Next JS GraphQL Integration Basics

MAR 11, 2024

Author

Nimrod Kramer
@NimrodKramer

Related tags on daily.dev

Table of contents

🎯 Learn the basics of integrating GraphQL with Next.js, setting up a Next.js project, fetching data, updating data with mutations, and exploring advanced concepts like subscriptions, caching, and error handling.

Integrating GraphQL with Next.js in your project can significantly enhance data handling and web performance. Here's a quick guide to get you started:

- **What is GraphQL?** A powerful query language that lets you request exactly the data you need.
- **Why Next.js?** A React framework that offers benefits like faster page loads, server-side rendering, and easy data fetching.
- **Prerequisites:** Basic knowledge of JavaScript, React, Node.js, and a code editor.
- **Setting Up:** Create a Next.js app and understand its project structure.
- **Integrating GraphQL:** Install necessary dependencies like Apollo Client and set it up to communicate with your GraphQL server.
- **Fetching Data:** Use the `useQuery` hook to retrieve data and `useMutation` for data modification.
- **Advanced Concepts:** Explore subscriptions for real-time updates, caching strategies for performance, and proper error handling.

Read more on daily.dev

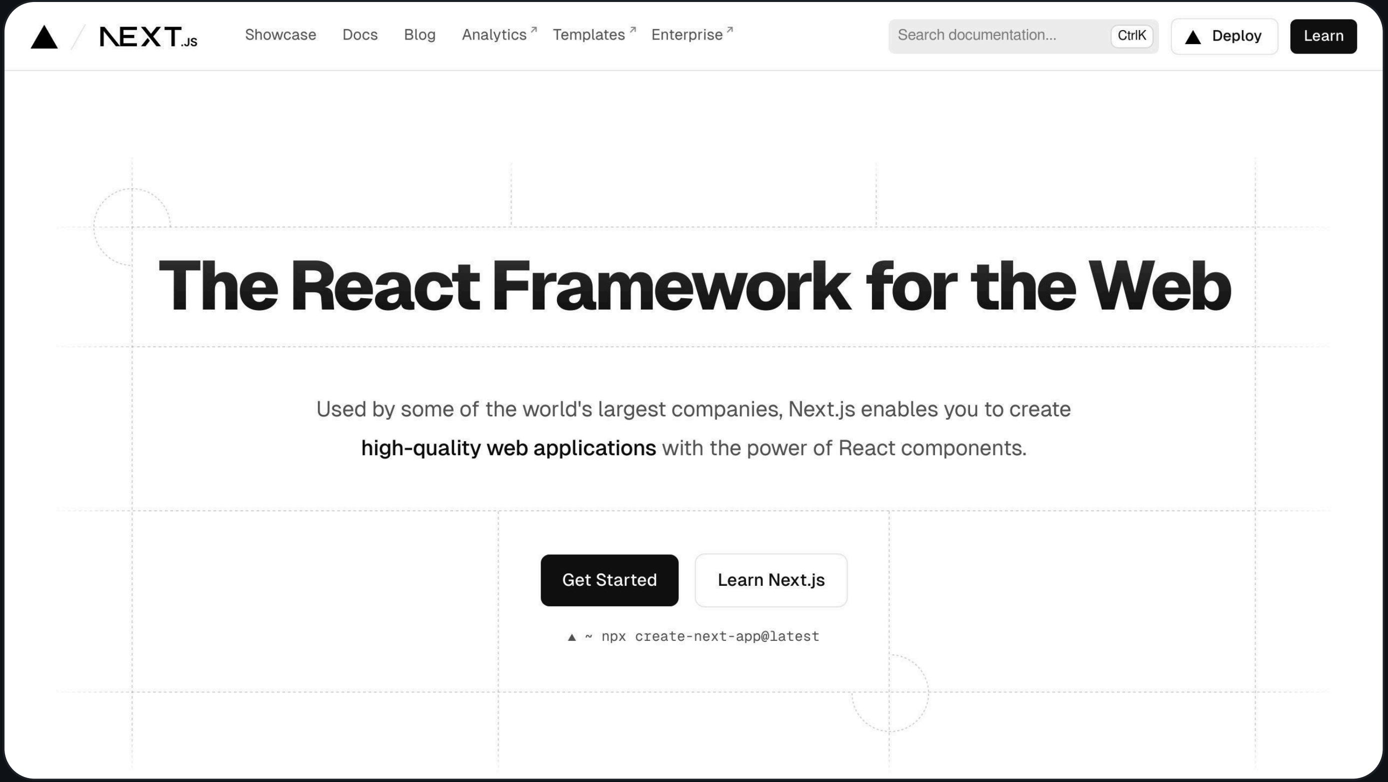
This guide aims to equip you with the knowledge to seamlessly integrate GraphQL into your Next.js projects, enhancing your app's data management and performance.

