Using GraphQL APIs with Apollo Client

Table of Contents

- **Using Apollo Client with JavaScript**
- Using Apollo Client with React
- Managing local app state
- Implementing and using GraphQL subscriptions

Using Apollo Client with JavaScript

- The first step to work with Apollo Client is to add it to the project dependencies.
- It's hosted under the npm package @apollo/client.

```
$ npm install @apollo/client
```

Making a query request

```
import {
  ApolloClient,
  HttpLink,
  InMemoryCache,
  gql,
} from '@apollo/client';

import * as config from './config';

const cache = new InMemoryCache();
const httpLink = new HttpLink({ uri: config.GRAPHQL_SERVER_URL });
const client = new ApolloClient({ cache, link: httpLink });

async function main() {
  const { data, errors } = await client.query({
    query: gql`
      query {
        numbersInRange(begin: 1, end: 100) {
          sum
        }
      }
    `,
  });

  console.log({ data, errors });
}

main();
```

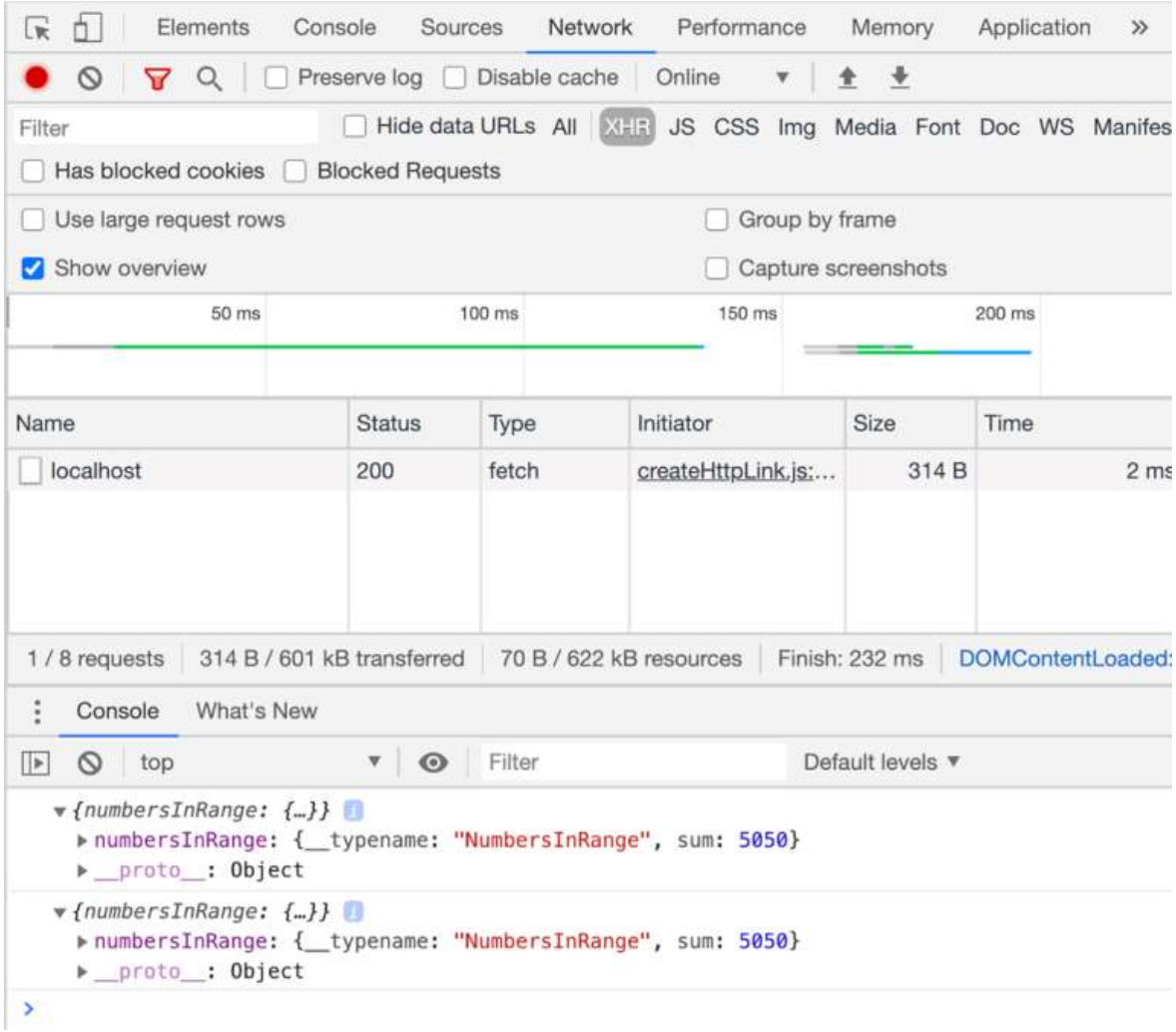
Using Apollo Client with JavaScript

```
top ▼   Filter   Default levels ▼
▼ {data: {…}, errors: undefined}
  ▼ data:
    ▶ numbersInRange: {__typename: "NumbersInRange", sum: 5050}
    ▶ __proto__: Object
    errors: undefined
  ▶ __proto__: Object
>
```

Making a query request

```
async function main() {
  const resp1 = await client.query({
    query: gql`
      {
        numbersInRange(begin: 1, end: 100) {
          sum
        }
      }
    `,
  });
  console.log(resp1.data);

  const resp2 = await client.query({
    query: gql`
      {
        numbersInRange(begin: 1, end: 100) {
          sum
        }
      }
    `,
  });
  console.log(resp2.data);
}
```

