Continue

This article shows how to fetch GraphQL data in React with the hook and attach the result to your UI. You'll also learn how simplifies data management code by tracking error and loading states for you.PrerequisitesThis article assumes you're familiar with building basic GraphQL queries. If you need a refresher, we recommend this guide. You can also build example queries against Apollo's full-stack tutorial server.This article also assumes that you've already set up Apollo Client and have wrapped your React app in an component. For more information, see the getting started guide.To follow along with the examples below, open up our starter project and sample GraphQL server on CodeSandbox. You can view the completed version of the app here.Executing a queryThe React hook is the primary API for executing queries in an Apollo application. To run a query within a React component, call and pass it a GraphQL query string. When your component renders, returns an object from Apollo Client that contains , , and properties you can use to render your UI.Note: in Apollo Client >= 3.8, Suspense data fetching hooks are available for querying data within boundaries using React 18's new concurrent rendering model. For more information see Apollo Client's Suspense docs.Let's look at an example. First, we'll create a GraphQL query named . Remember to wrap query strings in the function to parse them into query documents:Next, we'll create a component named . Inside it, we'll pass our query to the hook:As our query executes and the values of , , and change, the component can intelligently render different UI elements according to the query's state:As long as is (indicating the query is still in flight), the component presents a notice.When loading is and there is no , the query has completed. The component renders a dropdown menu that's populated with the list of dog breeds returned by the server.When the user selects a dog breed from the populated dropdown, the selection is sent to the parent component via the provided function.In the next step, we'll associate the dropdown with a more sophisticated query that uses GraphQL variables.Caching query resultsWhenever Apollo Client fetches query results from your server, it automatically caches those results locally. This makes later executions of that same query extremely fast.To see this caching in action, let's build a new component called . accepts a prop called that reflects the current value of the dropdown menu in our component:Notice that we're providing a configuration option () to the hook this time. The option is an object that contains all of the variables we want to pass to our GraphQL query. In this case, we want to pass the currently selected from the dropdown.Select from the dropdown to see its photo appear. Then switch to another breed, and then switch back to . You'll notice that the bulldog photo loads instantly the second time around. This is the cache at work!Next, let's learn some techniques for ensuring that our cached data is fresh.Updating cached query resultsSometimes, you want to make sure that your query's cached data is up to date with your server's data. Apollo Client supports two strategies for this: polling and refetching.PollingPolling provides near-real-time synchronization with your server by executing your query periodically at a specified interval. To enable polling for a query, pass a configuration option to the hook with an interval in milliseconds:By setting to 500, we fetch the current breed's image from the server every 0.5 seconds. Note that if you set to , the query does not poll.You can also start and stop polling dynamically with the and functions that are returned by the hook. When using these functions, set the configuration option as a parameter of the function.RefetchingRefetching enables you to refresh query results in response to a particular useraction, as opposed to using a fixed interval.Let's add a button to our component that calls our query's function whenever it's clicked.You can optionally provide a new object to the function.If you avoid passing a object and use only , the queryuses the same that it used in its previous execution.Click the button and notice that the UI updates with a new dog photo. Refetching is an excellent way to guarantee fresh data, but it introduces some complexity with loading state. In the next section, we'll cover strategies for handling complex loading and error state.Providing new variables to You call with a new set of variables like so:If you provide new values for some of your original query's variables but not all of them, uses each omitted variable's original value.Inspecting loading statesWe've already seen that the hook exposes our query's current loading state. This is helpful