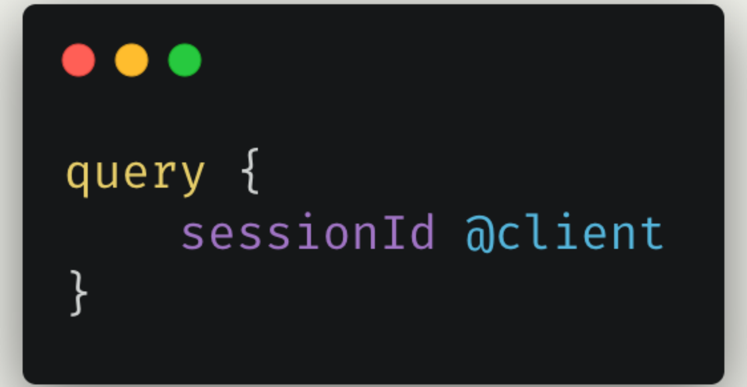


```
const QUERY = gql`
query getAlerts {
  workspace @client {
    id @export(as: "workspaceId")
  }

  alerts(workspaceId: $workspaceId) {
    id
    title
    # ...
  }

  onboardingNoticeClosed @client
}
`

const myComponent = () => {
    const { data, loading, error } = useQuery(QUERY)

    // ...
}
```

- **@client** directive for local state
- One language and hooks set for all data
- Local fields as variables

GraphQL as application state management

Local scalar values

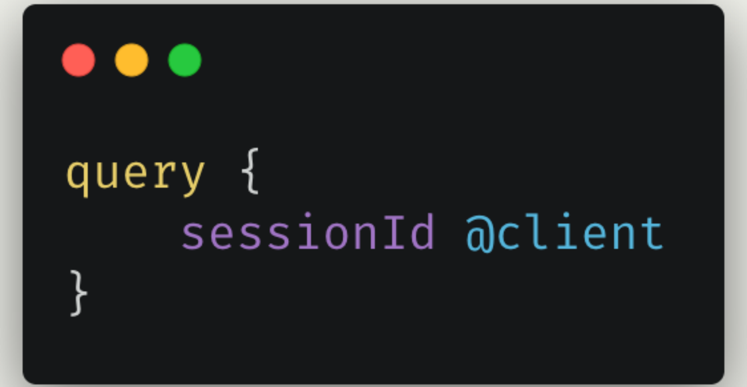


```
query {
    sessionId @client
}
```

- query without local resolver
- use "client.writeData()" to initialize and update state

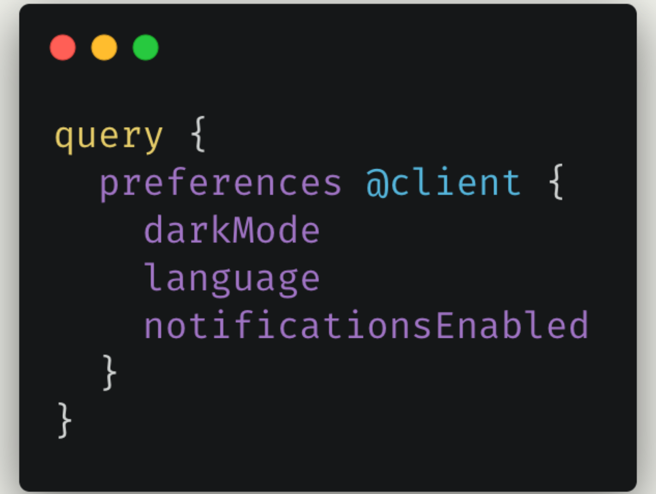
GraphQL as application state management

Local scalar values


```
query {
    sessionId @client
}
```

- query without local resolver
- use "client.writeData()" to initialize and update state

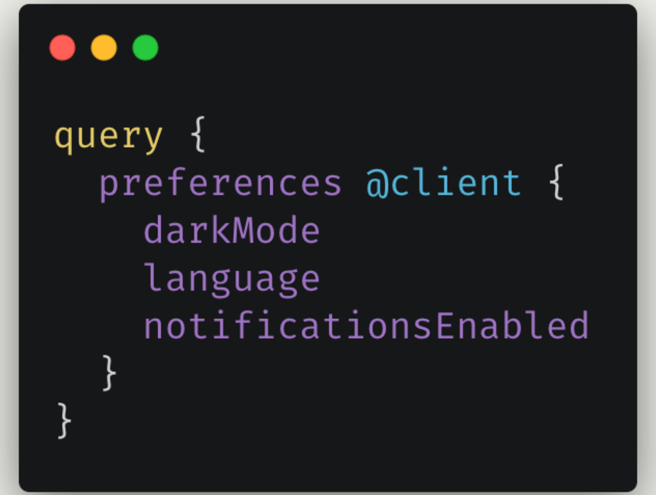
Local complex values or computed values


```
query {
  preferences @client {
    darkMode
    language
    notificationsEnabled
  }
}
```

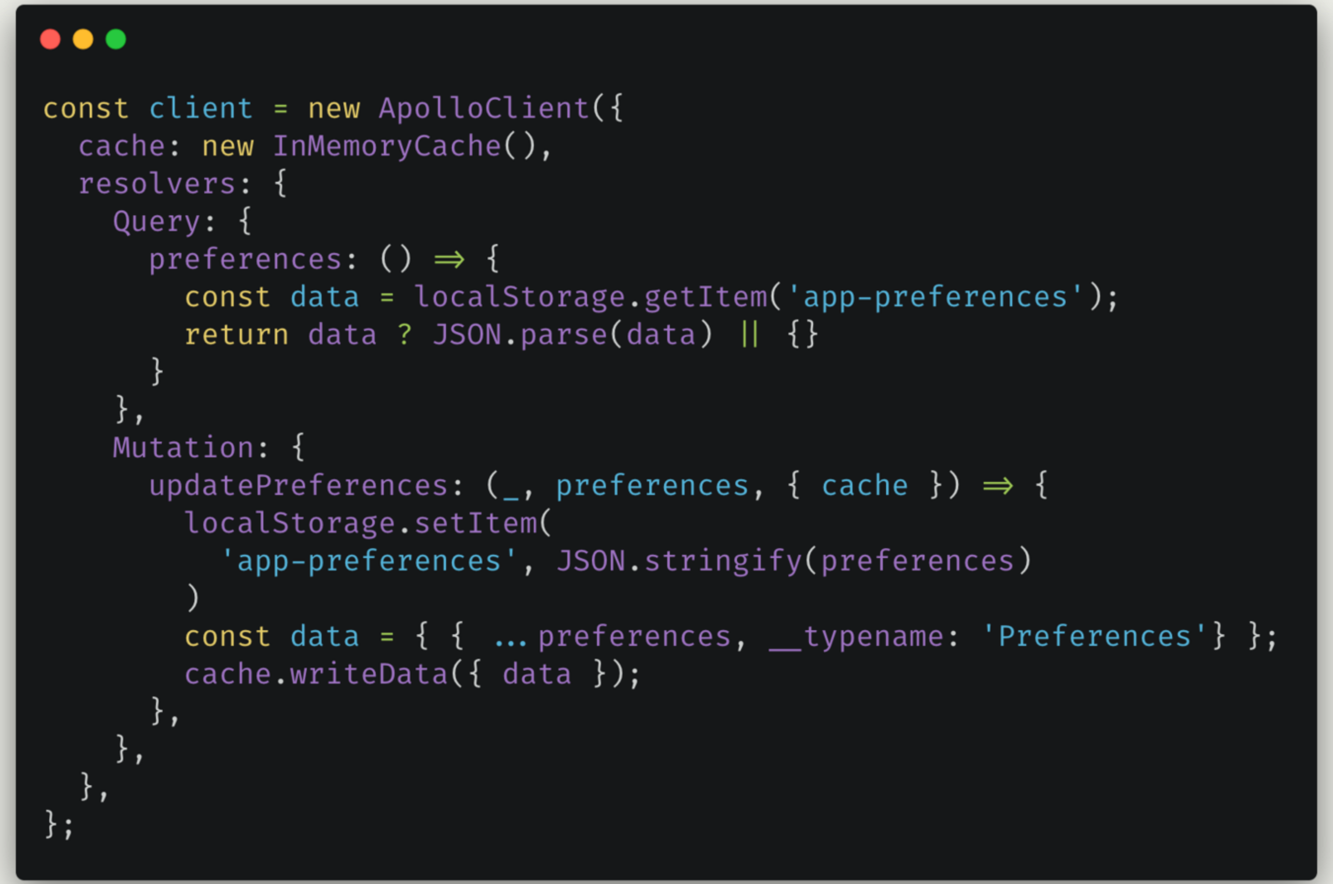
- local mutations
- local resolvers
- APC 3 TypePolicy (read)

GraphQL as application state management

Local complex values or computed values



```
query {
  preferences @client {
    darkMode
    language
    notificationsEnabled
  }
}
```



```
const client = new ApolloClient({
  cache: new InMemoryCache(),
  resolvers: {
    Query: {
      preferences: () ⇒ {
        const data = localStorage.getItem('app-preferences');
        return data ? JSON.parse(data) || {}
      }
    },
    Mutation: {
      updatePreferences: (_, preferences, { cache }) ⇒ {
        localStorage.setItem(
          'app-preferences', JSON.stringify(preferences)
        )
        const data = { { ...preferences, __typename: 'Preferences'} };
        cache.writeData({ data });
      },
    },
  },
};
```

GraphQL as application state management

Full local state management capabilities

GraphQL as application state management

Full local state management capabilities

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

GraphQL as application state management

Full local state management capabilities

- **State**: managed by ApolloCache, along side with APIs data
- **Computed values**: local resolvers (or APC 3 TypePolicy)

GraphQL as application state management

Full local state management capabilities

- **State**: managed by ApolloCache, along side with APIs data
- **Computed values**: local resolvers (or APC 3 TypePolicy)
- **Actions**: mutations or client.writeQuery()

GraphQL as application state management

Full local state management capabilities

- **State**: managed by ApolloCache, along side with APIs data
- **Computed values**: local resolvers (or APC 3 TypePolicy)
- **Actions**: mutations or client.writeQuery()
- **Reactions**: Apollo ObservableQuery