Making a query request

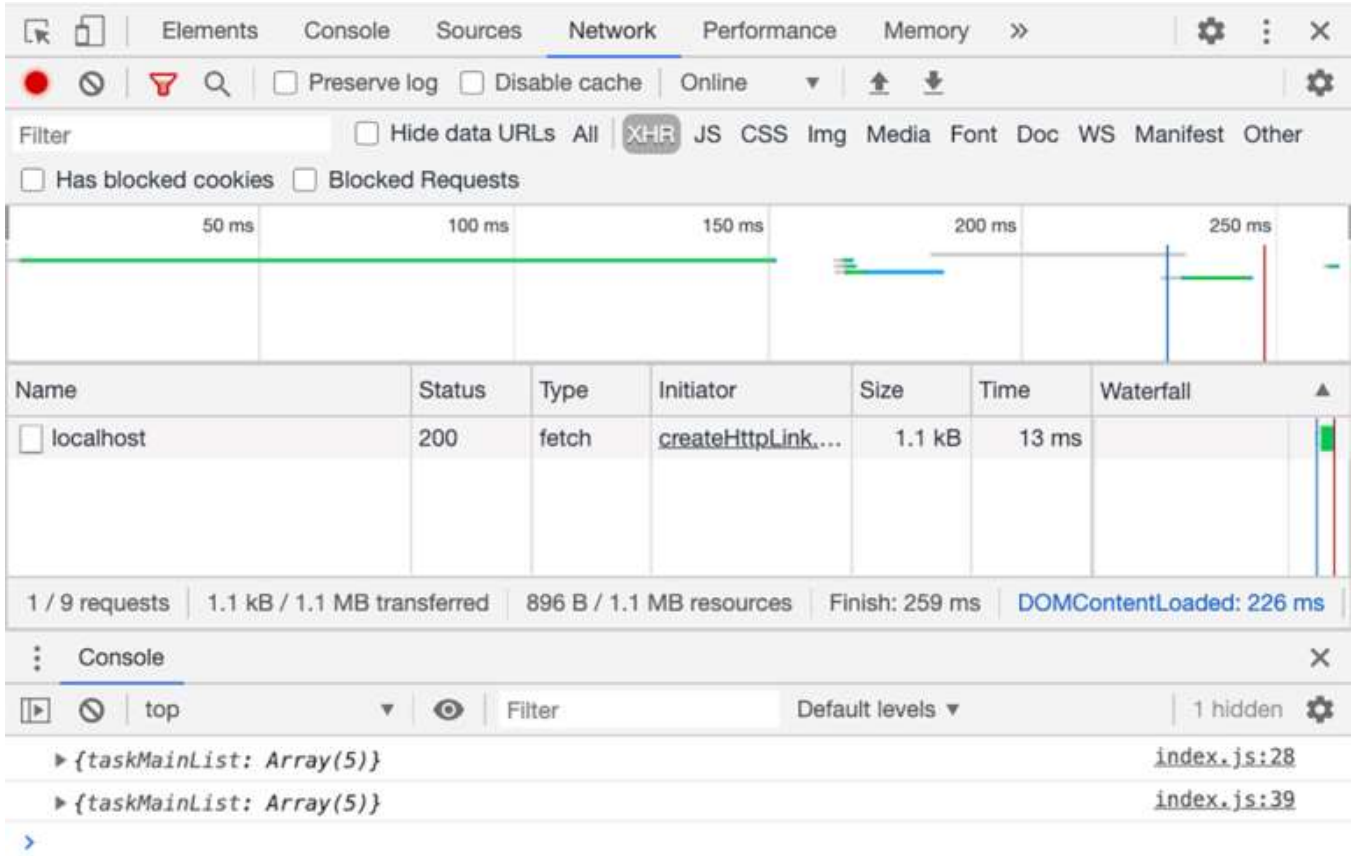


```
async function main() {
  const resp1 = await client.query({
    query: gql`
      {
        taskMainList {
          id
          content
          tags
          createdAt
        }
      }
    `,
  });
  console.log(resp1.data);

  const resp2 = await client.query({
    query: gql`
      {
        taskMainList {
          content
        }
      }
    `,
  });
  console.log(resp2.data);
}
```