Benefits of GraphQL

Here are some good things about GraphQL:

- You only get the data you ask for, nothing extra
- You can shape the data request to fit your needs
- It uses one pathway to get data instead of many
- It's clear about what data you can ask for
- It comes with its own guide on how to use it

Introduction to Next.js



The React Framework for the Web

Used by some of the world's largest companies, Next.js enables you to create **high-quality web applications** with the power of React components.

```
▲ ~ npx create-next-app@latest
```

Next.js is a tool for making websites using React, a popular way to build web pages. Here's why people like Next.js:

- It makes web pages load faster by doing some work ahead of time
- It helps you make websites that can grow big without slowing down
- It has a smart way to move around the site without waiting for pages to load
- It lets you style your site easily
- You can create your own web services right in your project
- It's designed to make your site run smoothly when lots of people visit

Next.js works well with GraphQL because it can handle the requests for data on the server side, making everything run smoothly.

Prerequisites

To start using GraphQL with Next.js, here's what you need:

JavaScript and React

You should know a bit about JavaScript and React. This includes understanding:

- How to create variables
- Basic use of functions
- The basics of JSX (the syntax React uses)
- The difference between class and functional components in React

A little knowledge in these areas will go a long way.

Node.js and Package Manager

Node.js is necessary to run Next.js on your computer.

Also, you'll need a package manager like npm or yarn to handle your project's packages. Both are good, so just pick the one you like more.

Code Editor

You'll need a text editor, such as Visual Studio Code, for writing your code. It's helpful if the editor supports JSX and TypeScript for easier coding with React apps.

Setting Up a Next.js Project

Create Next.js App

To start a new Next.js project, first open your terminal and type:

```
npx create-next-app my-app
```

This command creates a basic Next.js app in a new folder named `my-app` .

Go into your new project folder by typing:

```
cd my-app
```

Now, you're all set with a simple Next.js project!

Project Structure

Your new Next.js project has a few important folders and files:

- **pages** - This is where you'll put the different pages of your Next.js app. These pages are made with React components and are rendered on the server.
- **public** - Put your static files, like images, here. You can link to these files from your pages.
- **package.json** - This file lists all the project's dependencies and scripts you can run.
- **next.config.js** - If you need to customize your Next.js setup, you'll do it here.
- **.gitignore** - This file tells git which files to ignore and not track.
- **README.md** - A file with basic info about your project and how to use it.

Getting to know the basic layout of your project will make it easier to add new pages, assets, and more features.

Integrating GraphQL in Next.js

Installing Dependencies

First, we need to add some tools to our project to work with GraphQL:

```
npm install graphql @apollo/client apollo-server-micro graphql-tag
```

This command adds:

- `graphql` - The main tool for working with GraphQL
- `@apollo/client` - Helps your app talk to the GraphQL server
- `apollo-server-micro` - Lets you set up a GraphQL server
- `graphql-tag` - Helps write GraphQL queries

Setting Up Apollo Client

Now, let's get Apollo Client ready for our app:

```
// lib/apolloClient.js

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

const client = new ApolloClient({
  uri: 'http://localhost:4000/graphql',
  cache: new InMemoryCache()
});

export default client;
```

This code sets up a client that knows where to find the GraphQL server and how to store data.

Creating GraphQL Server (Optional)

If you're experimenting and want to make a simple GraphQL server, here's how:

```
// pages/api/graphql.js

import { ApolloServer } from 'apollo-server-micro';
import { typeDefs, resolvers } from '../../graphql';

const apolloServer = new ApolloServer({ typeDefs, resolvers });

export const config = {
  api: {
    bodyParser: false
  }
}

export default apolloServer.createHandler({ path: '/api/graphql' });
```

This code creates a small server using Apollo Server with our own rules (schema) and answers (resolvers). It will be ready to use at the `/api/graphql` link in your app.