GraphQL as application state management

Full local state management capabilities

- **State**: managed by ApolloCache, along side with APIs data
- **Computed values**: local resolvers (or APC 3 TypePolicy)
- **Actions**: mutations or client.writeQuery()
- **Reactions**: Apollo ObservableQuery
- **Tools:** Apollo Client Dev tools

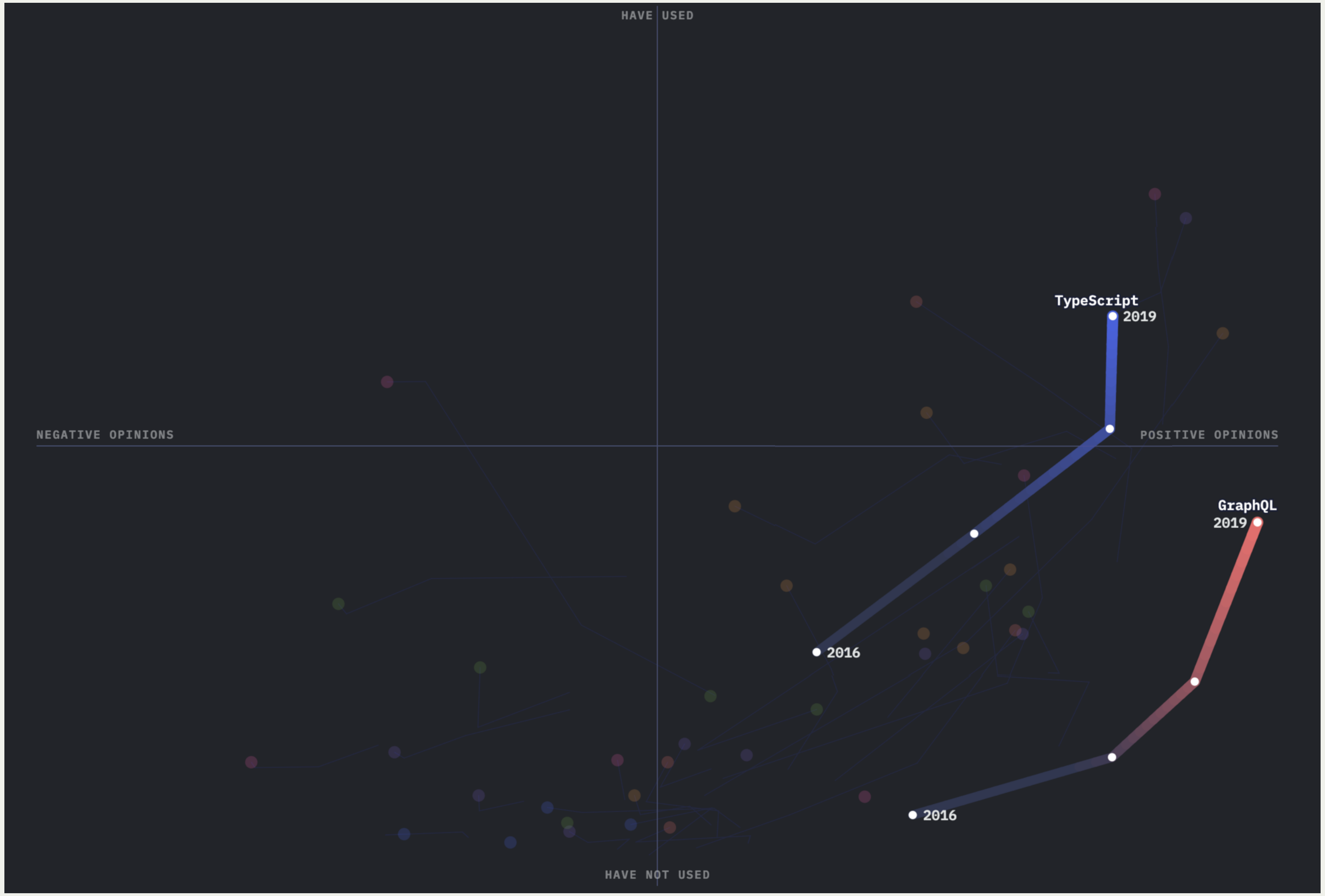
GraphQL generation powers

photo by Esteban Lopez

GraphQL generation powers

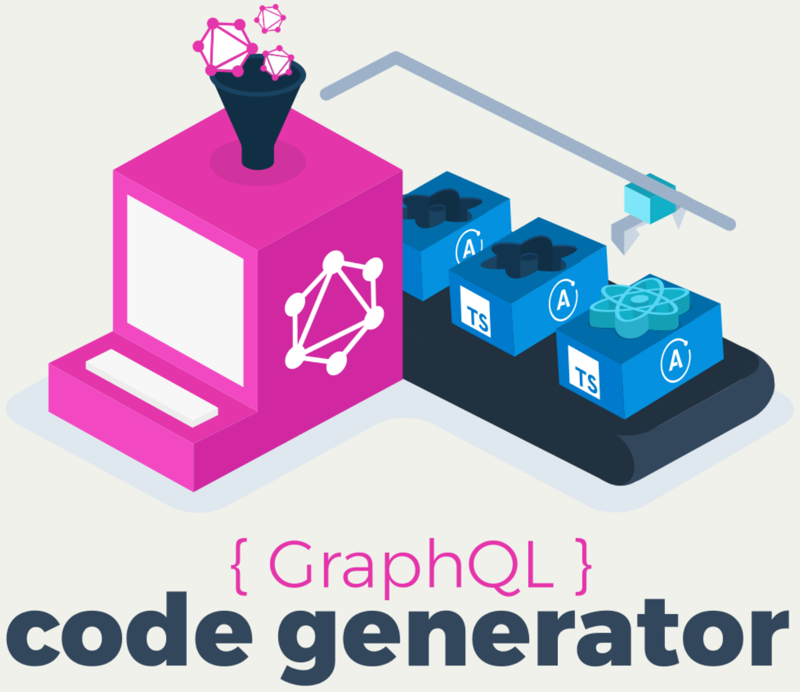
“ GraphQL introspection is a core - most underrated - feature of the language

GraphQL generation powers



State of JS 2019

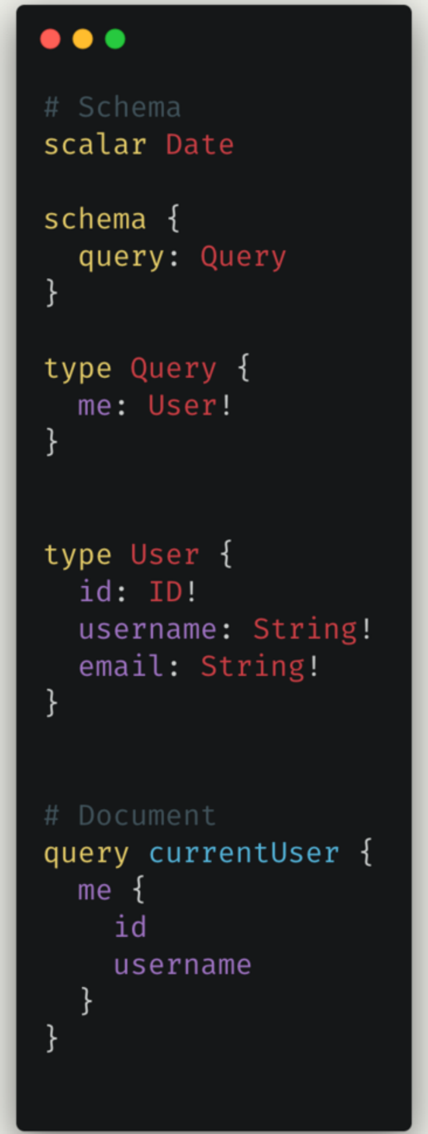
GraphQL generation powers



Given a GraphQL Schema, generates:

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

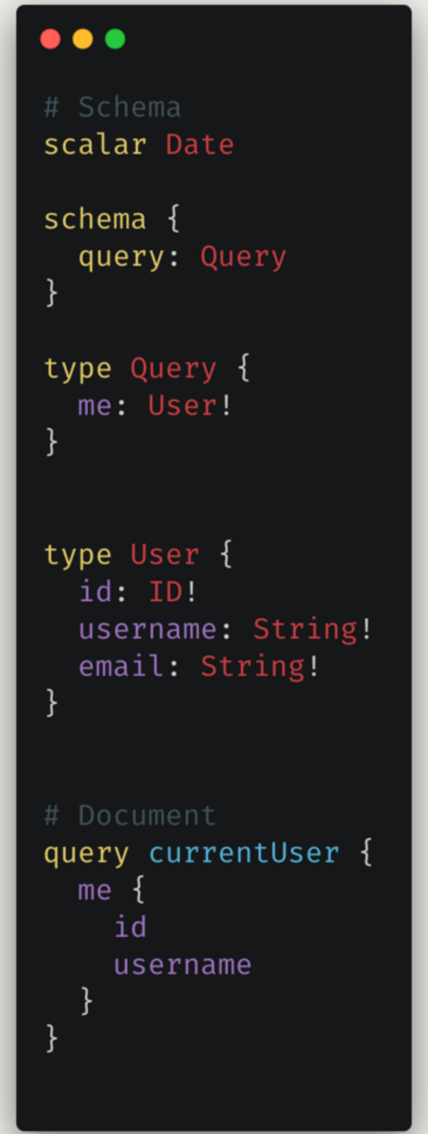
GraphQL generation powers



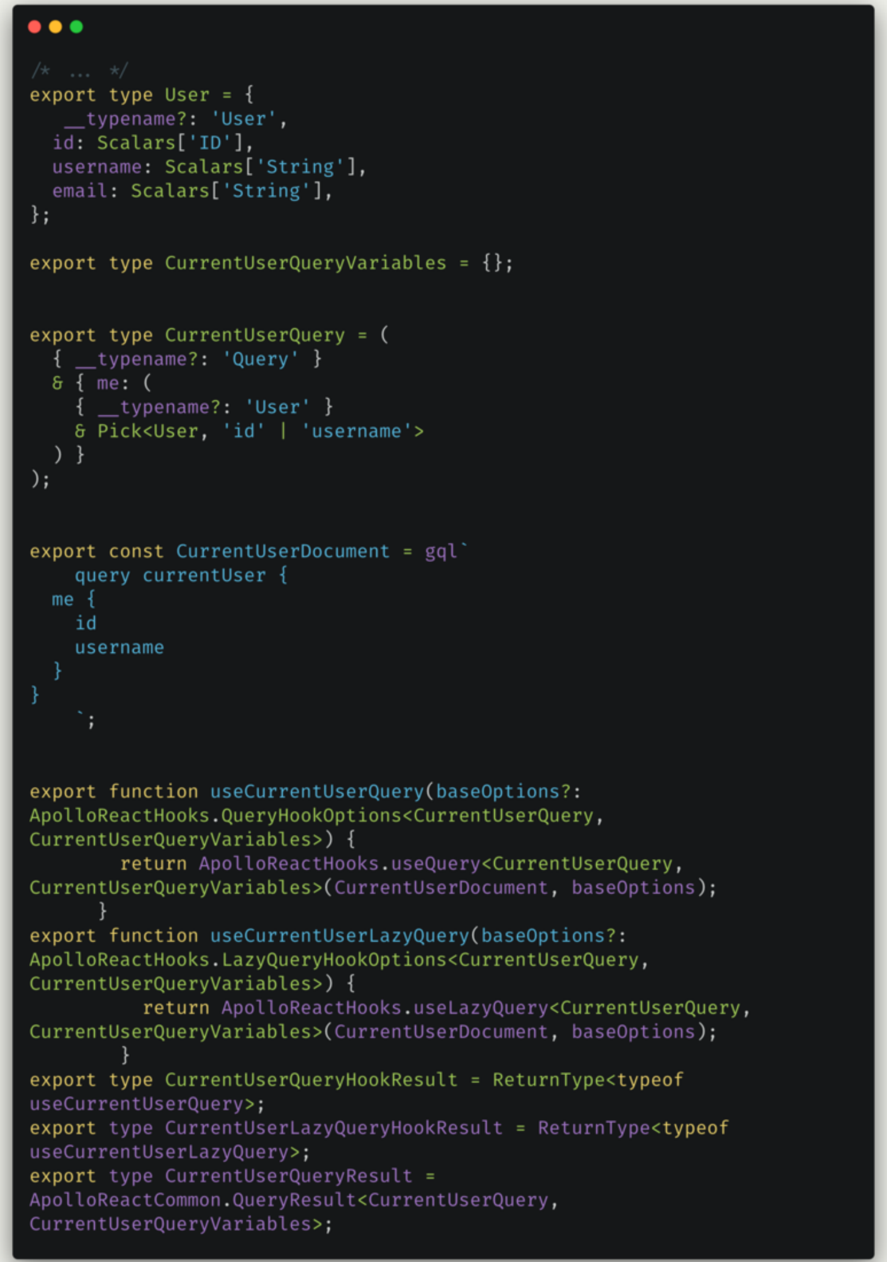
```graphql
# Schema
scalar Date

schema {
  query: Query
}

type Query {
  me: User!
}


type User {
  id: ID!
  username: String!
  email: String!
}


# Document
query currentUser {
  me {
    id
    username
  }
}
```

https://graphql-code-generator.com

GraphQL generation powers



```graphql
# Schema
scalar Date

schema {
  query: Query
}

type Query {
  me: User!
}


type User {
  id: ID!
  username: String!
  email: String!
}


# Document
query currentUser {
  me {
    id
    username
  }
}
```



```typescript
/* ... */
export type User = {
   __typename?: 'User',
  id: Scalars['ID'],
  username: Scalars['String'],
  email: Scalars['String'],
};

export type CurrentUserQueryVariables = {};


export type CurrentUserQuery = (
  { __typename?: 'Query' }
  & { me: (
    { __typename?: 'User' }
    & Pick<User, 'id' | 'username'>
  ) }
);


export const CurrentUserDocument = gql`
    query currentUser {
  me {
    id
    username
  }
}
    `;


export function useCurrentUserQuery(baseOptions?:
ApolloReactHooks.QueryHookOptions<CurrentUserQuery,
CurrentUserQueryVariables>) {
        return ApolloReactHooks.useQuery<CurrentUserQuery,
CurrentUserQueryVariables>(CurrentUserDocument, baseOptions);
      }
export function useCurrentUserLazyQuery(baseOptions?:
ApolloReactHooks.LazyQueryHookOptions<CurrentUserQuery,
CurrentUserQueryVariables>) {
          return ApolloReactHooks.useLazyQuery<CurrentUserQuery,
CurrentUserQueryVariables>(CurrentUserDocument, baseOptions);
        }
export type CurrentUserQueryHookResult = ReturnType<typeof
useCurrentUserQuery>;
export type CurrentUserLazyQueryHookResult = ReturnType<typeof
useCurrentUserLazyQuery>;
export type CurrentUserQueryResult =
ApolloReactCommon.QueryResult<CurrentUserQuery,
CurrentUserQueryVariables>;
```

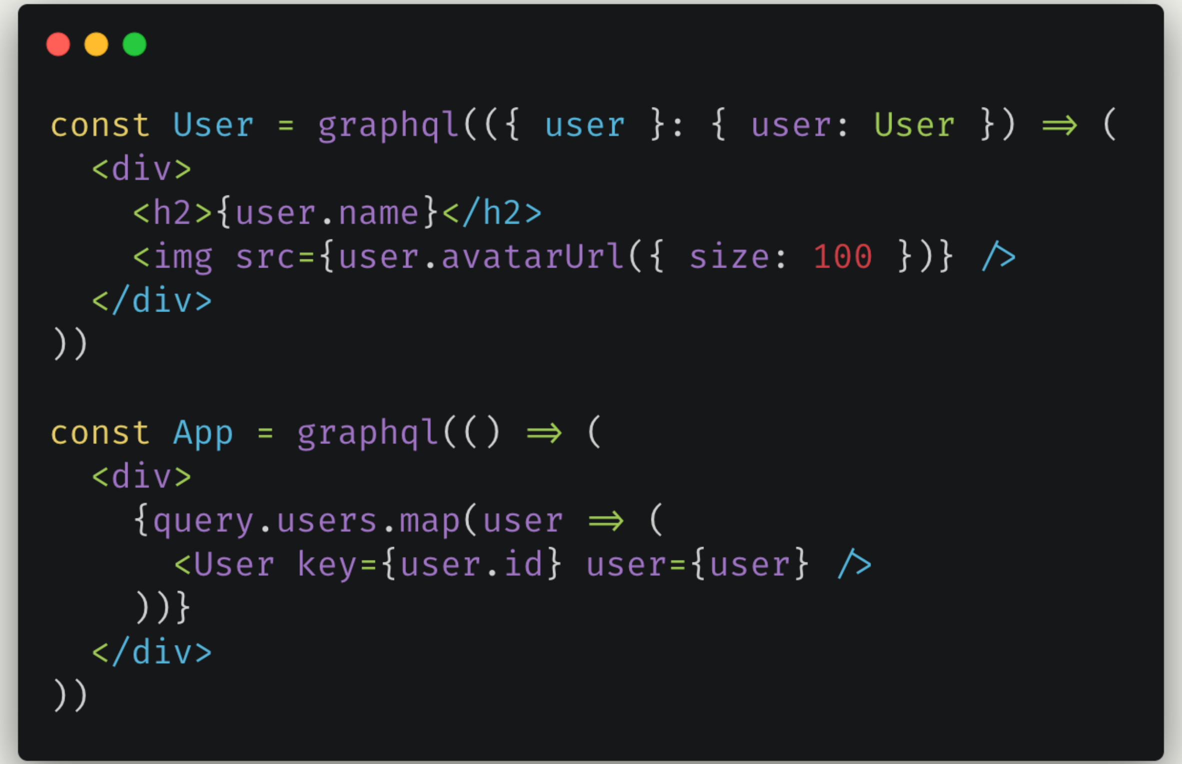
https://graphql-code-generator.com

GraphQL generation powers

GraphQL without query definition

GraphQL generation powers

GQLess: GraphQL without queries

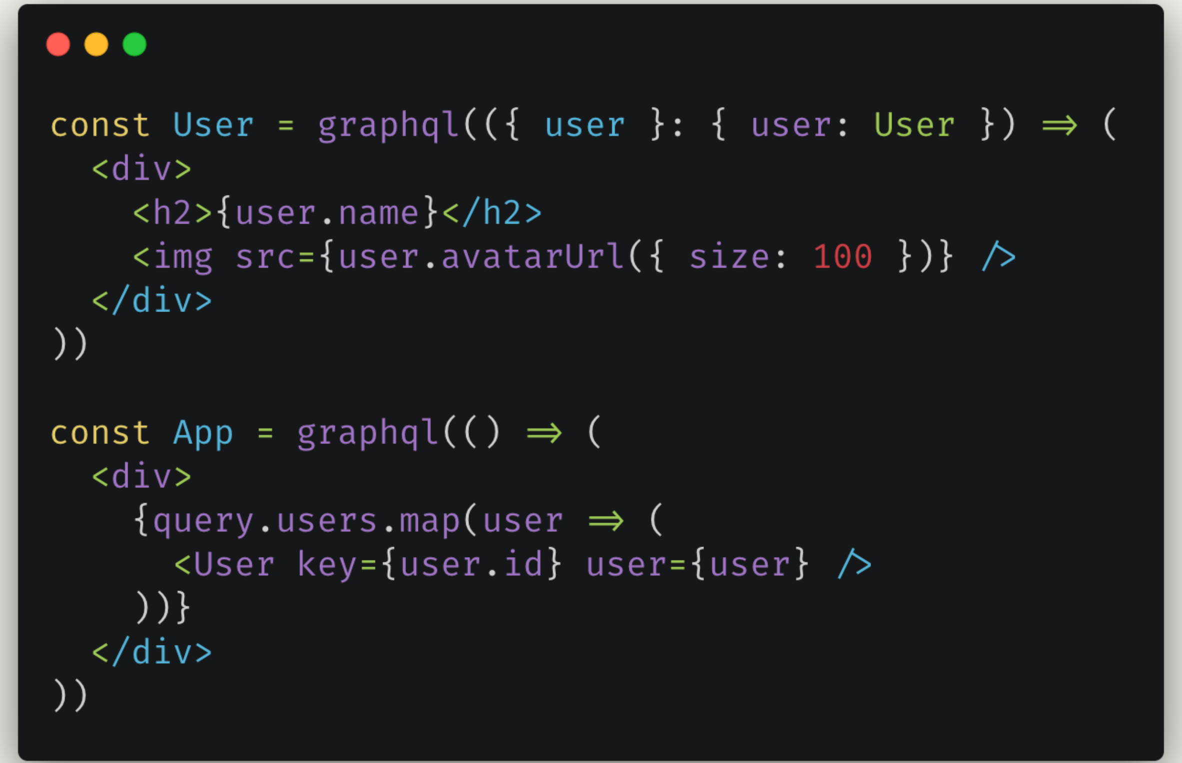


```
const User = graphql(({ user }: { user: User }) => (
  <div>
    <h2>{user.name}</h2>
    <img src={user.avatarUrl({ size: 100 })} />
  </div>
))

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

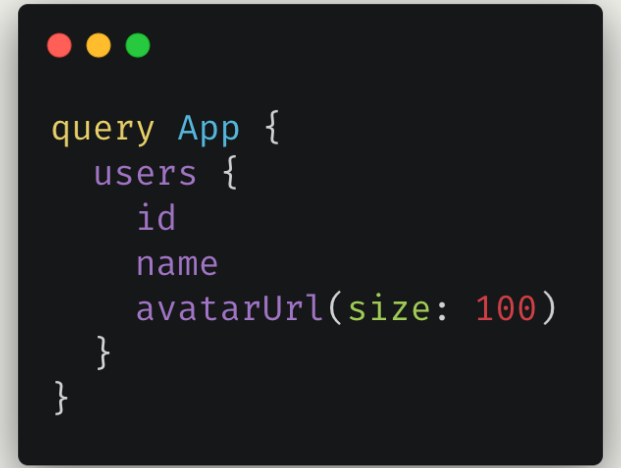
https://github.com/samdenty/gqless

GraphQL generation powers

GQLess: GraphQL without queries



