Real-time Chat Application

React, GraphQL over Web Sockets, Subscriptions,

and Module Federation

Contents

[Introduction 3](#_Toc162626566)

[Application Overview 3](#_Toc162626567)

[Prerequisite 3](#_Toc162626568)

[Download asset/library 3](#_Toc162626569)

[Workspace 3](#_Toc162626570)

[Hands-on Steps 3](#_Toc162626571)

[Server Application - Quick start – GraphQL Yoga (the-guild.dev) 3](#_Toc162626572)

[Chat Application 8](#_Toc162626573)

[Use Subscription to Notify Clients in Real-time - Subscriptions - Apollo GraphQL Docs 31](#_Toc162626574)

[Server changes 31](#_Toc162626575)

[Client changes 34](#_Toc162626576)

[Integrate Chat Component into Another Project 38](#_Toc162626577)

[Create a Home Project 38](#_Toc162626578)

[Integrate Chat Component to a Home Project 41](#_Toc162626579)

# Introduction

The intention of this document is to explain how to create a real-time multi-user chat application in React using GraphQL, WebSocket, and Subscription. We will be using Syncfusion React UI components for building chat application UI design. Also, using the module federation feature of webpack 5, we will see how to integrate this chat application into another react application.

# Application Overview

There will be 3 different applications created as part of this workshop as below,

* **Server** – A GraphQL server to read and write data.
* **Client** – A front-end chat application that fetches and sends data from & to the server.
* **Home** – A micro frontend application that integrates the chat application.

# Prerequisite

1. React supported version >= 15.5.4+
2. [node](https://nodejs.org/en/) version >= 14.4.0+
3. For Yarn package manager, ensure version >= 0.25+
4. Visual Studio Code or any code editor

# Download asset/library

GitHub repo -

# Workspace

Create a workspace folder ***Real-Time-Chat-App***, open it in *VSCode* and follow below steps to create projects. The above-mentioned 3 projects should be maintained under this created workspace.

# Hands-on Steps

## Server Application - [Quick start – GraphQL Yoga (the-guild.dev)](https://the-guild.dev/graphql/yoga-server/docs)

1. Create a new folder ***server***under the *Real-Time-Chat-App* workspace and follow the below instructions.
2. Add a new file under the *server* folder and name it as ***server.js***.
3. Open the terminal and type the command ***cd server***.
4. Type the command ***npm init -y*** (*-y* will answer automatically yes to all the questions asked in the terminal). Ensure that the *package.json* file is added under the *server* folder.
5. Install the *graphql-yoga* package using the command ***npm install graphql-yoga***
6. Create another file and name it as ***server/schema.js***.
7. In *server/schema.js* file, use the following code example.

|  |
| --- |
| import { createSchema } from 'graphql-yoga'  var messages = [];  export const schema = createSchema({  typeDefs: `  type Message {  id: ID!  name: String!  text: String!  }    type Query {  messages: [Message!]  }  type Mutation{  postMessage(name: String!, text: String!) : ID!  }  `,  resolvers: {  Query: {  messages: () => messages  },  Mutation: {  postMessage: (parent, {name, text}) =>{  const id = messages.length;  messages.push({  id,  name,  text  });  return id;  }  }  }  }) |