when a query first loads, but what happens to our loading state when we're refetching or polling?Let's return to our refetching example from the previous section. If you click the refetch button, you'll see that the component doesn't re-render until the new data arrives. What if we want to indicate to the user that we're refetching the photo?The hook's result object provides fine-grained information about the status of the query via the property. To take advantageof this information, we set the option to so our query component re-renders while a refetch is in flight:Enabling this option also ensures that the value of updates accordingly, even if you don't want to use the more fine-grained information provided by the property.The property is a enum that represents different loading states. Refetch is represented by , and there are also values for polling and pagination. For a full list of all the possible loading states, check out the source.To view a complete version of the app we just built, check out the CodeSandbox here.Inspecting error statesYou can customize your query error handling by providing the configuration option to the hook. The default value is , which tells Apollo Client to treat all GraphQL errors as runtime errors. In this case, Apollo Client discards any query response data returned by the server and sets the property in the result object.If you set to , does not discard query response data, allowing you to render partial results.For more information, see Handling operation errors.Manual execution with When React renders a component that calls , Apollo Client automatically executes the corresponding query. But what if you want to execute a query in response to a different event, such as a user clicking a button?The hook is perfect for executing queries in response to events besides component rendering. Unlike with , when you call , it does not immediately execute its associated query. Instead, it returns a query function in its result tuple that you call whenever you're ready to execute the query.Here's an example:The first item in 's return tuple is the query function, and the second item is the same result object returned by .As shown above, you can pass options to the query function just like you pass them to itself. If you pass a particular option to both, the value you pass to the query function takes precedence. This is a handy way to pass default options to and then customize those options in the query function.For a full list of supported options, see the API reference.Setting a fetch policyBy default, the hook checks the Apollo Client cache to see if all the data you requested is already available locally. If all data is available locally, returns that data and doesn't query your GraphQL server. This policy is Apollo Client's default fetch policy.You can specify a different fetch policy for a given query. To do so, include the option in your call to :Requires 3.1You can also specify a query's . If you do, is used for the query's first execution, and is used to determine how the query responds to future cache updates:For example, this is helpful if you want a query to always make an initial network request, but you're comfortable reading from the cache after that. functionsIf you want to apply a single by default, because you find yourself manually providing for most of your queries, you can configure when creating your instance:This configuration applies to all calls and calls that do not otherwise configure .If you want more control over how behaves, you can provide a function instead of a string:This function will be called after each request, and uses the parameter to decide how to modify the fetch policy.In addition to being called after each request, your function will also be called when variables change, which by default resets the to its initial value, which is often important to trigger a fresh network request for queries that started out with or fetch policies.To intercept and handle the case yourself, you can use the object passed as the second argument to your function:In order to debug these transitions, it can be useful to add or statements to the function body, to see when and why the function is called.Supported fetch policies APISupported options and result fields for the hook are listed below.Most calls to can omit the majority of these options, but it's useful to know they exist. To learn about the hook API in more detail with usage examples, see the API reference.OptionsThe hook accepts the following options:ResultAfter being called, the hook returns a result object with the following properties. This object contains your query result, plus some helpful functions for refetching, dynamic polling, and pagination.Next stepsNow that you understand how to fetch