```
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>
))
```



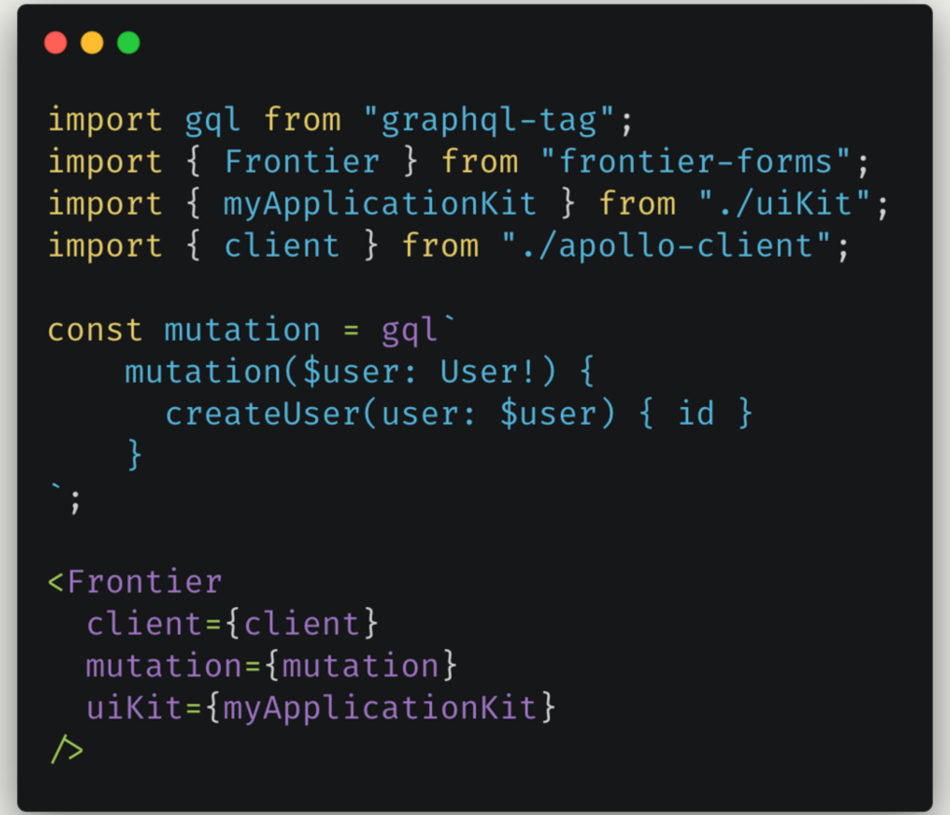
```
query App {
  users {
    id
    name
    avatarUrl(size: 100)
  }
}
```

GraphQL generation powers

React forms from GraphQL mutation

GraphQL generation powers

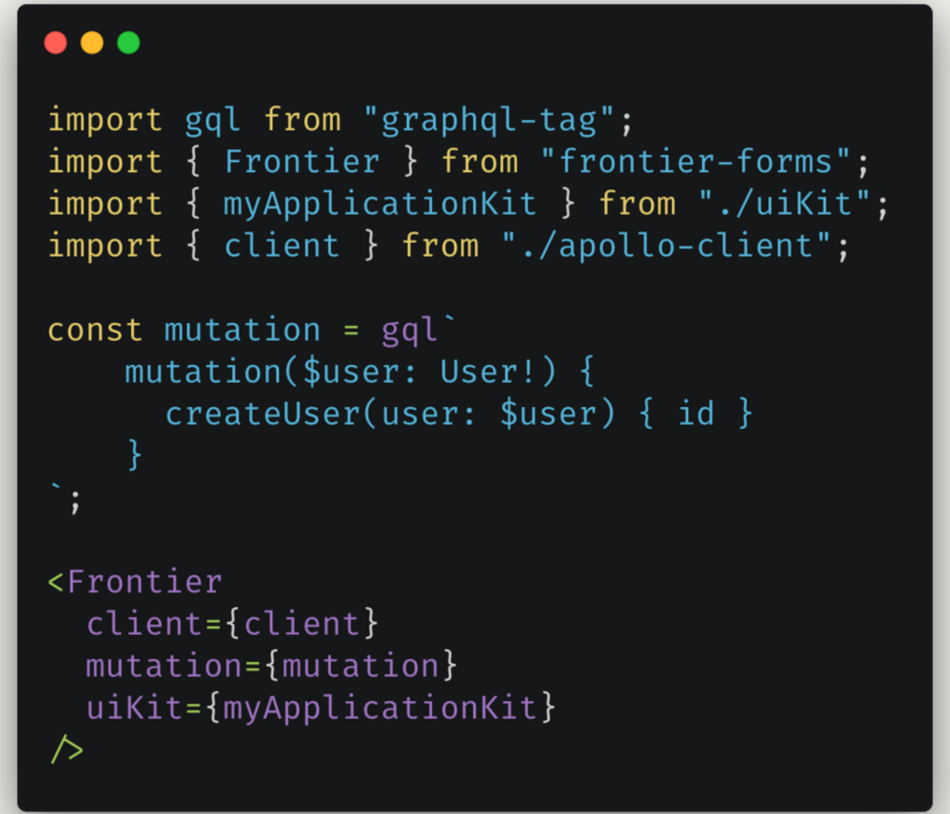
Frontier: Forms from GraphQL mutation


```
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}
/>
```

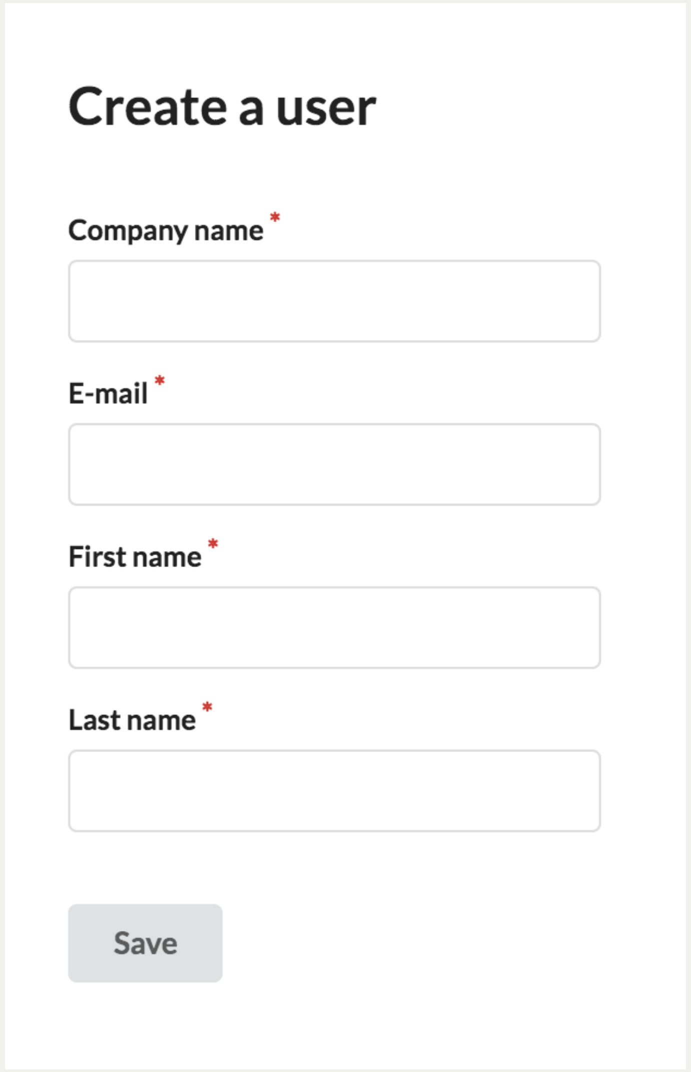
https://frontier-forms.dev

GraphQL generation powers

Frontier: Forms from GraphQL mutation