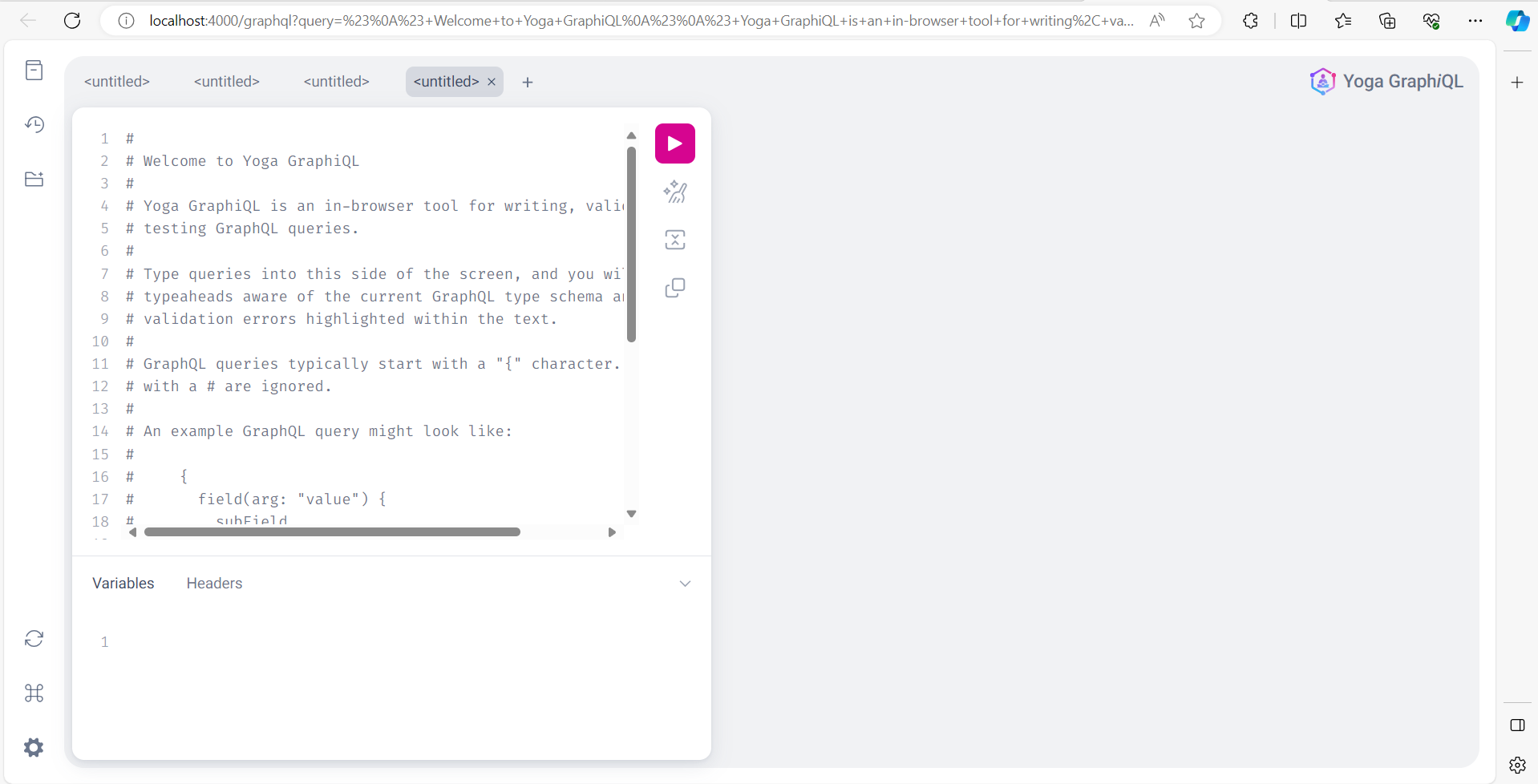
1. In *server/server.js* file, use the following code example.

|  |
| --- |
| import { createServer } from 'node:http'  import { createYoga } from 'graphql-yoga'  import { schema } from './schema.js'  // Create a new instance of graphql-yoga with the predefined schema  const yoga = createYoga({  schema: schema })  // Create a standard Node.js HTTP server with the yoga application  const server = createServer(yoga)  server.listen(4000, () => {  console.info(`Server is running on http://localhost:4000/graphql`)  }) |

1. In *server/package.json* file, add the below highlighted line to load an ES module,

|  |
| --- |
| {  "name": "server",  "version": "1.0.0",  "description": "",  "main": "server.js",  "type": "module",  "scripts": {  "test": "echo \"Error: no test specified\" && exit 1",  "start": "node server.js"  },  "keywords": [],  "author": "",  "license": "ISC",  "dependencies": {  "graphql-yoga": "^5.1.1"  }  } |

1. Now, run the app using the command *npm start* in the terminal.
2. The GraphiQL window opens like below,



1. Clear the content at the left side of the screen and type the below query. Then execute this query and look at the result from the right side of the screen which shows empty result.

|  |
| --- |
| query {  messages {  id  text  name  }  } |

A screenshot of a computer

Description automatically generated

1. Now open a new tab in GraphiQL and type the below mutation code, and then press the execute button.

|  |
| --- |
| mutation{  postMessage(name: "User 1", text: "Hello Everyone")  } |

A screenshot of a computer

Description automatically generated

1. Now, navigate again to the query tab and execute it to see the newly added data shown as query result.

A screenshot of a computer

Description automatically generated

Now, the server is all set up and running successfully.

## Chat Application

1. Open a new PowerShell terminal and run the command *npm install create-mf-app*.
2. Then type the command **npx create-mf-app** with below details.
   1. Pick the name of your App: ***ChatApp***
   2. Choose project type as ***Application.***
   3. Provide the port number as ***8080.***
   4. Choose the framework as ***React.***
   5. Choose ***JavaScript*** as language preference.
   6. Choose ***CSS*** for styling.
   7. Choose **Webpack** for Packer
   8. Use ***cd ChatApp*** and run ***npm install***command.
3. Install the Apollo client package using the command ***npm install @apollo/client***
4. Add a new file under *ChatApp/src* folder and name it as ***Chat.jsx***.
5. In *ChatApp/src/Chat.jsx*, type the following code to connect to the server,

|  |
| --- |
| import { ApolloClient, InMemoryCache, ApolloProvider } from '@apollo/client';  import { HttpLink } from '@apollo/client';  const httpLink = new HttpLink({  uri: 'http://localhost:4000/graphql'  });  const client = new ApolloClient({  link: httpLink,  cache: new InMemoryCache()  });  const Chat = () => {  return(  <div>I’m a chat window</div>  )  }  export default()=>(  <ApolloProvider client={client}>  <Chat/>  </ApolloProvider>  ); |