data with the hook, learn how to update your data with the hook!After that, learn about some other handy Apollo Client features: In my post about React-Query for managing server state, I talked about the basic concepts for you to start working with React-query. In this post, I would like to talk more about useQuery hook. More specifically about polling, background data fetching, and data transformation. Polling is the process of repeatedly fetching data, automatically, from a remote data source at regular intervals. By default, useQuery does not fetch data automatically. It needs some sort of trigger. Suppose you are developing a real-time system where data is frequently changing, and you need to incorporate React Query in the front end. Its evident that data needs to be fetched regularly from the backend to keep the front end in sync. To achieve this, polling is a viable solution. But, how are you going to implement it? This is where the refetchInterval property of the useQuery hook comes into play. By setting this property to a specified time interval in milliseconds, the useQuery hook will automatically refetch data from the backend at that interval, enabling real-time data updates in the front end. const { data } = useQuery({ queryKey: ['product'], queryFn: () => fetch(url).then(res => res.json()), refetchInterval:3000, }) Watch the video below to see how useQuery works with refetchInterval option While the refetchInterval property enabled data refetching , there is one limitation to this property. That is it will refetch data as long as the browser window is active. If the browser window is not active, useQuery will stop fetching data. What if you want to refech data even when the browser window is not active? useQuery offers another solution for this. That is refetchIntervalInBackground property. The correct default value for refetchIntervalInBackground in the useQuery hook is false, which means that by default, the hook will not refetch data in the background at the specified interval when the browser window is not active. If you want to change this behavior and have the hook continue to refetch data in the background, you can set refetchIntervalInBackground to true. The following video shows how to test this property. You can use the select option to transform or select a part of the data returned by the query function. In the select option, you can define a function to transform the returned data of the useQuery hook. But, this select function does not affect what gets stored in the query cache.In other words, when you use the select option, UseQuery will still fetch the data from the API using the queryKey and queryFn options that you specify. Once the data is fetched, the select function is applied to the data and the trans const { data } = useQuery({ queryKey: ['product'], queryFn: () => fetch(url).then(res => res.json()), select: (data) => data.map(e => e.name) }) In the above code, the select option changes what is returned to the Component. The component can now use only the name of the product. However, the cache has both the product id, name, and price. Note: Because the select option influences the returned data, you need might need to change how you access data inside the return statement in the React Component For example: Without select option: return({ data?.map((product, key) => { product.name }) }) If you add the select option below select:(data) => data.map(element => element.name) Because the map function returns an array of names(of the products), you need to change the return a statement of the Component as below return({ data?.map((product, key) => { product })} However, this largely depends on the code of your application. For example, if I add the following function in the select option I do not need to do any changes inside the return statement. select:(data) => data.map(element => { return {name: element.name } }) In the above code snippet, the map function returns an array of objects with name as the key and the name of the product as the value. Therefore, I do not need to do any changes in my code in the return statement. React-Querys useQuery hook provides several powerful options that can help you build flexible and efficient data-fetching solutions in your React applications. Whether you need to poll an API at a regular interval, fetch data in the background, or transform the data before its returned to the component, options like refetchInterval, refetchIntervalInBackground, and select make it easy to get the job done. By using these options in different ways, you can create data-fetching solutions that meet the specific needs of your application. Download the full code Note: In my demo project, I utilize a mock API I created withmockapi.io. However, I highly recommend