APOLLO

Making a query request

Making a query request



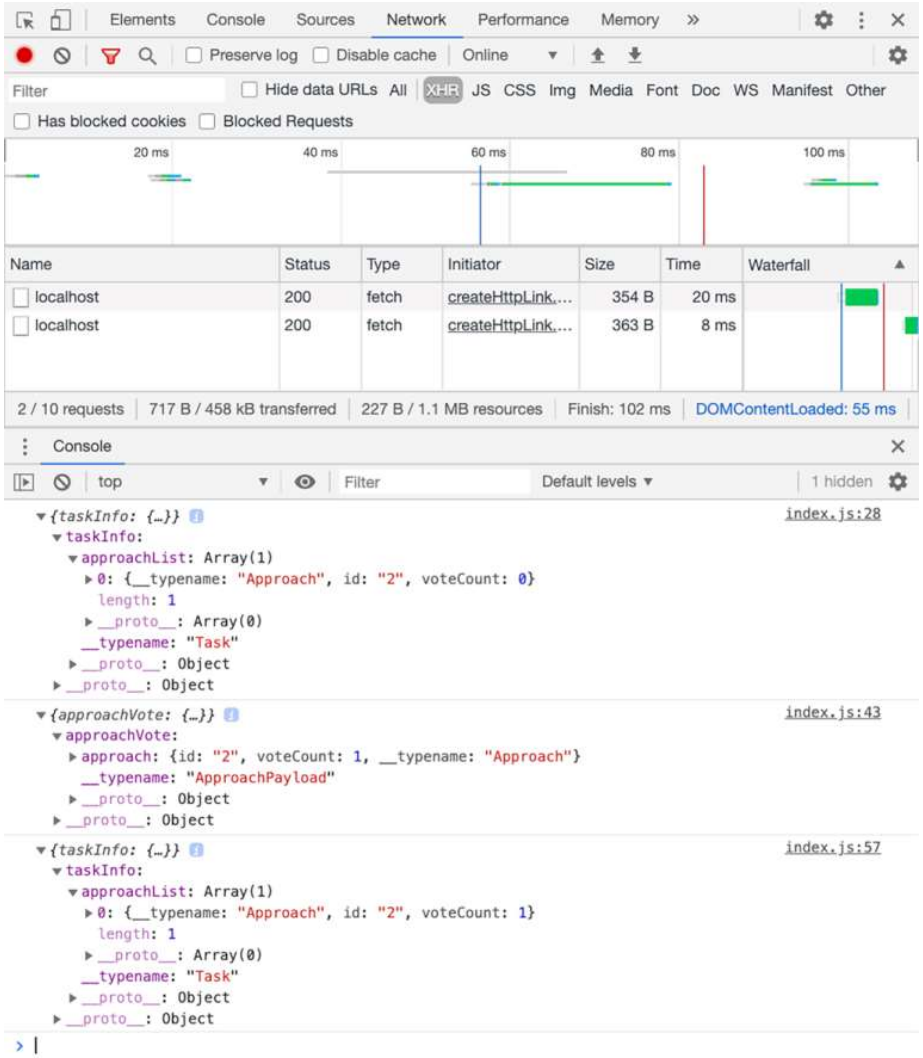
Making a mutation request

```
async function main() {
  const resp1 = await client.query({
    query: gql`
      query taskInfo {
        taskInfo(id: "2") {
          approachList {
            id
            voteCount
          }
        }
      }
    `,
  });
  console.log(resp1.data);

  const resp2 = await client.mutate({
    mutation: gql`
      mutation approachVote($approachId: ID!) {
        approachVote(approachId: $approachId, input: { up: true })
          approach {
            id
            voteCount
          }
        }
      }
    `,
```

Continued code

```
  variables: { approachId: '2' },
});
console.log(resp2.data);

const resp3 = await client.query({
  query: gql`
    query taskInfo {
      taskInfo(id: "2") {
        approachList {
          id
          voteCount
        }
      }
    }
  `,
});
console.log(resp3.data);
}
```

Making a mutation request



```
import 'regenerator-runtime/runtime';
import React from 'react';
import ReactDOM from 'react-dom';

import { useStoreObject, Provider } from './store';
import Root from './components/Root';

export default function App() {
  const store = useStoreObject();
  return (
    <Provider value={store}>
      <Root />
    </Provider>
  );
}

ReactDOM.render(<App />, document.getElementById('root'));
```

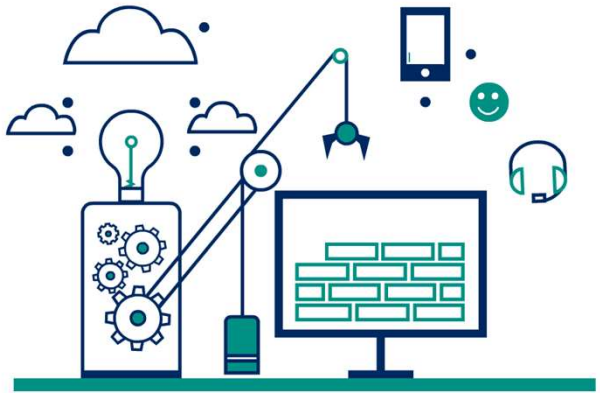
Making a mutation request

Table of Contents

- Using Apollo Client with JavaScript
- **Using Apollo Client with React**
- Managing local app state
- Implementing and using GraphQL subscriptions

Using Apollo Client with React

```
import React, { useState } from 'react';
import fetch from 'cross-fetch';

import * as config from './config';
import {
  ApolloClient,
  HttpLink,
  InMemoryCache,
} from '@apollo/client';

const httpLink = new HttpLink({ uri: config.GRAPHQL_SERVER_URL
});
const cache = new InMemoryCache();
const client = new ApolloClient({ link: httpLink, cache });
// ·-·-·
```

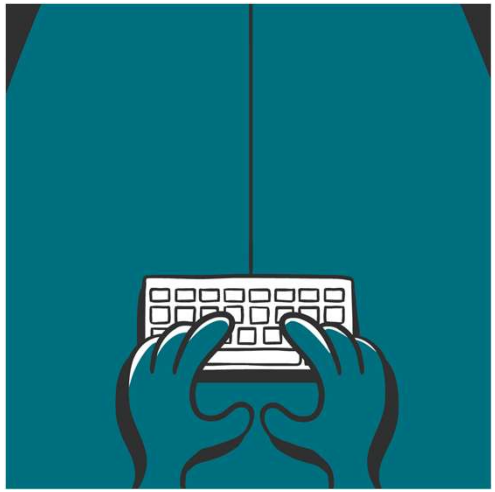


```
// ...

export const useStoreObject = () => {
  // ...

  const query = async (query, { variables } = {}) => {
    const resp = await client.query({ query, variables });
    return resp;
  };

  const mutate = async (mutation, { variables } = {}) => {
    const resp = await client.mutate({ mutation, variables });
    return resp;
  };

  return {
    useLocalAppState,
    setLocalAppState,
    AppLink,
    query,
    mutate,
  };
};
```