1. In *ChatApp/src/App.jsx*, do the following highlighted changes,

|  |
| --- |
| import React from "react";  import ReactDOM from "react-dom";  import "./index.css";  import Chat from "./Chat";  const App = () => (  <Chat/>  );  ReactDOM.render(<App />, document.getElementById("app")); |

1. Run the *ChatApp* using ***npm start*** command and look at the basic output.

A screenshot of a computer

Description automatically generated

1. Add the code highlighted below in *ChatApp/src/Chat.jsx* to retrieve the data from server as follows,

|  |
| --- |
| import { ApolloClient, InMemoryCache, ApolloProvider, useQuery, gql } from '@apollo/client';  import { HttpLink } from '@apollo/client';  const httpLink = new HttpLink({  uri: 'http://localhost:4000/graphql'  });  const client = new ApolloClient({  link: httpLink,  cache: new InMemoryCache()  });  const GET\_MESSAGES = gql`  query {  messages {  id  name  text  }  }`;  const Messages = () =>{  const {data} = useQuery(GET\_MESSAGES);  if(!data){  return null;  }  return JSON.stringify(data);  }  const Chat = () => {  return(  <div><Messages /></div>  )  }  export default()=>(  <ApolloProvider client={client}>  <Chat/>  </ApolloProvider>  ); |

1. Save the changes and you can see the data from server as follows:

A screenshot of a computer

Description automatically generated

1. Go to the *ChatApp* terminal and stop the execution by pressing *Ctrl+C* followed by typing the letter *y* and then press *enter*.
2. Now, let’s add UI components to the chat application and display the above retrieved data in it. First install the React [ListView](https://www.syncfusion.com/react-components/react-listview), [TextBox](https://www.syncfusion.com/react-components/react-textbox), and [Avatar](https://www.syncfusion.com/react-components/react-avatar) component packages of Syncfusion using the below commands in the *ChatApp* terminal one by one,

|  |
| --- |
| npm install @syncfusion/ej2-react-lists  npm install @syncfusion/ej2-react-inputs npm install @syncfusion/ej2-layouts |

1. Do the following changes in *ChatApp/src/Chat.jsx* to display the retrieved data in a Syncfusion React List View component.

|  |
| --- |
| import { ApolloClient, InMemoryCache, ApolloProvider, useQuery, gql } from '@apollo/client';  import { HttpLink } from '@apollo/client';  import { ListViewComponent } from '@syncfusion/ej2-react-lists';  import {registerLicense} from '@syncfusion/ej2-base'  registerLicense("enter your license key here");  const httpLink = new HttpLink({  uri: 'http://localhost:4000/graphql'  });  const client = new ApolloClient({  link: httpLink,  cache: new InMemoryCache()  });  const GET\_MESSAGES = gql`  query {  messages {  id  name  text  }  }`;  const Messages = ({user}) =>{  const {data} = useQuery(GET\_MESSAGES);  let listdata = [];  if(!data){  return null;  }  else {data.messages.map(({id, name: messageUser, text}) => (  listdata.push({id: id, user: messageUser, text: text, type: user === messageUser ? "sender" : "receiver" })  ))  }  return (  <>  <ListViewComponent id='List' dataSource={listdata} height= "500px" headerTitle="Let's Chat!" showHeader={true} />  </>  )  }  const Chat = () => {  return(  <div><Messages user="User 1"/></div>  )  }  export default()=>(  <ApolloProvider client={client}>  <Chat/>  </ApolloProvider>  ); |

1. Add the CSS reference of Syncfusion React UI components (ListView, TextBox, and Avatar) in *ChatApp/src/index.css* page as follows,

|  |
| --- |
| @import "../node\_modules/@syncfusion/ej2-base/styles/bootstrap5.css";  @import "../node\_modules/@syncfusion/ej2-react-lists/styles/bootstrap5.css";  @import "../node\_modules/@syncfusion/ej2-react-inputs/styles/bootstrap5.css";  @import '../node\_modules/@syncfusion/ej2-layouts/styles/bootstrap5.css'; |

1. Add CDN link reference for Bootstrap CSS in *ChatApp/src/index.html* as follows,

|  |
| --- |
| <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet" />  <title>ChatApp</title>  </head> |

1. Save the changes and run with the command *npm start* in the *ChatApp* terminal. Now, look at the output to ensure whether the basic list view component rendered on the page,