```
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}
/>
```



Create a user

Company name *

E-mail *

First name *

Last name *

GraphQL generation powers

GraphQL special power: introspection

GraphQL generation powers

GraphQL special power: introspection

- **Stronger types**

GraphQL generation powers

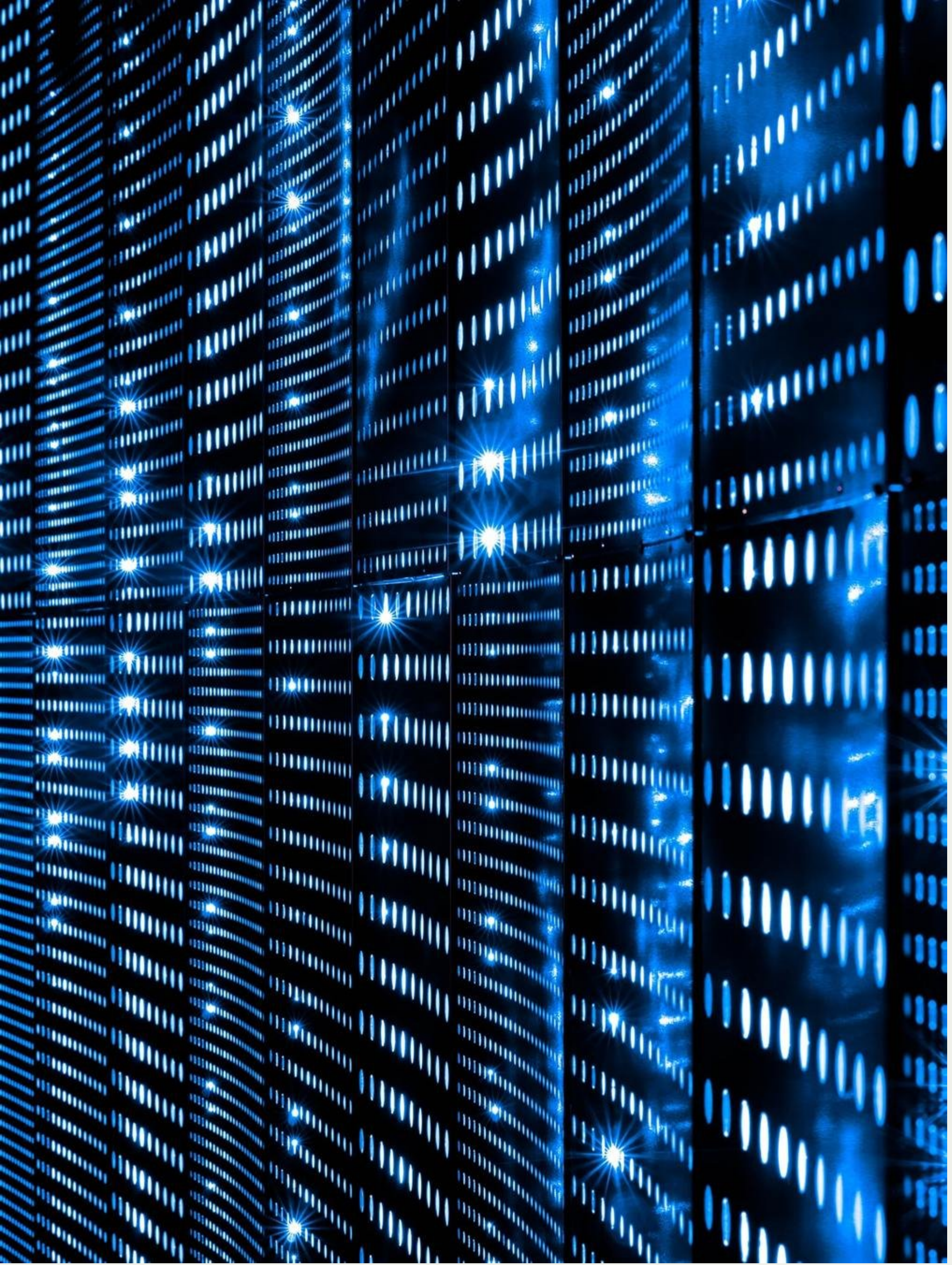
GraphQL special power: introspection

- **Stronger types**
- **Bootstrapping of client configuration**

GraphQL generation powers

GraphQL special power: introspection

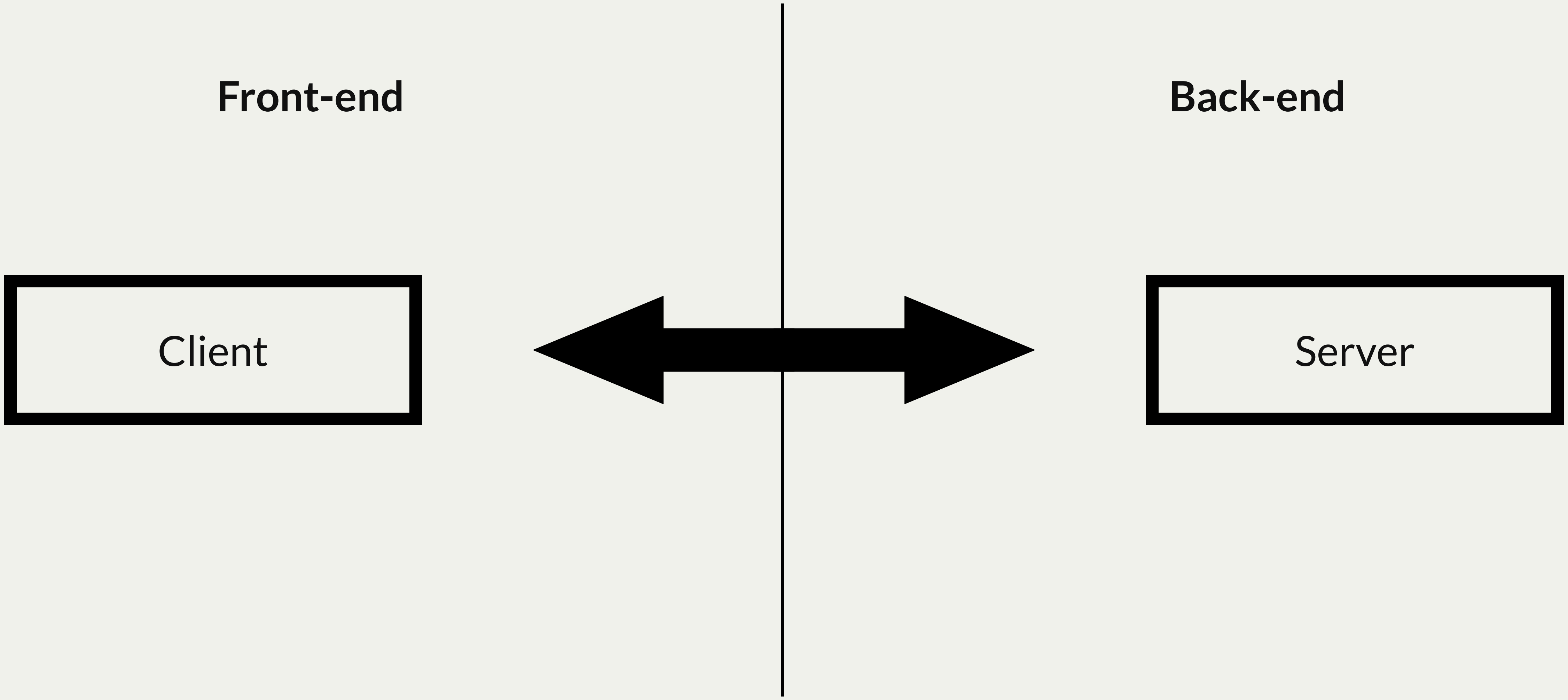
- **Stronger types**
- **Bootstrapping of client configuration**
- **Better developer experience via documents**



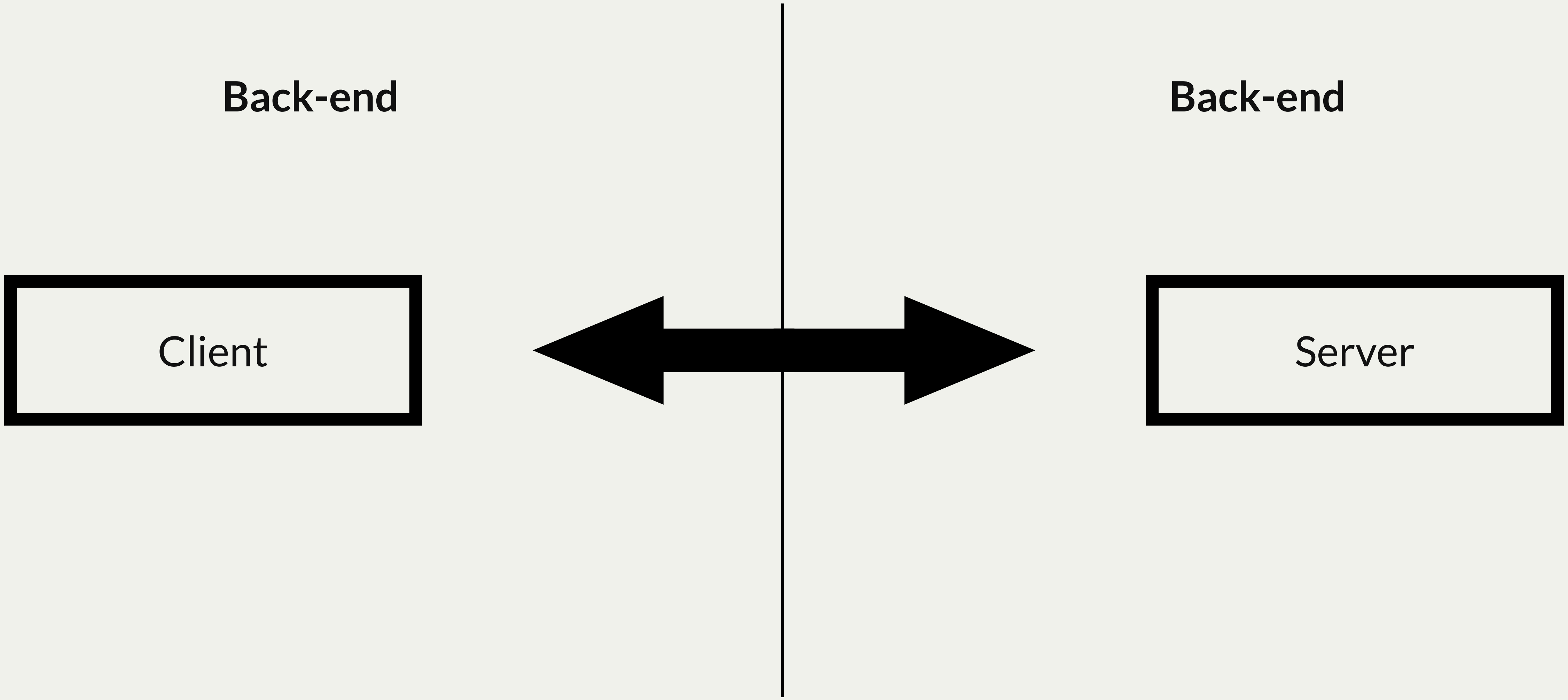
GraphQL for back-end to back-end

photo by Krzysztof Kowalik

GraphQL for back-end to back-end