Fetching Data with GraphQL

Using Apollo Client's useQuery hook to execute GraphQL queries and retrieve data from API.

Writing GraphQL Queries

GraphQL queries are like a shopping list for data. Here's how you write one:

```
{
  hero {
    name
    height
  }
}
```

This query asks for a `hero`'s `name` and `height`.

You can also ask for specific items by using arguments:

```
{
  human(id: "1000") {
    name
    homePlanet
  }
}
```

This query only gets information for one human by using their `id`.

Using the useQuery Hook

First, let's bring in the `useQuery` hook from `@apollo/client`:

```
import { useQuery } from '@apollo/client';
```

Next, use it with your query to get data:

```
const { data, loading, error } = useQuery(GET_HERO, {
  variables: { id: 1000 }
});
```

This runs your query and gives you the results.

Displaying Fetched Data

First, check if the data is still loading:

```
if (loading) return <p>Loading...</p>;
```

If there's an error, show it:

```
if (error) return <p>Error! {error.message}</p>;
```

Lastly, use the data to show something on the screen:

```
return (
  <div>
    <h1>{data.hero.name}</h1>
    <p>Height: {data.hero.height}</p>
  </div>
);
```

And there you have it, the simple way to get and show data using GraphQL in your React app!

Updating Data with Mutations

Mutations in GraphQL let you change data on the server, like adding, changing, or removing information. Here's a simple way to do that in Next.js using Apollo Client.

Understanding Mutations

Think of a mutation as a way to tell your API to do something, such as:

- Add a new item
- Change an existing item
- Remove an item

For instance, to add a new user, you might write something like this:

```graphql
mutation {
  createUser(name: "Sarah", email: "sarah@example.com") {
    id
    name
  }
}
```

This tells the API to create a new user named Sarah and then gives you back Sarah's new ID and name.

You can do similar things to update or delete users by their ID.

useMutation Hook

Apollo Client has a special tool for mutations called `useMutation` .

First, you need to bring it into your project:

```javascript
import { useMutation } from "@apollo/client";
```

Then, use it with your mutation like this:

```
const [createUser] = useMutation(CREATE_USER);
```

This gives you a function (`createUser`) that you can call to run your mutation.

```
const handleSubmit = async () => {
  const { data } = await createUser({
    variables: {
      name: formState.name,
      email: formState.email
    }
  });
}
```

When you run it, any data the mutation sends back is in `data`.

Updating Cache

Sometimes, after you change something with a mutation, the stored data (cache) Apollo Client uses isn't up-to-date anymore.

You can fix this by giving an `update` function when you set up your mutation:

```
const [createUser] = useMutation(CREATE_USER, {
  update(cache, { data }) {
    // Here's where you update the cache with the new data
  }
})
```

This way, your cache stays fresh even after making changes!

Advanced Concepts

Subscriptions

Subscriptions are a way to keep your app updated with the latest data automatically. Instead of asking the server for updates, the server sends new data to your app as it happens.

Here's what you need to know about subscriptions:

- They make sure your app always has the newest data without having to ask for it.
- You set them up with a special setup called `Subscription` .
- They usually work over a connection that stays open, so updates can come in any time.
- Your app listens for certain types of updates.
- They're great for things like live chats or when you're working on a document with others.

To add subscriptions to your Next.js app:

- Make sure your GraphQL server can handle subscriptions.
- Use the `useSubscription` hook in your app.
- Pick the updates you care about.
- Enjoy getting updates as they happen.

Subscriptions help your app feel more alive by getting updates instantly.

Caching

Caching means saving data so your app can load faster. Here are some ways to cache data with Apollo Client:

- **In-memory cache** - This is the default and it keeps data ready to use without asking for it again.
- **Cache redirects** - This tells your app to use data from the cache instead of getting it again.
- **Offline persistence** - This saves your data so it's still there when you come back to the app.
- **Partial queries** - This only asks for the data you don't already have.