A screenshot of a browser

Description automatically generated

1. To design the ListView component to look like a chat window, open the *ChatApp/src/Chat.jsx* file and use the template property like below,

|  |
| --- |
| import { ApolloClient, InMemoryCache, ApolloProvider, useQuery, gql } from '@apollo/client';  import { HttpLink } from '@apollo/client';  import { ListViewComponent } from '@syncfusion/ej2-react-lists';  import {registerLicense} from '@syncfusion/ej2-base'  registerLicense("enter your license key here");  const httpLink = new HttpLink({  uri: 'http://localhost:4000/graphql'  });  const client = new ApolloClient({  link: httpLink,  cache: new InMemoryCache()  });  const GET\_MESSAGES = gql`  query {  messages {  id  name  text  }  }`;  const Messages = ({user}) =>{  function listTemplate(data) {  const sendertemplate = (<div className="settings sender-settings">  <div className="content sender-content">  <div>{data.text}</div>  </div>  </div>);  const receivertemplate = (<div className="settings receiver-settings">  <div className="e-avatar e-avatar-circle">{data.user}</div>  <div className="content receiver-content">  {data.text}  </div>  </div>);  return <div>{data.type !== "receiver" ? sendertemplate : receivertemplate}</div>;  }  const {data} = useQuery(GET\_MESSAGES);  let listdata = [];  if(!data){  return null;  }  else {data.messages.map(({id, name: messageUser, text}) => (  listdata.push({id: id, user: messageUser, text: text, type: user === messageUser ? "sender" : "receiver" })  ))  }  return (  <>  <ListViewComponent id='List' dataSource={listdata} height= "500px" headerTitle="Let's Chat!" showHeader={true} template={listTemplate} />  </>  )  }  const Chat = () => {  return(  <div><Messages user="User 1"/></div>  )  }  export default()=>(  <ApolloProvider client={client}>  <Chat/>  </ApolloProvider>  ); |

1. Also, add the appropriate CSS properties for customizing list view in *ChatApp/src/index.css* as follows,

|  |
| --- |
| .settings {  display: flex;  font-size: 13px;  }  .sender-settings{  justify-content: flex-end;  }  .receiver-settings{  justify-content: flex-start;  }  .content{  color: black;  padding: 8px;  border-radius: 8px;  max-width: 60%;  }  .receiver-content{  background-color: #fff;  margin-left: 12px;  }  .sender-content{  background-color: #dbf0f9;  }  .e-avatar{  height: 34px;  width: 34px;  }  .e-send:before {  content: '\e763';  font-family: e-icons;  font-size: 14px;  }  #List .e-list-item {  height: auto;  padding-bottom: 0px;  padding-top: 10px;  border-bottom: none !important;  }  .e-listview{  background-color: #F9F8F8;  }  .e-listview .e-list-header{  background: rgb(2, 120, 215);  height: 50px;  font-size: 18px;  font-weight: 700;  color: white;  }  .e-listview .e-list-item:hover {  background: #F9F8F8 !important;  cursor: pointer !important;  }  .e-listview{  background-color: #F9F8F8;  } |

1. Save all the changes and look at the output to notice the changes.

A screenshot of a computer

Description automatically generated

1. In *ChatApp/src/Chat.jsx* file, if you change the user value manually from User 1 to User 2 like below, you can notice the difference in output as follows,

|  |
| --- |
| const Chat = () => {  return(  <div><Messages user="User 2"/></div>  )  } |

A screenshot of a computer

Description automatically generated

1. Next, we need to add a textbox component and a CSS button to the bottom of the chat UI to allow the messages to be added dynamically. Also, I am changing the static username value with a state variable, as I need to maintain the user name values on every session. In *ChatApp/src/Chat.jsx* file, do the highlighted changes below in your code.

|  |
| --- |
| import React from "react";  import { ApolloClient, InMemoryCache, ApolloProvider, useQuery, gql } from '@apollo/client';  import { HttpLink } from '@apollo/client';  import { TextBoxComponent } from '@syncfusion/ej2-react-inputs';  import { ListViewComponent } from '@syncfusion/ej2-react-lists';  import {registerLicense} from '@syncfusion/ej2-base'  registerLicense("enter your license key here");  const httpLink = new HttpLink({  uri: 'http://localhost:4000/graphql'  });  const client = new ApolloClient({  link: httpLink,  cache: new InMemoryCache()  });  const GET\_MESSAGES = gql`  query {  messages {  id  name  text  }  }`;  const Messages = ({user}) => {  …  …  return (  <>  <ListViewComponent id='List' dataSource={listdata} height= "500px" headerTitle="Let's Chat!" showHeader={true} template={listTemplate} />  </>  )  }  const Chat = () => {  const [state,setState] = React.useState({  name: "User 1",  text: ""  });  return(  <>  <Messages user={state.name}/>  <div className="row custom-margin custom-padding-5">  <div className="col-xs-12 col-sm-12 col-lg-12 col-md-12">  <div className="e-input-group">  <TextBoxComponent placeholder="Type a message" value={state.text} onBlur={(evt)=> setState({  ...state,  text: evt.target.value,  })}/>  <span className="e-input-group-icon e-send" onClick={()=>onSend()}></span>  </div>  </div>  </div>  </>  )  }  export default()=>(  <ApolloProvider client={client}>  <Chat/>  </ApolloProvider>  ); |