GraphQL for back-end to back-end



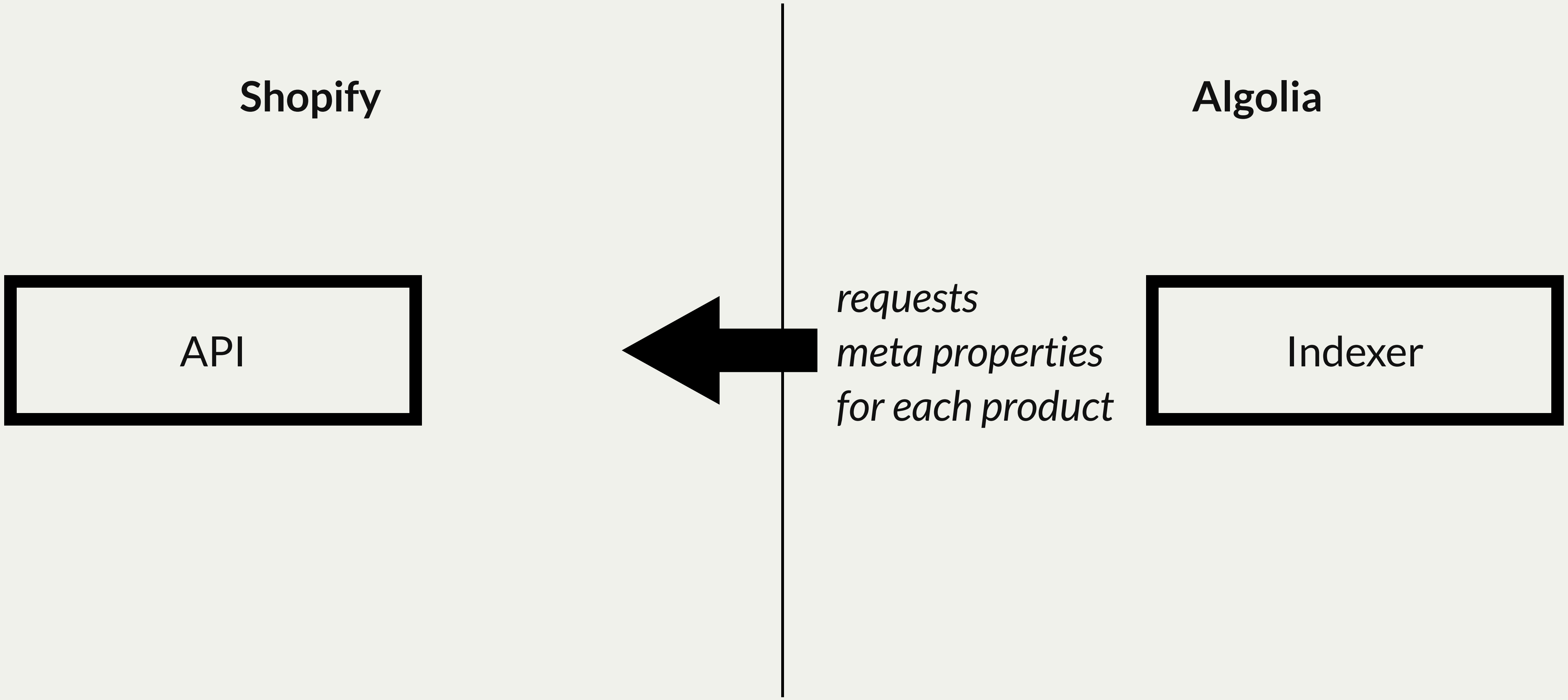
GraphQL for back-end to back-end

More flexible rate limiting

A story

GraphQL for back-end to back-end

Story: more flexible rate-limiting



GraphQL for back-end to back-end

Story: more flexible rate-limiting

Shopify REST API rate limiting

- per shop
- rate limit = 2 requests / seconds
  (Request-based limit)

GraphQL for back-end to back-end

Story: more flexible rate-limiting

Shopify REST API rate limiting

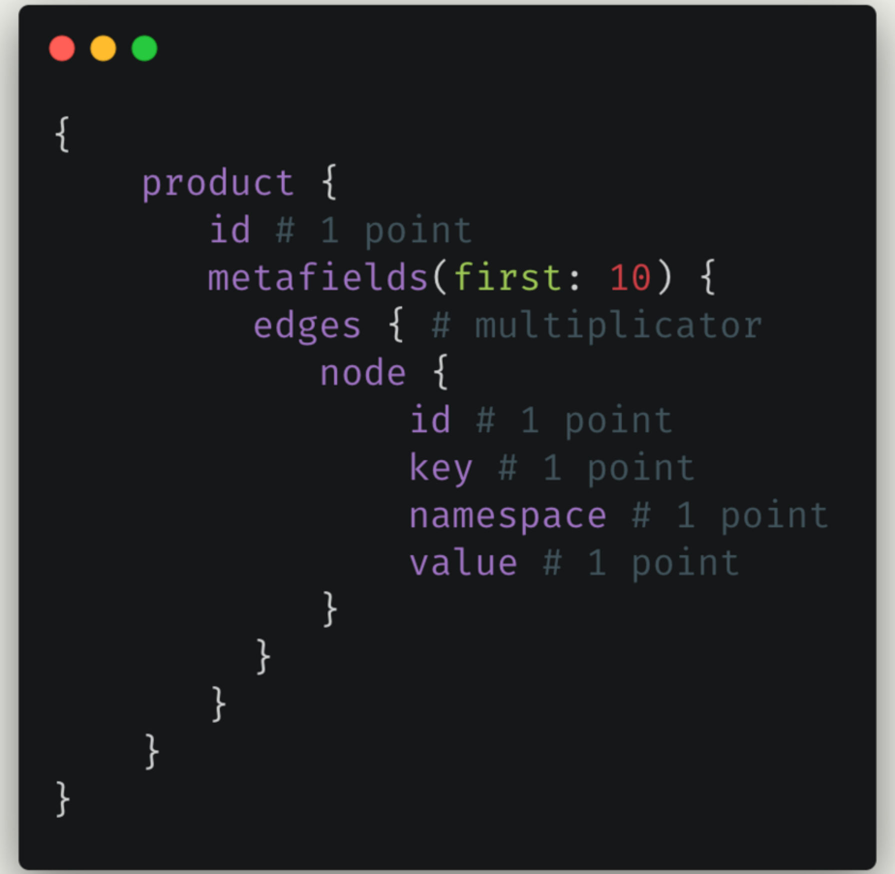
- per shop
- rate limit = 2 requests / seconds (Request-based limit)

Shopify GraphQL API rate limiting

- per shop
- 1 field = 1 point (Calculated query cost)
- rate limit = 50 points / seconds

GraphQL for back-end to back-end

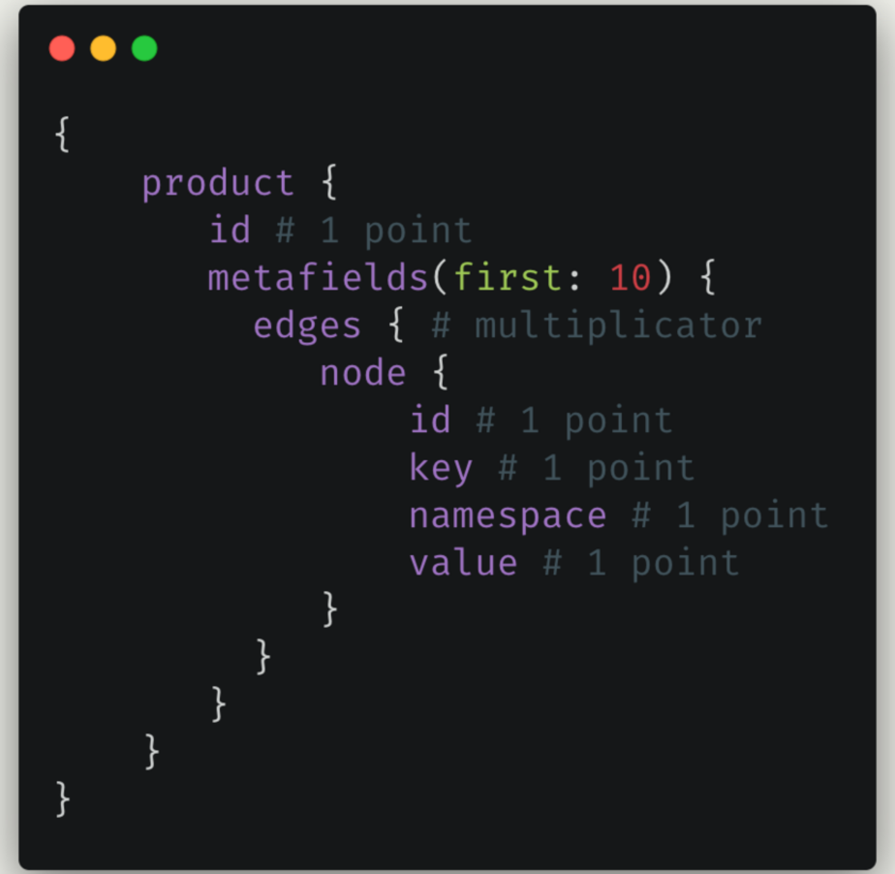
Story: more flexible rate-limiting