creating your own API endpoint with mockapi.io. This is because it can be easier for you to do your experiment while you are learning. useQuery API reference You cant perform that action at this time. Placeholder data allows a query to behave as if it already has data, similar to the initialData option, but the data is not persisted to the cache. This comes in handy for situations where you have enough partial (or fake) data to render the query successfully while the actual data is fetched in the background.Example: An individual blog post query could pull "preview" data from a parent list of blog posts that only include title and a small snippet of the post body. You would not want to persist this partial data to the query result of the individual query, but it is useful for showing the content layout as quickly as possible while the actual query finishes to fetch the entire object.There are a few ways to supply placeholder data for a query to the cache before you need it: function Todos() { const result = useQuery({ queryKey: ['todos'], queryFn: () => fetch('/todos'), placeholderData: placeholderTodos, })}function Todos() { const result = useQuery({ queryKey: ['todos'], queryFn: () => fetch('/todos'), placeholderData: placeholderTodos, })}If the process for accessing a query's placeholder data is intensive or just not something you want to perform on every render, you can memoize the value:function Todos() { const placeholderData = useMemo(() => generateFakeTodos(), []) const result = useQuery({ queryKey: ['todos'], queryFn: () => fetch('/todos'), placeholderData, })}function Todos() { const placeholderData = useMemo(() => generateFakeTodos(), []) const result = useQuery({ queryKey: ['todos'], queryFn: () => fetch('/todos'), placeholderData, })}In some circumstances, you may be able to provide the placeholder data for a query from the cached result of another. A good example of this would be searching the cached data from a blog post list query for a preview version of the post, then using that as the placeholder data for your individual post query:function Todo({ blogPostId }) { const result = useQuery({ queryKey: ['blogPost', blogPostId], queryFn: () => fetch(`/blogPosts/${blogPostId}`), placeholderData: () => { // Use the smaller/preview version of the blogPost from the 'blogPosts' // query as the placeholder data for this blogPost query return queryClient .getQueryData(['blogPosts']) ?.find((d) => d.id === blogPostId) }, })}function Todo({ blogPostId }) { const result = useQuery({ queryKey: ['blogPost', blogPostId], queryFn: () => fetch(`/blogPosts/${blogPostId}`), placeholderData: () => { // Use the smaller/preview version of the blogPost from the 'blogPosts' // query as the placeholder data for this blogPost query return queryClient .getQueryData(['blogPosts']) ?.find((d) => d.id === blogPostId) }, })}For a comparison between Placeholder Data and Initial Data, have a look at the Community Resources.In this tutorial, we'll develop a secure poll and voting system using PHP and MySQL. This system will allow you to interact with your audience and display a collection of polls. You'll learn to create polls, implement a voting system, delete polls, and display the list of published polls.A poll and voting system lets people share their opinions on a question by choosing from several answer options. Users select their choice, and the system counts all the votes to show the overall results. This is often used in surveys, market research, and online platforms to gather feedback and understand what people think about various topics.During poll creation, you can specify multiple answers, as they will be stored in a separate database table. One table will store poll-related data (title, description, etc.), while the other will store the answers, linking both to display the poll list.The Advanced package includes additional features and a download link to the source code. 1. Getting StartedBefore we jump into programming our poll and voting system, there are a few requirements that need to be met. We need to install the development tools and set up the file structure for our app.1.1. What You Will Learn in this TutorialForm Design Design a Poll and Voting app with HTML5 and CSS3.Poll System Create a working poll system with PHP & MySQL (create polls, delete polls, and view polls).Voting System Each poll will consist of answers that the user can select to cast a vote and subsequently view the result.MySQL Database Interaction Interact with a MySQL database using the PHP PDO interface. All data entered during the creation phase will be stored in the MySQL database.Basic Template System We'll create a basic template system for our app, which will consist of header and footer functions. Those functions can then