1. Let’s check the output first.

A screenshot of a computer

Description automatically generated

1. For this demo purpose, let’s generate the usernames randomly at the initial loading of the application and save it to the state variable as highlighted below.

|  |
| --- |
| import { React, useEffect, useState } from "react";  import { ApolloClient, InMemoryCache, ApolloProvider, useQuery, gql } from '@apollo/client';  import { HttpLink } from '@apollo/client';  import { TextBoxComponent } from '@syncfusion/ej2-react-inputs';  import { ListViewComponent } from '@syncfusion/ej2-react-lists';  import {registerLicense} from '@syncfusion/ej2-base'  registerLicense("enter your license key here");  const httpLink = new HttpLink({  uri: 'http://localhost:4000/graphql'  });  const client = new ApolloClient({  link: httpLink,  cache: new InMemoryCache()  });  const GET\_MESSAGES = gql`  query {  messages {  id  name  text  }  }`;  const Messages = ({user}) => {  const userName = "Hi " + user + ", Let's Chat!";  function listTemplate(data) {  const sendertemplate = (<div className="settings sender-settings">  <div className="content sender-content">  <div>{data.text}</div>  </div>  </div>);  const receivertemplate = (<div className="settings receiver-settings">  <div className="e-avatar e-avatar-circle">{data.user.split(' ').map(word=>word.charAt(0)).join('').toUpperCase()}</div>  <div className="content receiver-content">  {data.text}  </div>  </div>);  return <div>{data.type !== "receiver" ? sendertemplate : receivertemplate}</div>;  }  …  …  return (  <>  <ListViewComponent id='List' dataSource={listdata} height= "500px" headerTitle={userName} showHeader={true} template={listTemplate} />  </>  )  }  const Chat = () => {  const [state,setState] = useState({  name: "",  text: ""  });  useEffect(() =>{  setState({  ...state,  name: "User " + Math.floor(Math.random() \* (9 - 1 + 1) + 1),  })  }, [])  return(  <>  <Messages user={state.name}/>  <div className="row custom-margin custom-padding-5">  <div className="col-xs-12 col-sm-12 col-lg-12 col-md-12">  <div className="e-input-group">  <TextBoxComponent placeholder="Type a message" value={state.text} onBlur={(evt)=> setState({  ...state,  text: evt.target.value,  })}/>  <span className="e-input-group-icon e-send" onClick={()=>onSend()}></span>  </div>  </div>  </div>  </>  )  }  export default()=>(  <ApolloProvider client={client}>  <Chat/>  </ApolloProvider>  ); |

1. Now, the output looks below due to the randomly generated username for the current session. Also, you can notice that the list view title is modified as well as the avatar text is customized to display only 2 letters.

A screenshot of a computer

Description automatically generated

1. Next, to add messages dynamically, we need to use mutation. So, let’s define the ***onSend*** function that should be triggered whenever the send button is clicked. [Mutations in Apollo Client - Apollo GraphQL Docs](https://www.apollographql.com/docs/react/data/mutations#executing-a-mutation)
2. Also, to get the changes at some interval of time after clicking the send button, I am going to set pollInterval option inside the useQuery() function as highlighted. Look at the highlighted changes below to be made in *ChatApp/src/Chat.jsx* file.

|  |
| --- |
| import { React, useEffect, useState } from "react";  import { ApolloClient, InMemoryCache, ApolloProvider, useQuery, gql, useMutation } from '@apollo/client';  …  …  const client = new ApolloClient({  link: httpLink,  cache: new InMemoryCache()  });  const GET\_MESSAGES = gql`  query {  messages {  id  name  text  }  }`;  export const POST\_MESSAGE = gql`  mutation ($name:String!, $text:String!) {  postMessage(name: $name, text: $text)  }`;  const Messages = ({user}) => {  …  …  const {data} = useQuery(GET\_MESSAGES, { pollInterval: 500});  let listdata = [];  if(!data){  return null;  }  else {  …  …  }  return (  <>  <ListViewComponent id='List' dataSource={listdata} height= "500px" headerTitle={userName} showHeader={true} template={listTemplate} />  </>  )  }  const Chat = () => {  const [state,setState] = useState({  name: "",  text: ""  });  useEffect(() =>{  setState({  ...state,  name: "User " + Math.floor(Math.random() \* (9 - 1 + 1) + 1),  })  }, [])  const [postMessage] = useMutation(POST\_MESSAGE);  const onSend = () => {  if(state.text.length > 0){  postMessage({  variables: state  });  }  setState({  ...state,  text: ''  })  }  return(  <div>  <Messages user={state.name}/>  <div style={{paddingTop: "35px"}} className="row custom-margin custom-padding-5">  …  …  </div>  </div>  )  }  export default()=>(  <ApolloProvider client={client}>  <Chat/>  </ApolloProvider>  ); |