```
{
    product {
       id # 1 point
       metafields(first: 10) {
         edges { # multiplicator
            node {
                id # 1 point
                key # 1 point
                namespace # 1 point
                value # 1 point
            }
         }
       }
    }
}
```

GraphQL for back-end to back-end

Story: more flexible rate-limiting



```
{
    product {
      id # 1 point
      metafields(first: 10) {
        edges { # multiplicator
          node {
              id # 1 point
              key # 1 point
              namespace # 1 point
              value # 1 point
          }
        }
      }
    }
}
```

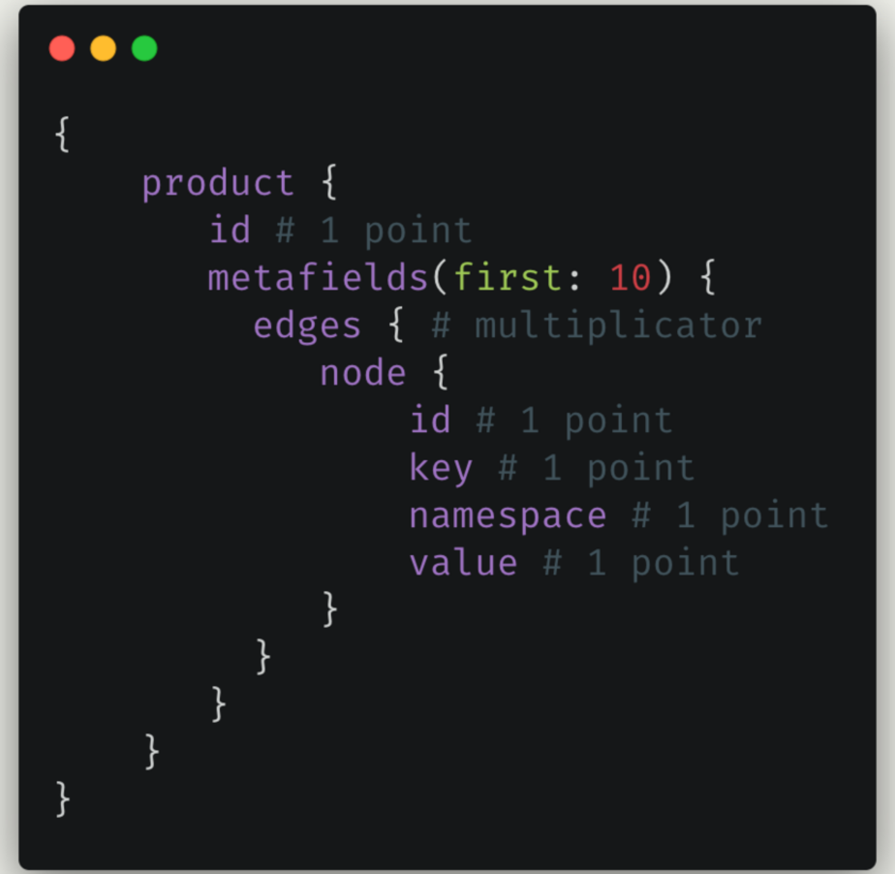
Cost of the query =

1 +
  10 x (
      1 + 1 + 1 +1
)

= 41 points

GraphQL for back-end to back-end

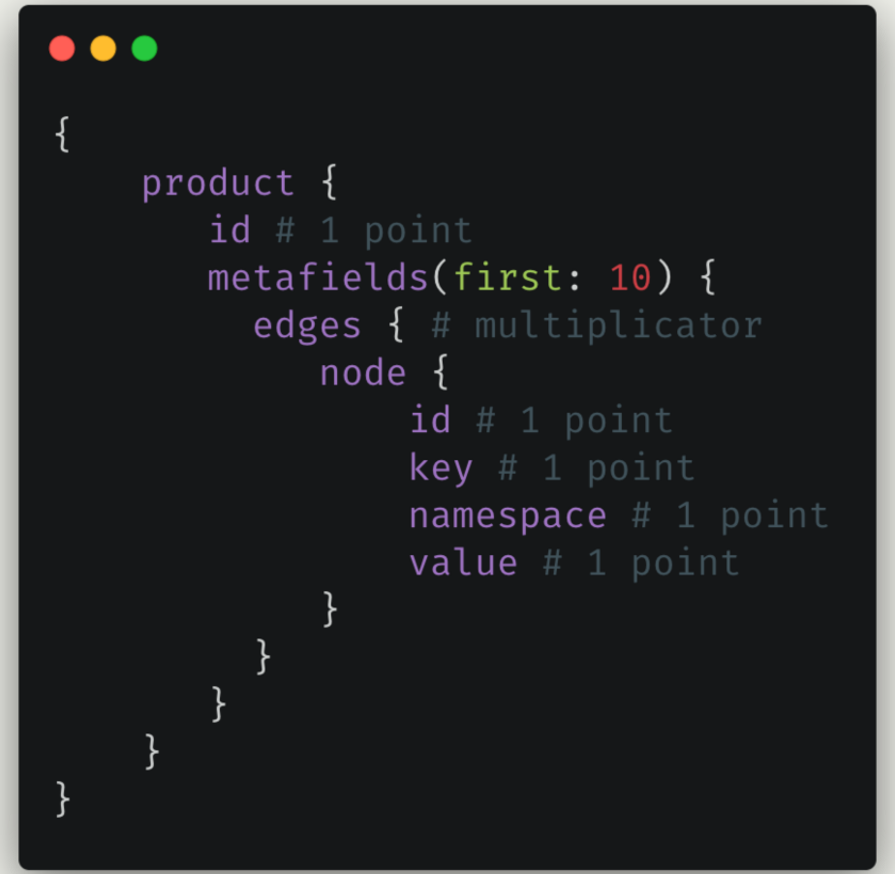
Story: more flexible rate-limiting



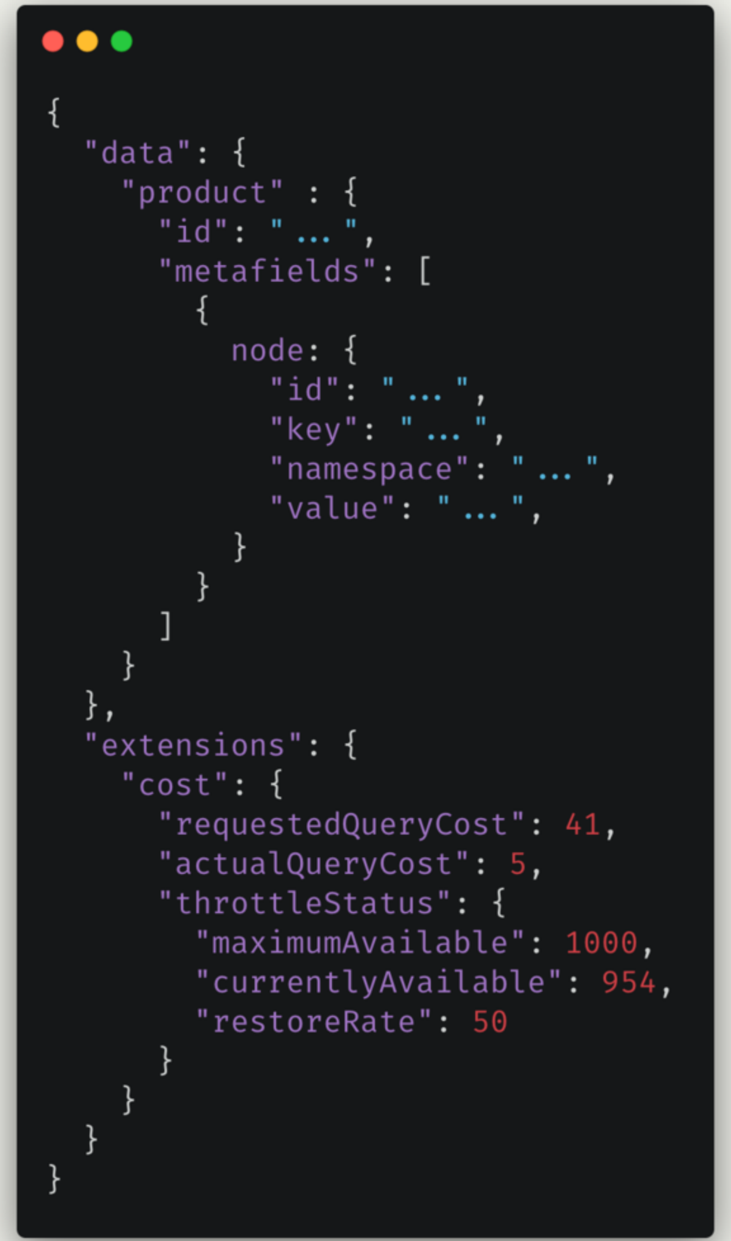
```
{
    product {
      id # 1 point
      metafields(first: 10) {
        edges { # multiplicator
          node {
              id # 1 point
              key # 1 point
              namespace # 1 point
              value # 1 point
          }
        }
      }
    }
}
```

GraphQL for back-end to back-end

Story: more flexible rate-limiting



```
{
    product {
       id # 1 point
       metafields(first: 10) {
         edges { # multiplicator
            node {
                id # 1 point
                key # 1 point
                namespace # 1 point
                value # 1 point
            }
         }
       }
    }
}
```