Using the query and mutate methods directly

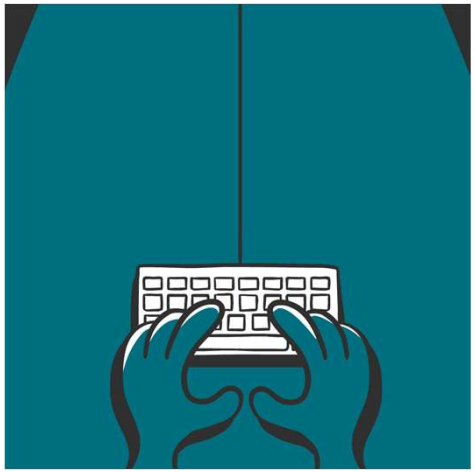
Using the query and mutate methods directly

```
// ....
import { gql } from '@apollo/client';

const TASK_MAIN_LIST = gql`
  query taskList {
    taskList {
      id
      ...TaskSummary
    }
  }

  ${TASK_SUMMARY_FRAGMENT}
`;

export default function Home() {
  const { query } = useStore();
  const [ taskList, setTaskList ] = useState(null);

  useEffect(() => {
    query(TASK_MAIN_LIST).then(({ data }) => {
      setTaskList(data.taskList);
    });
  }, [query]);

  if (!taskList) {
    return <div className="loading">Loading...</div>;
  }

  // ....
}
```



Using the query and mutate methods directly

```
// ....
import { gql } from '@apollo/client';

const USER_LOGIN = gql`
  mutation userLogin($input: AuthInput!) {
    userLogin(input: $input) {
      errors {
        message
        field
      }
      user {
        id
        name
      }
      authToken
    }
  }
`;

export default function Login() {
  const { mutate, setLocalAppState } = useStore();
  const [ uiErrors, setUIErrors ] = useState();
  const handleLogin = async (event) => {
    event.preventDefault();
    const input = event.target.elements;
    const { data, errors: rootErrors } = await mutate(USER_LOGIN, {
      variables: {
        input: {
          username: input.username.value,
          password: input.password.value,
        },
      },
    });
    // ....
  };
```



Including authentication headers

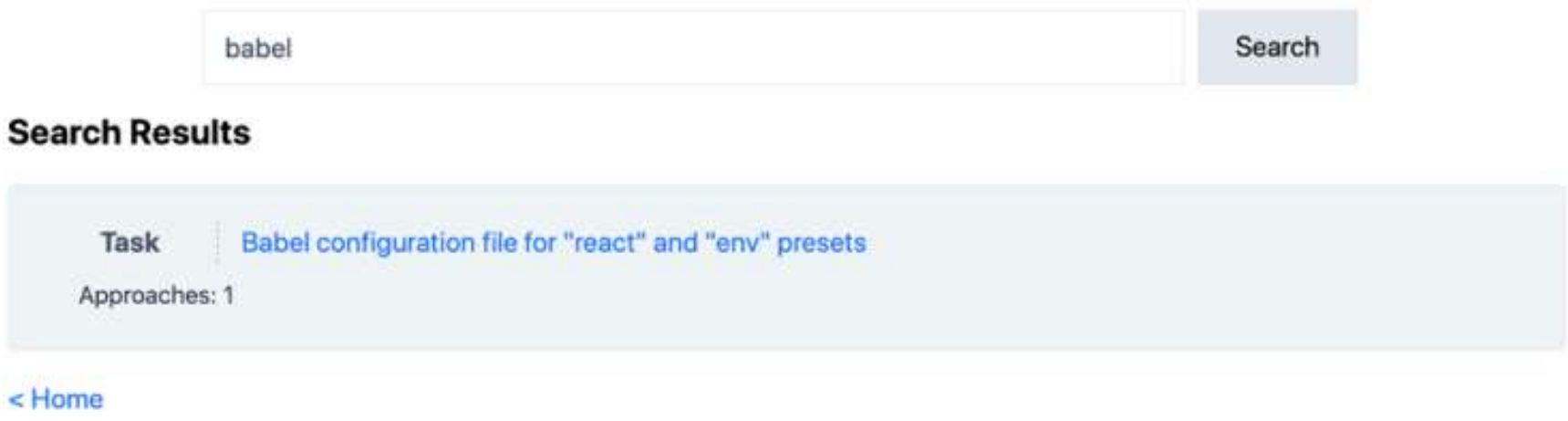
- To make the current user's authToken value part of the link chain context, start by installing the new package.

```
$ npm install @apollo/link-context
```

Including authentication headers

```
// ....
import { setContext } from '@apollo/link-context';
// ....

export const useStoreObject = () => {
  // ....

  const AppLink = ({ children, to, ...props }) => {
    // ....
  };

  const authLink = setContext((_, { headers }) => {
    return {
      headers: {
        ...headers,
        authorization: state.user
          ? `Bearer ${state.user.authToken}`
          : '',
      },
    };
  });

  client.setLink(authLink.concat(httpLink));

  // ....
};
```

Including authentication headers



Including authentication headers

```
const setLocalAppState = (newState) => {
  if (newState.component) {
    newState.component.props = newState.component.props
?? {};
  }
  setState((currentState) => {
    return { ...currentState, ...newState };
  });
  // Reset cache when users login/logout
  if (newState.user || newState.user === null) {
    client.resetStore();
  }
};
```

```
// ·-·-·
export const useStoreObject = () => {
  // ·-·-·

  const authLink = setContext((_, { headers }) => {
    // ·-·-·
  });

  client.setLink(authLink.concat(httpLink));

  // Remove query/mutate methods

  return {
    useLocalAppState,
    setLocalAppState,
    AppLink,
    client,
  };
};
```