1. Save the changes and have a look at the output window. Type some messages and click the send button.

A screenshot of a computer

Description automatically generated

The next step is to find out the way to automatically notify the messages to all clients as soon as the send button is clicked.

# Use Subscription to Notify Clients in Real-time - [Subscriptions - Apollo GraphQL Docs](https://www.apollographql.com/docs/react/data/subscriptions#defining-a-subscription)

The [GraphQL over WebSocket](https://the-guild.dev/blog/graphql-over-websockets) protocol supports all 3 GraphQL operations: Queries, Mutations and Subscriptions.

## Server changes

1. Stop the server project, open *server/schema.js* file and start making the following changes in it to include code for subscription.

|  |
| --- |
| import { createSchema, createPubSub } from 'graphql-yoga'  import { Repeater } from 'graphql-yoga';  const pubSub = createPubSub();  var messages = [];  export const schema = createSchema({  typeDefs: `  type Message {  id: ID!  name: String!  text: String!  }    type Query {  messages: [Message!]  }  type Mutation{  postMessage(name: String!, text: String!) : ID!  }  type Subscription {  messages: [Message!]  }  `,  resolvers: {  Query: {  messages: () => messages  },  Mutation: {  postMessage: (parent, {name, text}) =>{  const id = messages.length;  messages.push({  id,  name,  text  });  pubSub.publish("messages", { messages })  return id;  }  },  Subscription: {  messages: {  subscribe: () => {  return Repeater.merge(  [  new Repeater(async (push, stop) => {  push({ messages });  await stop;  }),  // Allows clients to subscribe to real-time updates of messages pubSub.subscribe("messages"),  ]  )  }  }  }  }  }) |

1. Open the *server* terminal, stop the execution and install the two packages below one by one.

|  |
| --- |
| npm install graphql-ws  npm install ws |

1. Do the below highlighted changes in *server/server.js* file.

|  |
| --- |
| import { createServer } from 'node:http'  import { createYoga } from 'graphql-yoga'  import { useServer } from 'graphql-ws/lib/use/ws'  import { WebSocketServer } from 'ws'  import { schema } from '../server/schema.js'  const yoga = createYoga({  schema: schema  })  const server = createServer(yoga)  // Create a WebSocket server that attaches to the HTTP server  const wsServer = new WebSocketServer({  server: server,  // path: yoga.graphqlEndpoint  })  // Integrate graphql-yoga with the WebSocket server using graphql-ws  useServer(  {  execute: (args) => args.execute(args),  subscribe: (args) => args.subscribe(args),  // To handle the incoming subscription requests  **onSubscribe**: async (ctx, msg) => {  const { schema, execute, subscribe, contextFactory, parse, validate } =  // To get the necessary functions and context for executing GraphQL operations.  **yoga.getEnveloped**({  ...ctx,  req: ctx.extra.request,  socket: ctx.extra.socket,  params: msg.payload  })    // parses and validates the subscription request and returns the arguments needed for execution.  const args = {  schema,  operationName: msg.payload.operationName,  document: parse(msg.payload.query),  variableValues: msg.payload.variables,  contextValue: await contextFactory(),  execute,  subscribe  }    const errors = validate(args.schema, args.document)  if (errors.length) return errors  return args  }  },  wsServer  );  server.listen(4000, () => {  console.info(`Server is running on http://localhost:4000/graphql`)  }) |

## Client changes

1. Open the *ChatApp* terminal, stop the execution, and install the *graphql-ws* package using ***npm i graphql-ws***.
2. In *ChatApp/src/Chat.jsx* file, start including the below highlighted changes to include code for subscription.