```
{
  "data": {
    "product" : {
      "id": " ... ",
      "metafields": [
        {
          node: {
            "id": " ... ",
            "key": " ... ",
            "namespace": " ... ",
            "value": " ... ",
          }
        }
      ]
    }
  },
  "extensions": {
    "cost": {
      "requestedQueryCost": 41,
      "actualQueryCost": 5,
      "throttleStatus": {
        "maximumAvailable": 1000,
        "currentlyAvailable": 954,
        "restoreRate": 50
      }
    }
  }
}
```

GraphQL for back-end to back-end

Story: more flexible rate-limiting

"Flexible throttling" indexing system using GraphQL

GraphQL for back-end to back-end

Story: more flexible rate-limiting

"Flexible throttling" indexing system using GraphQL

- Each customer (shop) has a points score assigned

GraphQL for back-end to back-end

Story: more flexible rate-limiting

"Flexible throttling" indexing system using GraphQL

- Each customer (shop) has a points score assigned
- Given the customer score, we compute if a query can be performed

GraphQL for back-end to back-end

Story: more flexible rate-limiting

"Flexible throttling" indexing system using GraphQL

- Each customer (shop) has a points score assigned
- Given the customer score, we compute if a query can be performed
- After each query, we update the shop score

GraphQL for back-end to back-end

Story: more flexible rate-limiting

"Flexible throttling" indexing system using GraphQL

- Each customer (shop) has a points score assigned
- Given the customer score, we compute if a query can be performed
- After each query, we update the shop score

Went from 2-4 products per second

GraphQL for back-end to back-end

Story: more flexible rate-limiting

"Flexible throttling" indexing system using GraphQL

- Each customer (shop) has a points score assigned
- Given the customer score, we compute if a query can be performed
- After each query, we update the shop score

Went from 2-4 products per second ➡ to 10-50 products per second

GraphQL for back-end to back-end

Companies providing GraphQL API with
”Calculated query cost” rate limiting

shopify GitHub yelp

GraphQL for back-end to back-end

Take-aways

GraphQL for back-end to back-end

Take-aways

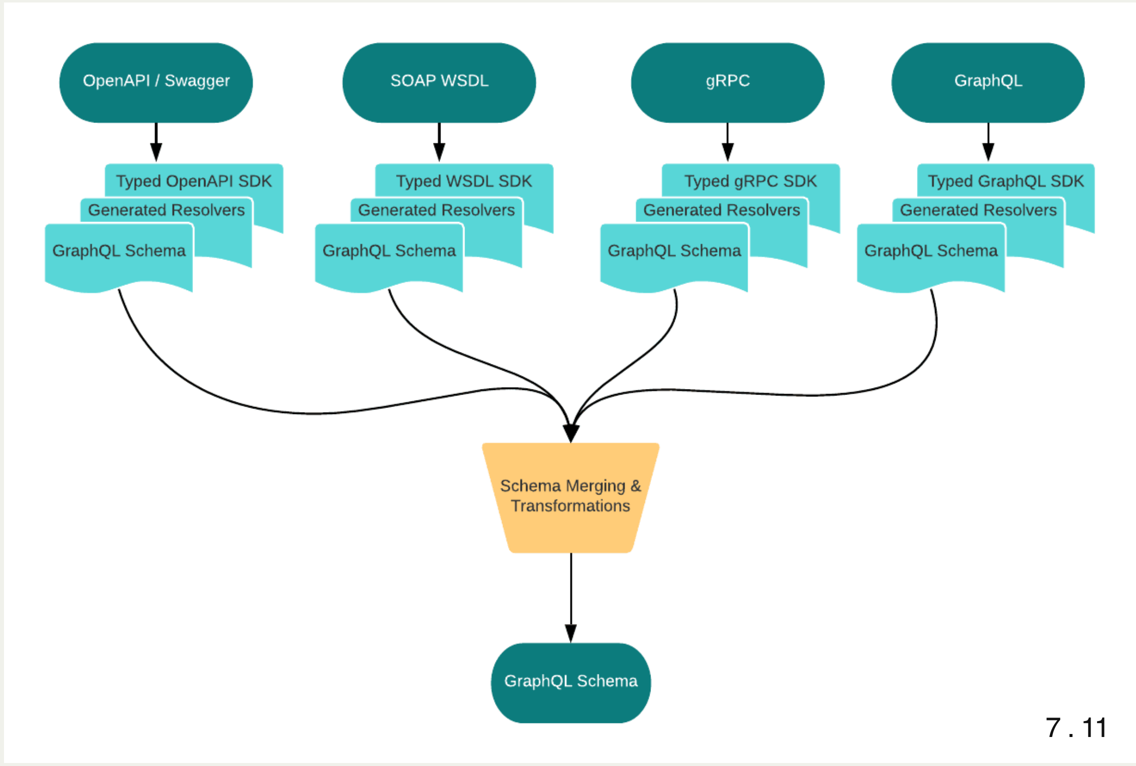
- **New API rate limiting offering**
  More granular throttling

GraphQL for back-end to back-end

Take-aways

- **New API rate limiting offering**
  More granular throttling

- **Maintainable data pipelines**
  Abstract complex APIs, ex: GraphQL Mesh



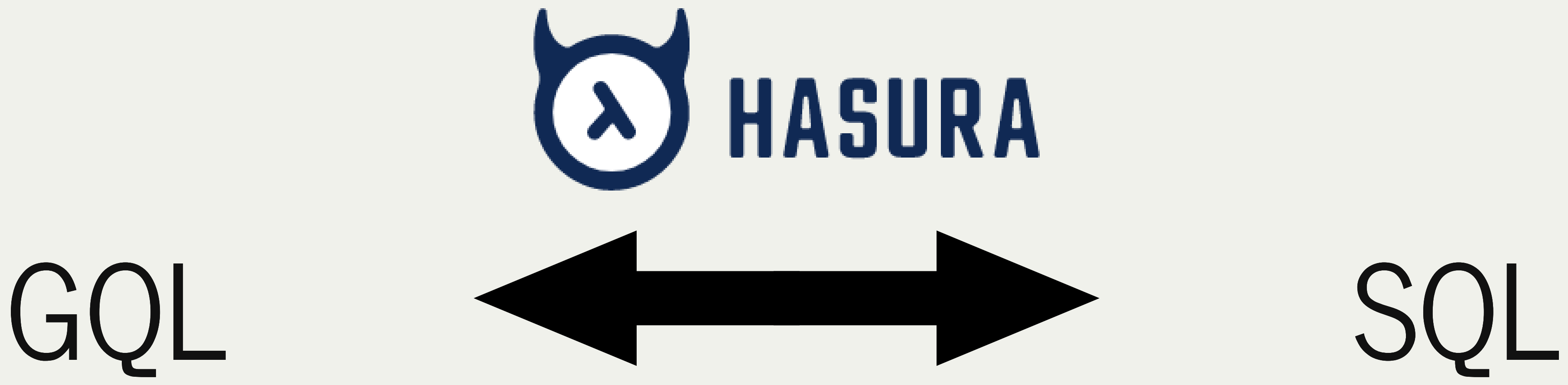
”resolvers-less” GraphQL

photo by Matias Caceres

”resolvers-less” GraphQL

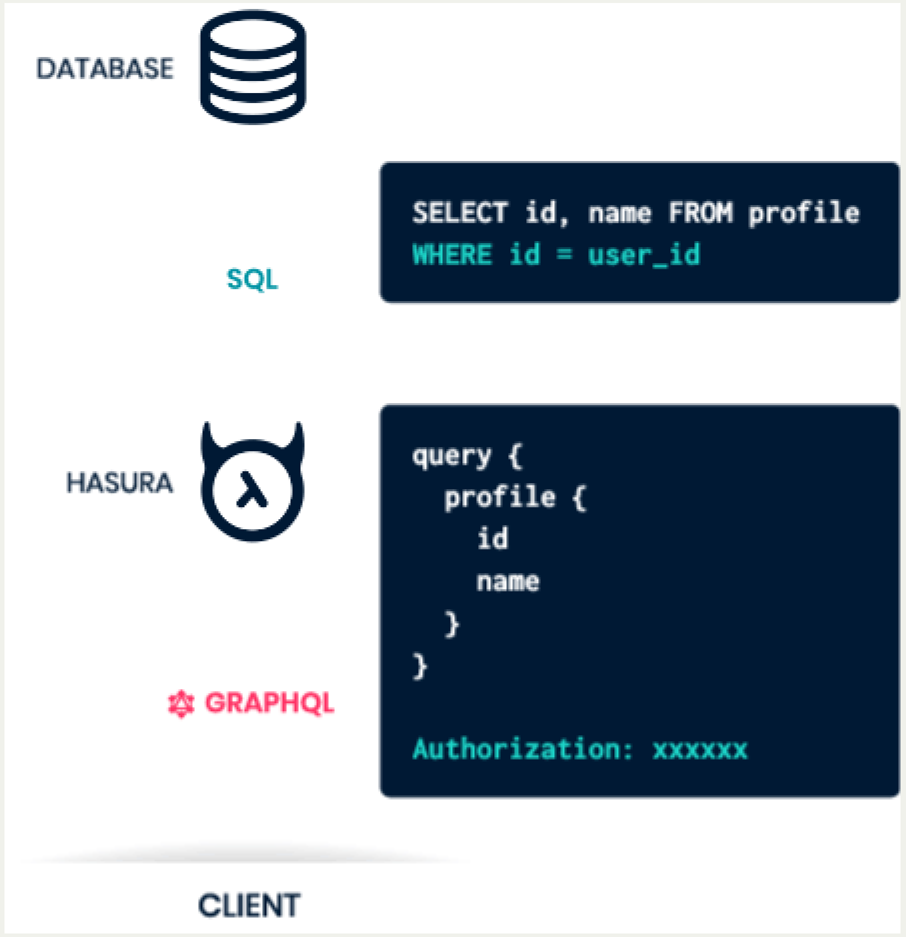
$$\begin{tikzcd} \text{GQL} \arrow[rr, leftrightarrow, "?"] && \text{SQL} \end{tikzcd}$$

”resolvers-less” GraphQL



”resolvers-less” GraphQL

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



```
SELECT id, name FROM profile
WHERE id = user_id
```

```
query {
  profile {
    id
    name
  }
}

Authorization: xxxxxx
```

photo by jesse orrico

”resolvers-less” GraphQL

Hasura features

”resolvers-less” GraphQL

Hasura features

- ACL support

”resolvers-less” GraphQL

Hasura features

- ACL support
- Authentication / Authorization (JWT)

”resolvers-less” GraphQL

Hasura features

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

”resolvers-less” GraphQL

Hasura features

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

”resolvers-less” GraphQL

Hasura features

- ACL support
- Authentication / Authorization (JWT)
- Remote schemas support
- Subscriptions support
- Trigger web-hooks on database events

”resolvers-less” GraphQL

Hasura features

- ACL support
- Authentication / Authorization (JWT)
- Remote schemas support
- Subscriptions support
- Trigger web-hooks on database events
- Bonus: one-click install on most cloud providers

Conclusion

GraphQL brings innovation beyond ”front-end querying APIs”:

Conclusion

GraphQL brings innovation beyond ”front-end querying APIs”:

- Apollo GraphQL is reliable for local state management

Conclusion

GraphQL brings innovation beyond ”front-end querying APIs”:

- Apollo GraphQL is reliable for local state management
- GraphQL brings flexibility in back-end to back-end use-cases

Conclusion

GraphQL brings innovation beyond ”front-end querying APIs”:

- Apollo GraphQL is reliable for local state management
- GraphQL brings flexibility in back-end to back-end use-cases
- GraphQL introspection finally brought great tools on front-end

Conclusion

GraphQL brings innovation beyond ”front-end querying APIs”:

- Apollo GraphQL is reliable for local state management
- GraphQL brings flexibility in back-end to back-end use-cases
- GraphQL introspection finally brought great tools on front-end
- GraphQL to SQL brings powerful server-less GraphQL use-cases

Thank you!

slides on noti.st/charlypoly

@whereischarly

@wittydeveloper