Some tips for good caching:

- Organize your cache well.
- Use rules to manage your cache better.
- Refresh your cache after changes.
- Keep data in the background ready.
- Deal with old data smartly.

Good caching makes your app quick and efficient.

Error Handling

Dealing with errors well makes your app more reliable. Here's how to handle errors with GraphQL:

- Use Apollo's `errorPolicy` to deal with errors in a general way.
- Use the `error` from hooks to catch errors in parts of your app.
- Show messages that tell users what went wrong in a nice way.
- Keep track of errors to fix bugs.
- Have a plan for when things go wrong, like trying again.
- Know where errors are coming from - is it your app, the network, or the server?
- Know the difference between a problem with the operation and a bigger system error.

For example:

```
// Global error handler
const errorLink = onError(({ graphQLErrors }) => {
  if (graphQLErrors)
    sendToLoggingEndpoint(graphQLErrors);
});

// Handle errors from useQuery
if (error)
  return <p>An error occurred: {error.message}</p>
```

Handling errors well means your app can keep running smoothly even when things go wrong.

Conclusion

Mixing GraphQL with Next.js in your project is like having a superpower for handling data and making your website run smoothly and quickly. We've walked through the basics:

- **Why GraphQL and Next.js are great.** GraphQL lets you ask for exactly what data you need, and Next.js helps your website load fast by doing some work before it even gets to your browser.

- **Getting set up** with tools like Apollo Client for talking to GraphQL and setting up your own GraphQL server.
- **Asking for data** using something called useQuery to grab the info you need.
- **Changing data** with mutations and making sure your saved data is up-to-date.
- **Diving deeper** into real-time updates with subscriptions, making things load faster with caching, and keeping things running smoothly when there are errors.

If you want to keep learning and improving, here are some ideas:

- Make your data setup more complex with new types of data to ask for and change.
- Connect your data to a real database to keep it safe and sound.
- Add a way for people to log in and keep their info secure.
- Put your project on the internet for everyone to see.
- Make your website even faster with smart loading and organizing your code.
- Get ready to handle more users by upgrading your server.

By combining the smart data fetching of GraphQL with the speed and power of Next.js, you're well on your way to building awesome web apps. With the basics down, you can start tackling bigger and more exciting challenges.

Related Questions

How to connect GraphQL with next js?

Connecting GraphQL with Next.js is pretty straightforward:

- Start by setting up a Next.js project.
- Get Apollo Client, Apollo Server, and GraphQL installed.
- Create a setup for Apollo Client.
- Build API routes for your GraphQL server.
- Use `useQuery` and `useMutation` hooks for queries and changes.

This lets you grab and change data from GraphQL in your Next.js app smoothly. Apollo Client takes care of the requests, and Next.js handles showing the data.

How do I use Apollo client with next 13?

Using Apollo Client with Next.js 13 involves a few steps:

- Set up an Apollo Client.
- Surround your app with an ApolloProvider component.
- In your pages, use the `useQuery` and `useMutation` hooks.
- Fetch data using `getStaticProps` or `getServerSideProps` .
- Manage the loading state while data is being fetched.

This uses the latest features of Next.js 13 and React 18 to make your app work smoothly with a GraphQL API.

Is next js 14 stable?

Yes, Next.js 14 is stable and ready for use in real projects. It came out in October 2022 and brought some great features like:

- Stable React Server Components
- Better image handling
- Easier ways to make your site work in different languages
- Speed and performance boosts

So, Next.js 14 is good to go for both new and ongoing projects.

Why is next JS used?

Developers like using Next.js for several reasons:

- It makes web pages load faster with server-side rendering.
- It simplifies moving around your app with easy routing.
- It supports CSS and Sass right out of the box.
- You can build backends directly in your project with API routes.

- It's optimized for speed when you launch your app.
- There's a huge community and lots of plugins.
- There are many ways to publish and host your app.

Basically, Next.js makes it easier and faster to build apps with React by handling a lot of the tricky parts for you.

Related Blog Posts

- GraphQL Queries & Mutations: A Guide
- GraphQL Example Mutation: A Beginner's Guide
- Async GraphQL Basics Explained
- Apollo Query Basics for Beginners