|  |
| --- |
| import { React, useEffect, useState } from "react";  import { ApolloClient, InMemoryCache, ApolloProvider, useQuery, gql, useMutation, useSubscription } from '@apollo/client';  import { GraphQLWsLink } from '@apollo/client/link/subscriptions';  import { createClient } from 'graphql-ws';  import { split, HttpLink } from '@apollo/client';  import { getMainDefinition } from '@apollo/client/utilities';  …  …  const httpLink = new HttpLink({  uri: 'http://localhost:4000/graphql'  });  const wsLink = new GraphQLWsLink(createClient({  url: 'ws://localhost:4000/graphql'  }));  const splitLink = split(  ({ query }) => {  const definition = getMainDefinition(query);  return (  definition.kind === 'OperationDefinition' &&  definition.operation === 'subscription'  );  },  wsLink,  httpLink,  );  const client = new ApolloClient({  link: splitLink,  cache: new InMemoryCache()  });  const GET\_MESSAGES = gql`  subscription {  messages {  id  name  text  }  }`;  export const POST\_MESSAGE = gql`  mutation ($name:String!, $text:String!) {  postMessage(name: $name, text: $text)  }`;  const Messages = ({user}) =>{  const userName = "Hi " + user + ", Let's Chat!";  function listTemplate(data) {  …  …  }  const {data} = useSubscription(GET\_MESSAGES);  let listdata = [];  …  …  return (  <>  …  …  </>  )  }  const Chat = () => {  const [state,setState] = useState({  name: "",  text: ""  });  useEffect(() =>{  setState({  ...state,  name: "User " + Math.floor(Math.random() \* (9 - 1 + 1) + 1),  })  }, [])  const [postMessage] = useMutation(POST\_MESSAGE);  const onSend = () => {  …  …  }  return(  <div>  <Messages user={state.name}/>  <div style={{paddingTop: "35px"}} className="row custom-margin custom-padding-5">  …  …  </div>  </div>  )  }  export default()=>(  <ApolloProvider client={client}>  <Chat/>  </ApolloProvider> ); |

Now, save the changes and re-run the server and chat applications. Open 3 browsers running on localhost:8080 and change the username in all the 3 browsers as shown below. Also, type messages on every browser and click on the send button to see the changes happening in real-time on all the browsers.

A screenshot of a computer screen

Description automatically generated

To move the scroller to the bottom position of the listview for every new message we send, it is necessary to do the following highlighted changes,

|  |
| --- |
| import React, { useRef, useEffect, useState } from "react";  import { ApolloClient, InMemoryCache, ApolloProvider, useQuery, gql, useMutation, useSubscription } from '@apollo/client';  …  import { split, HttpLink } from '@apollo/client';  import { getMainDefinition } from '@apollo/client/utilities';  …  …  const httpLink = new HttpLink({  uri: 'http://localhost:4000/graphql'  });  …  const client = new ApolloClient({  link: splitLink,  cache: new InMemoryCache()  });  const GET\_MESSAGES = gql`  subscription {  messages {  id  name  text  }  }`;  export const POST\_MESSAGE = gql`  mutation ($name:String!, $text:String!) {  postMessage(name: $name, text: $text)  }`;  const Messages = ({user}) =>{  const listObj = useRef(null);  const userName = "Hi " + user + ", Let's Chat!";  function listTemplate(data) {  …  …  }  const {data} = useSubscription(GET\_MESSAGES);  let listdata = [];  …  …  function onActionComplete(args) {  setTimeout(function () {  if(listObj){  listObj.current.element.scrollTop = listObj.current.element.scrollHeight;  }  }, 0);  }  return (  <>  <ListViewComponent id='List' ref={listObj} dataSource={listdata} height= "500px" headerTitle={userName} showHeader={true} statelessTemplates={['template']} template={listTemplate} actionComplete={onActionComplete} />  </>  )  }  const Chat = () => {  const [state,setState] = useState({  name: "",  text: ""  });  …  const [postMessage] = useMutation(POST\_MESSAGE);  const onSend = () => {  …  …  }  return(  <div>  <Messages user={state.name}/>  <div style={{paddingTop: "35px"}} className="row custom-margin custom-padding-5">  …  …  </div>  </div>  )  } |

# Integrate Chat Component into Another Project

## Create a Home Project

1. Open a new PowerShell terminal and type the command **npx create-mf-app** with below details.
   1. Pick the name of your App: ***HomeApp***
   2. Choose project type as ***Application.***
   3. Provide the port number as ***8081.***
   4. Choose the framework as ***React.***
   5. Choose ***JavaScript*** as language preference.
   6. Choose ***CSS*** for styling.
   7. Choose **Webpack** for Packer
   8. Use ***cd HomeApp*** and run ***npm install***command.
2. Open *HomeApp/src/App.jsx* page, and add the following code,