Using Apollo hook functions

```
// ...
import { ApolloProvider } from '@apollo/client';

import { useStoreObject, Provider as StoreProvider } from './store'
import Root from './components/Root';

export default function App() {
  const store = useStoreObject();
  return (
    <ApolloProvider client={store.client}>
      <StoreProvider value={store}>
        <Root />
      </StoreProvider>
    </ApolloProvider>
  );
}

ReactDOM.render(<App />, document.getElementById('root'));
```

Using Apollo hook functions

Using Apollo hook functions

```
import React from 'react';
import { gql, useQuery } from '@apollo/client';

import Search from './Search';
import TaskSummary, { TASK_SUMMARY_FRAGMENT } from './TaskSummary';
// ...

export default function Home() {
  const { loading, data } = useQuery(TASK_MAIN_LIST);

  if (loading) {
    return <div className="loading">Loading...</div>;
  }

  return (
    <div>
      <Search />
      <div>
        <h1>Latest</h1>
        {data.taskMainList.map((task) => (
          <TaskSummary key={task.id} task={task} link={true} />
        ))}
      </div>
    </div>
  );
}
```

```diff
+++ b/web/src/components/Home.js
@@ -1,7 +1,6 @@
-import React, { useState, useEffect } from 'react';
-import { gql } from '@apollo/client';
+import React from 'react';
+import { gql, useQuery } from '@apollo/client';

-import { useStore } from '../store';
 import Search from './Search';
 import TaskSummary, { TASK_SUMMARY_FRAGMENT } from './TaskSummary';

@@ -17,16 +16,9 @@ const TASK_MAIN_LIST = gql`
 `;

 export default function Home() {
-  const { query } = useStore();
-  const [ taskList, setTaskList ] = useState(null);
+  const { loading, data } = useQuery(TASK_MAIN_LIST);

-  useEffect(() => {
-    query(TASK_MAIN_LIST).then(({ data }) => {
-      setTaskList(data.taskMainList);
-    });
-  }, [query]);
-
-  if (!taskList) {
+  if (loading) {
     return <div className="loading">Loading...</div>;
   }

@@ -35,7 +27,7 @@ export default function Home() {
       <Search />
       <div>
         <h1>Latest</h1>
-        {taskList.map((task) => (
+        {data.taskMainList.map((task) => (
           <TaskSummary key={task.id} task={task} link={true} />
         ))}
```

Using Apollo hook functions

Using Apollo hook functions

```
export default function Home() {
  const { error, loading, data } = useQuery(TASK_MAIN_LIST);

  if (error) {
    return <div className="error">{error.message}</div>
  }

  if (loading) {
    return <div className="loading">Loading...</div>;
  }

  // ·-·-·
}
```

Using Apollo hook functions

- Here's an example of how we can use both items in the returned tuple.

```
const [ loginUser, { error, loading, data } ] =
useMutation(USER_LOGIN);
```

Using Apollo hook functions

```
import React, { useState } from 'react';
import { gql, useMutation } from '@apollo/client';
// ...

export default function Login() {
  const { setLocalAppState } = useStore();
  const [ uiErrors, setUIErrors ] = useState();

  const [ loginUser, { error, loading } ] = useMutation(USER_LOGIN);

  if (error) {
    return <div className="error">{error.message}</div>;
  }

  const handleLogin = async (event) => {
    event.preventDefault();
    const input = event.target.elements;
    const { data, errors: rootErrors } = await loginUser({
      variables: {
        input: {
          username: input.username.value,
          password: input.password.value,
        },
      },
    });
    if (rootErrors) {
      return setUIErrors(rootErrors);
    }
    const { errors, user, authToken } = data.userLogin;
    if (errors.length > 0) {
      return setUIErrors(errors);
    }
    // ...
  };
// ...
```

```
import React, { useState } from 'react';
import { gql, useQuery } from '@apollo/client';
// ·-·-·

export default function TaskPage({ taskId }) {
  const { AppLink } = useStore();
  // const [ taskInfo, setTaskInfo ] = useState(null);
  const [ showAddApproach, setShowAddApproach ] = useState(false);
  const [ highlightedApproachId, setHighlightedApproachId ] = useState();

  const { error, loading, data } = useQuery(TASK_INFO, {
    variables: { taskId },
  });

  if (error) {
    return <div className="error">{error.message}</div>;
  }

  if (loading) {
    return <div className="loading">Loading...</div>;
  }

  const { taskInfo } = data;

   const handleAddNewApproach = (newApproach) => {
    // setTaskInfo((pTask) => ({
    // ...pTask,
    // approachList: [newApproach, ...pTask.approachList],
    // }));
     setHighlightedApproachId(newApproach.id);
     setShowAddApproach(false);
   };

  return (
    // ·-·-·
```

Using Apollo hook functions

Performing operations conditionally

```
import React from 'react';
import { gql, useQuery } from '@apollo/client';
// ...

export default function Search({ searchTerm = null }) {
  const { setLocalAppState, AppLink } = useStore();
  const { error, loading, data } = useQuery(SEARCH_RESULTS, {
    variables: { searchTerm },
  });

  if (error) {
    return <div className="error">{error.message}</div>;
  }

  const handleSearchSubmit = async (event) => {
    // ...
  };

  return (
    <div>
      {/* ... */}
      {data && data.searchResults && (
        <div>
          <h2>Search Results</h2>
          <div className="y-spaced">
            {data.searchResults.length === 0 && (
              <div className="box box-primary">No results</div>
            )}
            {data.searchResults.map((item, index) => (
              <div key={index} className="box box-primary">
                {/* ... */}
              </div>
            ))}
          </div>
          <AppLink to="Home">{'<'} Home</AppLink>
```

Performing operations conditionally

Response not successful: Received status code 500

Latest

Make an image in HTML change based on the theme color mode (dark or light)

test

Get rid of only the unstaged changes since the last git commit

Elements Console Sources Network Performance Mem

top Filter Default levels

POST http://localhost:4321/ 500 (Internal Server Error)