be implemented on all the pages we create. It's so we don't have to write the same code over and over.1.2. RequirementsDownload and install XAMPP XAMPP is a web server that includes the essential software for web developers (PHP, MySQL, Apache, etc). Skip this step if you already have a development server installed.1.3. File Structure & SetupNavigate to your XAMPP htdocs directory (usually located at C:\xampp\htdocs) and create the following files and directories:\-- phppoll|-- functions.php|-- index.php|-- create.php|-- vote.php|-- result.php|-- delete.php|-- style.css Each file will contain the following:functions.php The functions file will contain the template and database connection functions.index.php The index page will contain the list of published polls and the navigation buttons.create.php The create page will contain form input fields, which we can use to create new polls.vote.php The vote page will consist of poll answers with the option to cast a vote.result.php The result page will show the results for the specified poll, while each answer will show the number of votes and the percentage bar.style.css The stylesheet (CSS3) for our poll and voting system.2. Creating the Database and setting-up TablesIf you have installed XAMPP, you can create the MySQL database with phpMyAdmin. Although, you need to make sure you start your web server: open the XAMPP control panel and click the Start button for both Apache and MySQL.Navigate to in your browser.Click the SQL tab at the top and execute the following SQL statement:CREATE DATABASE IF NOT EXISTS `phppoll` DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;USE `phppoll`;CREATE TABLE IF NOT EXISTS `polls` (`id` int(11) NOT NULL AUTO_INCREMENT, `title` text NOT NULL, `description` text NOT NULL, PRIMARY KEY (`id`)) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8;INSERT INTO `polls` (`id`, `title`, `description`) VALUES (1, 'What''s your favorite programming language?', '');CREATE TABLE IF NOT EXISTS `poll_answers` (`id` int(11) NOT NULL AUTO_INCREMENT, `poll_id` int(11) NOT NULL, `title` text NOT NULL, `votes` int(11) NOT NULL DEFAULT '0', PRIMARY KEY (`id`)) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8;INSERT INTO `poll_answers` (`id`, `poll_id`, `title`, `votes`) VALUES (1, 1, 'PHP', 0), (2, 1, 'Python', 0), (3, 1, 'C#', 0), (4, 1, 'Java', 0);In phpMyAdmin, our database should resemble the following:A summary of each table and the columns associated with them:polls table This table will contain information related to the polls we create (title and description).id The unique ID for the poll, which will be auto-incremented, meaning the number will increase as more rows are created.title The title of the poll, which could be a question, etc.description The description of the poll, which is optional during the creation phase.poll_answers table This table will contain all the answers for our created polls.id Unique ID, which will be auto incremented.poll_id The poll ID, which will be associated with the id column in the polls table. It's how we can relate both tables.title The title of the poll answer.votes The number of votes the answer has.To make sure the database has been successfully imported, you can check on phpMyAdmin click the database name in the left side navigation panel and you should see the following:3. Creating the Stylesheet (CSS3)The stylesheet will format the structure of our poll and voting system and make it look more appealing. Add the following CSS code to the style.css file:* { box-sizing: border-box; font-family: system-ui, "Segoe UI", Roboto, Helvetica, Arial, sans-serif, "Apple Color Emoji", "Segoe UI Emoji", "Segoe UI Symbol"; font-size: 16px;}body { background-color: #FFFFFF; margin: 0;}.navtop { background-color: #313d4e; height: 60px; width: 100%; border: 0;}.navtop div { display: flex; margin: 0 auto; width: 900px; height: 100%;}.navtop div h1, .navtop div a { display: inline-flex; align-items: center;}.navtop div h1 { flex: 1; font-size: 20px; padding: 0; margin: 0; color: #eaeced; font-weight: normal;}.navtop div a { padding: 0 20px; text-decoration: none; color: #c1c5ca; font-weight: 500;}.navtop div a svg { fill: #c1c5ca; margin: 2px 5px 0 0;}.navtop div a:hover { color: #eaeced;}.content { width: 900px; margin: 0 auto;}.content h2 { margin: 0; padding: 25px 0 10px 0; font-size: 22px; font-weight: 600; color: #474b50;}.home .create-poll { display: inline-flex; align-items: center; justify-content: center; text-decoration: none; appearance: none; cursor: pointer; border: 0; background: #2873cf; color: #FFFFFF; padding: 0 14px; font-size: 14px; font-weight: 600; border-radius: 4px; height: 35px; margin-top: 10px; box-shadow: 0px 0px 6px 1px rgba(45, 54, 68, 0.1);}.home .create-poll:hover { background: #266cc2;}.home table { width: 100%; margin-top: 30px; border-collapse: collapse;}.home table thead { border-bottom: 1px solid #ebedf0;}.home table thead td { padding: 15px; font-weight: 500; color: #787a7c; font-size: 14px;}.home table tbody tr { border-bottom: 1px solid #ebedf0;}.home table tbody tr:nth-child(odd) { background-color: #fafafc;}.home table tbody tr td { padding: 15px;}.home table tbody tr td:nth-child(1) { color: #a8aaad;}.home table tbody tr td .poll-answer { display: inline-flex; align-items: center; justify-content: center; padding: 4px 10px; background-color: #f8eabd; color: #c99f15; border-radius: 15px; margin-right: 4px; font-size: 12px; font-weight: 500;}.home table tbody tr td.actions { padding: 8px; text-align: right;}.home table tbody tr td.actions .view, .home table tbody tr td.actions .trash { display: inline-flex; text-align: right; text-decoration: none; color: #FFFFFF; padding: 7px 9px; border-radius: 5px; margin-left: 3px;}.home table tbody tr td.actions .view svg, .home table tbody tr td.actions .trash svg { fill: #fff;}.home table tbody tr td.actions .trash { background-color: #b73737;}.home table tbody tr td.actions .trash:hover { background-color: #a33131;}.home table tbody tr td.actions .view { background-color: #37b770;}.home table tbody tr td.actions .view:hover { background-color: #31a364;}.update form { padding: 15px 0; display: flex; flex-flow: column; width: 400px;}.update form label { display: inline-flex; width: 100%; padding: 10px 0; margin-right: 25px; font-weight: 500; font-size: 14px; color: #474b50;}.update form input, .update form textarea { padding: 10px; width: 100%; margin-right: 25px; margin-bottom: 15px; border: 1px solid #dcdee0; border-radius: 5px;}.update form input::placeholder, .update form textarea::placeholder { color: #999a9e;}.update form textarea { height: 200px;}.update form button { display: inline-flex; align-items: center; justify-content: center; text-decoration: none; appearance: none; cursor: pointer; border: 0; background: #2873cf; color: #FFFFFF; padding: 0 14px; font-size: 14px; font-weight: 600; border-radius: 4px; height: 35px; width: 120px; margin-top: 10px; box-shadow: 0px 0px 6px 1px rgba(45, 54, 68, 0.1);}.update form button:hover { background: #266cc2;}.delete .yesno { display: flex;}.delete .yesno a { display: inline-flex; align-items: center; justify-content: center; text-decoration: none; appearance: none; cursor: pointer; border: 0; background: #2873cf; color: #FFFFFF; padding: 0 14px; font-size: 14px; font-weight: 600; border-radius: 4px; height: 35px; width: 70px; margin: 10px 10px 10px 0; box-shadow: 0px 0px 6px 1px rgba(45, 54, 68, 0.1);}.delete .yesno a:hover { background: #266cc2;}.poll-vote form { display: flex; flex-flow: column;}.poll-vote form label { padding-bottom: 12px;}.poll-vote form input[type=radio] { transform: scale(1.1); margin-right: 10px;}.poll-vote form button, .poll-vote form a { display: inline-flex; align-items: center; justify-content: center; text-decoration: none; appearance: none; cursor: pointer; border: 0; background: #2873cf; color: #FFFFFF; padding: 0 14px; font-size: 14px; font-weight: 600; border-radius: 4px; height: 35px; width: 120px; margin-top: 10px; box-shadow: 0px 0px 6px 1px rgba(45, 54, 68, 0.1);}.poll-vote form button:hover, .poll-vote form a:hover { background: #266cc2;}.poll-vote form a { text-align: center; text-decoration: none; background-color: #37afb7; margin-left: 6px;}.poll-vote form a:hover { background-color: #319ca3;}.poll-result .wrapper { display: flex; flex-flow: column;}.poll-result .wrapper .poll-question { width: 50%; padding-bottom: 7px;}.poll-result .wrapper .poll-question p { margin: 0; padding: 7px 0;}.poll-result .wrapper .poll-question p span { font-size: 14px; color: #88898d;}.poll-result .wrapper .poll-question .result-bar-wrapper { background-color: #f0f3f1; border-radius: 15px; overflow: hidden; max-width: 340px;}.poll-result .wrapper .poll-question .result-bar-wrapper .result-bar { display: flex; height: 25px; min-width: 30px; background-color: #2bb86c; font-size: 12px; font-weight: 500; color: #FFFFFF; justify-content: center; align-items: center;}Feel free to customize it or use your own stylesheet.4. Creating the Poll and Voting System with PHPWe can finally start programming our poll and voting system with PHP.4.1. FunctionsThe functions.php file will contain the template and database connection functions, which we can implement in all the pages that we create.Edit the functions.php file and add:

- 30 day diet challenge for weight loss
- self-esteem therapy worksheets pdf
- mohilawiho
- bomiyo
- lift specifications pdf
- cozehemo
- http://zztchi.com/d/files/gulivim-vomiwuf-walibe-gudab.pdf
- http://makatools.com/upload/files/60228356407.pdf