|  |
| --- |
| const App = () => (  <div className="chat-container">  <h1>Let's Chat!</h1>  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Faucibus et molestie ac feugiat sed. Sagittis id consectetur purus ut. Arcu non odio euismod lacinia at quis risus sed. Arcu dui vivamus arcu felis bibendum. In nibh mauris cursus mattis molestie a. Congue nisi vitae suscipit tellus mauris. Egestas tellus rutrum tellus pellentesque. Nisl tincidunt eget nullam non. Integer malesuada nunc vel risus commodo viverra maecenas. Consectetur adipiscing elit pellentesque habitant morbi tristique senectus et. Ornare quam viverra orci sagittis eu volutpat odio facilisis. Eu lobortis elementum nibh tellus molestie nunc non.</p>  <p>Dui accumsan sit amet nulla facilisi morbi tempus iaculis urna. In fermentum posuere urna nec tincidunt praesent semper feugiat nibh. Nunc vel risus commodo viverra maecenas. Congue nisi vitae suscipit tellus mauris a diam maecenas sed. Eget egestas purus viverra accumsan. Orci dapibus ultrices in iaculis nunc sed augue. Sem nulla pharetra diam sit amet. Feugiat sed lectus vestibulum mattis. Ornare arcu odio ut sem nulla pharetra diam. Velit ut tortor pretium viverra suspendisse potenti nullam ac tortor.</p>  <div className="chat-window">  <div className="chat">  Place your chat window here!  </div>  </div>  </div>  );  ReactDOM.render(<App />, document.getElementById("app")); |

1. Open *HomeApp/src/index.css* file, remove the existing ***container*** style and add the below code,

|  |
| --- |
| .chat-window {  font-size: 14px;  margin: auto;  max-width: 800px;  margin-top: 20px;  position: fixed;  bottom: 15px;  right: 0;  pointer-events: none;  width: 30%;  }  .chat {  display: inline-block;  vertical-align: bottom;  position: relative;  margin: 0 5px;  pointer-events: auto;  width: 95%;  }  .chat-container {  padding: 10px;  } |

1. Save the changes and run the *HomeApp* using the command ***npm start***.

A screenshot of a computer

Description automatically generated

The next step is to integrate the chat component created earlier into this home project at the bottom right corner as marked below,

A screenshot of a computer

Description automatically generated

## Integrate Chat Component to a Home Project

1. Open the *ChatApp/webpack.config.js* and expose chat component to be accessed by external projects.

|  |
| --- |
| plugins: [  new ModuleFederationPlugin({  name: "ChatApp",  filename: "remoteEntry.js",  remotes: {},  exposes: {  "./Chat": "./src/Chat"  },  shared: {  ...deps,  react: {  singleton: true,  requiredVersion: deps.react,  }, |

1. Open *ChatApp/src/Chat.jsx* and add the below highlighted code to export the CSS added for Chat component to external projects,

|  |
| --- |
| import React from "react";  import "./index.css";  import { ApolloClient, InMemoryCache, ApolloProvider, useQuery, gql, useMutation, useSubscription } from '@apollo/client';  import { GraphQLWsLink } from '@apollo/client/link/subscriptions';  import { createClient } from 'graphql-ws';  import { split, HttpLink } from '@apollo/client'; |

1. Open *HomeApp/webpack.config.js* and include *chat* component on remotes object,

|  |
| --- |
| plugins: [  new ModuleFederationPlugin({  name: "HomeApp",  filename: "remoteEntry.js",  remotes: {  ChatApp: "ChatApp@http://localhost:8080/remoteEntry.js",  },  exposes: {},  shared: {  ...deps,  react: { |

1. Open *HomeApp/src/App.jsx* and import *Chat* component,

|  |
| --- |
| import React from "react";  import ReactDOM from "react-dom";  import "./index.css";  const Chat = React.lazy(() => import("ChatApp/Chat"));  const App = () => (  <div className="chat-container">  …  … |

1. Call Chat component inside the return function of *HomeApp/src/App.jsx****,*** wrapping it inside the ***React.Suspense***.

|  |
| --- |
| import React from "react";  import ReactDOM from "react-dom";  import "./index.css";  const Chat = React.lazy(() => import("ChatApp/Chat"));  const App = () => (  <div className="chat-container">  <h1>Let's Chat!</h1>  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Faucibus et molestie ac feugiat sed. Sagittis id consectetur purus ut. Arcu non odio euismod lacinia at quis risus sed. Arcu dui.</p>  <p>Dui accumsan sit amet nulla facilisi morbi tempus iaculis urna. In fermentum posuere urna nec tincidunt praesent semper feugiat nibh. Nunc vel risus commodo viverra maecenas. Congue nisi.</p>  <div className="chat-window">  <div className="chat">  <React.Suspense fallback="Loading">  <Chat></Chat>  </React.Suspense>  </div>  </div>  </div>  ); |

1. Open *HomeApp/src/index.html* page and add the bootstrap CDN reference link,

|  |
| --- |
| <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet" />  <title>HomeApp</title>  </head> |

1. Stop the execution of *ChatApp* and *HomeApp* projects in terminal. Save all the changes and re-run them and then verify if the *Chat* component is integrated into the home page.

A screenshot of a computer

Description automatically generated