Performing operations conditionally

```
import React from 'react';
import { gql, useQuery } from '@apollo/client';
// ·-·-·
export default function Search({ searchTerm = null }) {
  const { setLocalAppState, AppLink } = useStore();
  const { error, loading, data } =
useQuery(SEARCH_RESULTS, {
    variables: { searchTerm },
    skip: !searchTerm,
  });

  // ·-·-·
 }
```

```
import React from 'react';
import { gql, useLazyQuery } from '@apollo/client';
// ·-·-·

export default function Search({ searchTerm = null }) {
  const { setLocalAppState, AppLink } = useStore();
  const [
    performSearch,
     { error, loading, data },
  ] = useLazyQuery(SEARCH_RESULTS, { variables: { searchTerm } });

  useEffect(() => {
    if (searchTerm) {
      performSearch();
    }
  }, [searchTerm, performSearch]);

  if (error) {
    return <div className="error">{error.message}</div>;
  }

  // ·-·-·
}
```

Table of Contents

- Using Apollo Client with JavaScript
- Using Apollo Client with React
- **Managing local app state**
- Implementing and using GraphQL subscriptions

Managing local app state

```
import {
  ApolloClient,
  HttpLink,
  InMemoryCache,
  gql,
} from '@apollo/client';
// ...
export const LOCAL_APP_STATE = gql`
  query localAppState {
    component @client {
      name
      props
    }
    user @client {
      username
      authToken
    }
  }
`;
```

Managing local app state

- Now we can use Apollo to read and update the query.
- For example, in places where we previously used state.user in the store, we can now read it from the cache.

```
const { user } = cache.readQuery({ query:
LOCAL_APP_STATE });
```

Managing local app state

- To update the user/component objects, instead of the current setState calls in the store, we can do the following.

```
cache.writeQuery({
  query: LOCAL_APP_STATE,
  data: { ...currentState, ...newState },
})
```

Managing local app state

```
import 'regenerator-runtime/runtime';
import React from 'react';
import ReactDOM from 'react-dom';
import {
  ApolloProvider,
  ApolloClient,
  HttpLink,
  InMemoryCache,
} from '@apollo/client';
import { setContext } from '@apollo/link-context';

import * as config from './config';
import Root from './components/Root';

const httpLink = new HttpLink({ uri: config.GRAPHQL_SERVER_URL });
const cache = new InMemoryCache();
const client = new ApolloClient({ link: httpLink, cache });

export default function App() {
  return (
    <ApolloProvider client={client}>
      <Root />
    </ApolloProvider>
  );
}

ReactDOM.render(<App />, document.getElementById('root'));
```

Managing local app state

```
// ...
const authLink = setContext((_, { headers }) => {
  const { user } = client.readQuery({ query: LOCAL_APP_STATE });
  return {
    headers: {
      ...headers,
      authorization: user ? `Bearer ${user.authToken}` : '',
    },
  };
});

const client = new ApolloClient({
  link: authLink.concat(httpLink),
  cache,
});
// ...
```

Managing local app state

```
// ....
import { LOCAL_APP_STATE } from './store';
// ....

const client = new ApolloClient({
  link: authLink.concat(httpLink),
  cache,
});
const initialLocalAppState = {
  component: { name: 'Home', props: {} },
  user: JSON.parse(window.localStorage.getItem('azdev:user')),
};
client.writeQuery({
  query: LOCAL_APP_STATE,
  data: initialLocalAppState,
});

export default function App() {
  // ....
}
```

Managing local app state

```
// ...
import { useQuery, gql } from '@apollo/client';
// ...
export const useStore = () => {
  // ...

  const useLocalAppState = (...stateMapper) => {
    const { data } = useQuery(LOCAL_APP_STATE);
    if (stateMapper.length === 1) {
      return data[stateMapper[0]];
    }
    return stateMapper.map((element) => data[element]);
  };

  // ...
};
```

```
// ...
import { useApolloClient, useQuery, gql } from '@apollo/client';
// ...
export const useStore = () => {
  // Delete the useState line
  const client = useApolloClient();

  // ...

  const setLocalAppState = (newState) => {
    if (newState.component) {
      newState.component.props = newState.component.props ?? {};
    }
    const currentState = client.readQuery({
      query: LOCAL_APP_STATE,
    });
    const updateState = () => {
      client.writeQuery({
        query: LOCAL_APP_STATE,
        data: { ...currentState, ...newState },
      });
    };
```

Managing local app state

```
    if (newState.user || newState.user === null) {
      client.onResetStore(updateState);
      client.resetStore();
    } else {
      updateState();
    }
  };

  const AppLink = ({ children, to, ...props }) => {
    // ....
  };

  return {
    useLocalAppState,
    setLocalAppState,
    AppLink,
  };
};
// Delete the React Context lines
```

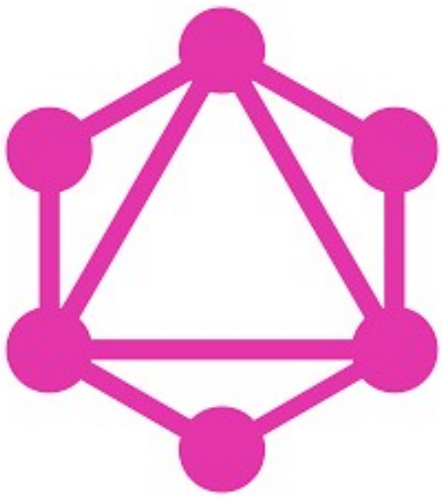
Continued code

Table of Contents

- Using Apollo Client with JavaScript
- Using Apollo Client with React
- Managing local app state
- **Implementing and using GraphQL subscriptions**

Implementing and using GraphQL subscriptions

```
export default function Home() {
  const { error, loading, data } =
useQuery(TASK_MAIN_LIST, {
    pollInterval: 5000,
  });

  // ·-·-·
}
```



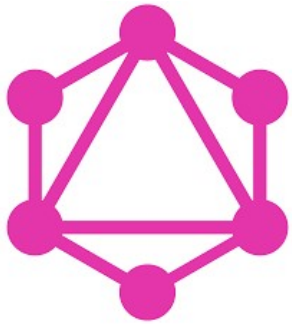
Implementing and using GraphQL subscriptions

```
export default function Home() {
  const { error, loading, refetch, data } = useQuery(TASK_MAIN_LIST);

  // ·-·-·

  return (
    <div>
      <Search />
      <div>
        <h1>Latest</h1>
        <button onClick={() => refetch()}>Refresh</button>
        {/* ·-·-· */}
      </div>
    </div>
  );
}
```

Implementing subscriptions

- Start by installing the apollo-server package.

```
$ npm install apollo-server
```



```
import { ApolloServer } from 'apollo-server';
// ...

async function main() {
  // ...

  server.listen(config.port, () => {
    console.log(`API server is running on port ${config.port}`);
  });

  const serverWS = new ApolloServer({ schema });

  serverWS.listen({ port: 4000 }).then(({ subscriptionsUrl }) => {
    console.log(`Subscriptions URL: ${subscriptionsUrl}`);
  });
};

main();
```

Implementing subscriptions

Implementing subscriptions

- The Pub/Sub operations will happen in multiple places, let's create a new file under api/src/pubsub.js to prepare a PubSub instance for any part of the API server code to use.

```
import { PubSub } from 'apollo-server';

const pubsub = new PubSub();

export { pubsub };
```

Implementing subscriptions

```
import { pubsub } from '../pubsub';
// ...

const MutationType = new GraphQLObjectType({
  name: 'Mutation',
  fields: () => ({
    // ...
    taskCreate: {
      type: TaskPayload,
      args: {
        input: { type: new GraphQLNonNull(TaskInput) },
      },
      resolve: async (
        source,
        { input },
        { mutators, currentUser },
      ) => {
        const { errors, task } = await mutators.taskCreate({
          input,
          currentUser,
        });
        if (errors.length === 0 && !task.isPrivate) {
          pubsub.publish(`TASK_MAIN_LIST_CHANGED`, {
            newTask: task,
          });
        }
        return { errors, task };
      },
    },
```

Continued code

```
    approachVote: {
      // ...
      resolve: async (
        source,
        { approachId, input },
        { mutators },
      ) => {
        const { errors, approach } = await mutators.approachVote({
          approachId,
          input,
        });
        if (errors.length === 0) {
          pubsub.publish(`VOTE_CHANGED_${approach.taskId}`, {
            updatedApproach: approach,
          });
        }
        return { errors, approach };
      },
    },
  },
});
```

Implementing subscriptions

```
import { GraphQLNonNull, GraphQLObjectType } from 'graphql';

import { pubsub } from '../pubsub';
import Task from './types/task';

const SubscriptionType = new GraphQLObjectType({
  name: 'Subscription',
  fields: () => ({
    taskMainListChanged: {
      type: new GraphQLNonNull(Task),
      resolve: async (source) => {
        return source.newTask;
      },
      subscribe: async () => {
        return pubsub.asyncIterator(['TASK_MAIN_LIST_CHANGED']);
      },
    },
  }),
});

export default SubscriptionType;
```

Implementing subscriptions

```
// ·-·-·
import SubscriptionType from './subscriptions';

export const schema = new GraphQLSchema({
  query: QueryType,
  mutation: MutationType,
  subscription: SubscriptionType,
});

console.log(printSchema(schema));
```

Implementing subscriptions

- To test the taskMainListChanged mutation, open the GraphQL Playground editor at http://localhost:4000/graphql and run the following operation.

```
subscription {
  taskMainListChanged {
    id
    content
  }
}
```

Playground - http://localhost:4

localhost:4000/graphql

taskMainListChanged

PRETTIFY HISTORY http://localhost:4000/graphql COPY CURL

```
subscription {
  taskMainListChanged {
    id
    content
  }
}
```

Listening …

DOCS

SCHEMA

QUERY VARIABLES HTTP HEADERS

TRACING QUERY PLAN

Playground - http://localhost:4

localhost:4000/graphql

taskMainListChanged

PRETTIFY HISTORY http://localhost:4000/graphql COPY CURL

```
subscription {
  taskMainListChanged {
    id
    content
  }
}
```

```
{
  "data": {
    "taskMainListChanged": {
      "id": "37",
      "content": "This is a new Task ..."
    }
  }
}
```

Listening …

DOCS

SCHEMA

QUERY VARIABLES HTTP HEADERS TRACING QUERY PLAN

```
import {
  GraphQLNonNull,
  GraphQLObjectType,
  GraphQLID,
} from 'graphql';

import { pubsub } from '../pubsub';
import Task from './types/task';
import Approach from './types/approach';

const SubscriptionType = new GraphQLObjectType({
  name: 'Subscription',
  fields: () => ({
    taskMainListChanged: {
      // ....
    },
    voteChanged: {
      type: new GraphQLNonNull(Approach),
      args: {
        taskId: { type: new GraphQLNonNull(GraphQLID) },
      },
      resolve: async (source) => {
        return source.updatedApproach;
      },
      subscribe: async (source, { taskId }) => {
        return pubsub.asyncIterator([`VOTE_CHANGED_${taskId}`]);
      },
    },
  }),
});
```

Implementing subscriptions

Apollo Server

```
import DataLoader from 'dataloader';
import { ApolloServer } from 'apollo-server';

import { schema } from './schema';
import pgApiWrapper from './db/pg-api';
import mongoApiWrapper from './db/mongo-api';

import * as config from './config';

async function main() {
  const pgApi = await pgApiWrapper();
  const mongoApi = await mongoApiWrapper();

  const server = new ApolloServer({
    schema,
    formatError: (err) => {
      const errorReport = {
        message: err.message,
        locations: err.locations,
        stack: err.stack ? err.stack.split('\n') : [],
        path: err.path,
      };
      console.error('GraphQL Error', errorReport);
      return config.isDev
        ? errorReport
        : { message: 'Oops! Something went wrong! :(' };
    },
```

Continued Code

```
    context: async ({ req }) => {
      const authToken =
        req && req.headers && req.headers.authorization
          ? req.headers.authorization.slice(7) // "Bearer "
          : null;
      const currentUser = await pgApi.userFromAuthToken(authToken);
      if (authToken && !currentUser) {
        throw Error('Invalid access token');
      }
      const loaders = {
        // ...
      };
      const mutators = {
        ...pgApi.mutators,
        ...mongoApi.mutators,
      };

      return { loaders, mutators, currentUser };
    },
  });

  server
    .listen({ port: config.port })
    .then(({ url, subscriptionsUrl }) => {
      console.log(`Server URL: ${url}`);
      console.log(`Subscriptions URL: ${subscriptionsUrl}`);
    });
}

main();
```

Implementing subscriptions

- Note that I got rid of the 4000 port and used the default config port (which is 4321).
- The new URLs are as follows:

Server URL: http://localhost:4321/
Subscriptions URL: ws://localhost:4321/graphql

Using subscriptions in UIs

```
import { WebSocketLink } from
"@apollo/client/link/ws";

const wsLink = new WebSocketLink({
  uri: GRAPHQL_SUBSCRIPTIONS_URL,
  options: { reconnect: true },
});
```

Using subscriptions in UIs

- Let's define the new GRAPHQL_SUBSCRIPTIONS_URL config value for this project.

```
export const GRAPHQL_SERVER_URL =
  process.env.GRAPHQL_SERVER_URL ||
'http://localhost:4321';
export const GRAPHQL_SUBSCRIPTIONS_URL =
  process.env.GRAPHQL_SUBSCRIPTIONS_URL ||
`ws://localhost:4321/graphql`;
```

```
// ...
import {
  ApolloProvider,
  ApolloClient,
  HttpLink,
  InMemoryCache,
  split,
} from '@apollo/client';
import { getMainDefinition } from '@apollo/client/utilities';
import { WebSocketLink } from "@apollo/client/link/ws";
// ...
const wsLink = new WebSocketLink({
  uri: config.GRAPHQL_SUBSCRIPTIONS_URL,
  options: { reconnect: true },
});
const splitLink = split(
  ({ query }) => {
    const definition = getMainDefinition(query);
    return (
      definition.kind === 'OperationDefinition' &&
      definition.operation === 'subscription'
    );
  },
  wsLink,
  authLink.concat(httpLink),
);

const client = new ApolloClient({
  link: splitLink,
  cache,
});
```

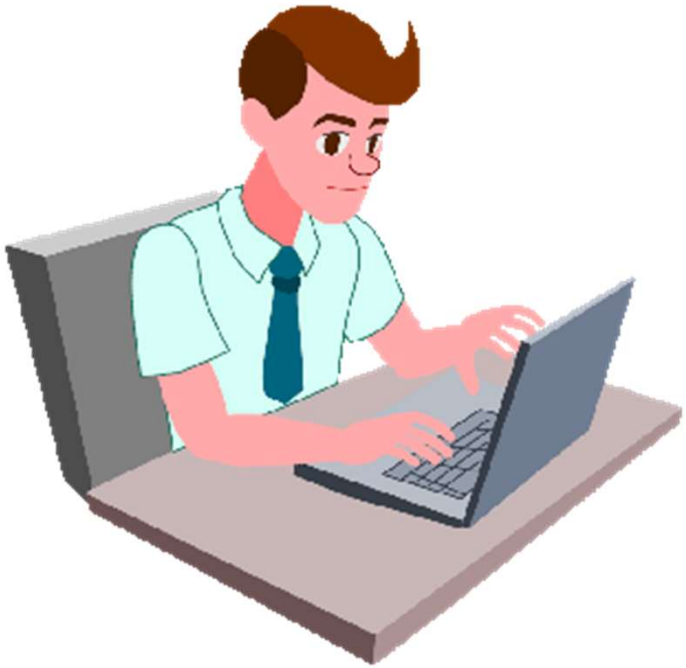
Using subscriptions in UIs

```
import { gql, useQuery, useSubscription, } from '@apollo/client';
const VOTE_CHANGED = gql`
  subscription voteChanged($taskId: ID!) {
    voteChanged(taskId: $taskId) {
      id
      voteCount
    }
  }
`;
// ....

export default function TaskPage({ taskId }) {
  // ....

  const { error, loading, data } = useQuery(TASK_INFO, {
    variables: { taskId },
  });

  useSubscription(VOTE_CHANGED, {
    variables: { taskId },
  });

  // ....
}
```

Using subscriptions in UIs

Summary

- A GraphQL client library like Apollo manages all the communications between a frontend application and its GraphQL API service.
- It issues data requests and makes their data responses available where needed.
- You can use Apollo Client with plain JavaScript or with view libraries like React, Vue, and Angular.
- For React, Apollo Client provides custom hook functions that greatly simplify the code in function components.



"Complete Lab"