TASK

Next.js II: Fetching Data

Visit our website

Introduction

WELCOME TO THE NEXT.JS FETCHING DATA TASK!

In this task, you will learn to use Next.js to Fetch data. You will also learn how to customise your Next.js applications more.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact an expert code reviewer to get support on any aspect of your course.

The best way to get help is to login to Discord at **https://discord.com/invite/hyperdev** where our specialist team is ready to support you.

Our team is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

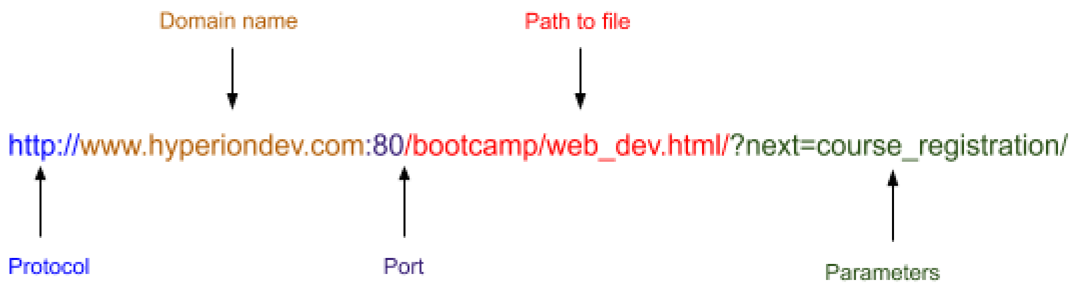
CREATING DYNAMIC WEB PAGES WITH NEXT.JS

A dynamic web page is a page that displays content dynamically. To create web pages that display content dynamically we typically need to do at least two things:

1. Pass data through from one page to another. In this task, you will use query strings to do this.
2. Fetch data from somewhere. The data that are going to be used to populate the page need to be stored somewhere.

PASSING DATA USING QUERY STRINGS

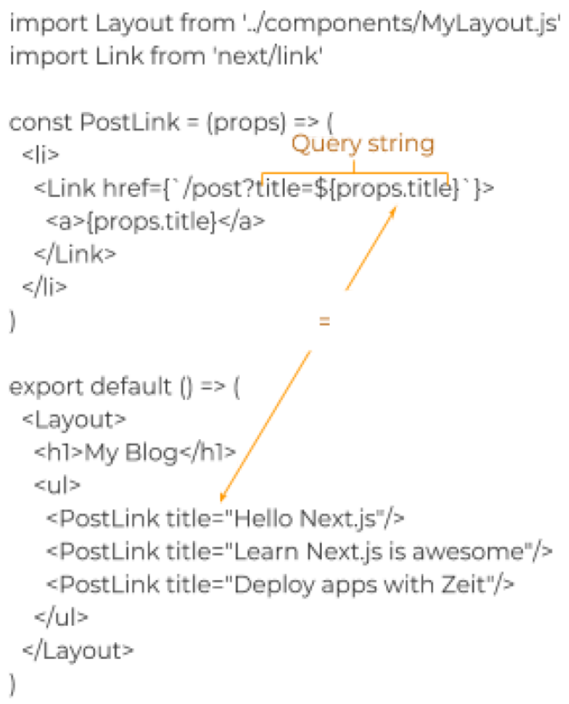
As you know, a URL (Uniform Resource Locator) is used to find certain resources on the web. Consider the following fictional URL:


Domain name
Path to file
http://www.hyperiondev.com:80/bootcamp/web_dev.html/?next=course_registration/
Protocol
Port
Parameters

For this task, we are interested in the last part of the URL. We can append bits of information to a URL. The section of the URL that contains the parameters we want to pass is sometimes referred to as a ***query string***. All the data after the question mark (?) is part of the query string.

For the rest of the task, we will be using the example of a blog to demonstrate how to pass info using a query string. Open the code in 'Example 1' for this task to follow along.

Notice how easy it is to add a query string as demonstrated in the code below. The query string contains a '?' symbol followed by a key (used to identify the data we are passing), then an '=' symbol, and finally the value (or data) we want to pass.



```
import Layout from '../components/MyLayout.js'
import Link from 'next/link'

const PostLink = (props) => (
  <li>
    <Link href={`/post?title=${props.title}`}>
      <a>{props.title}</a>
    </Link>
  </li>
)

export default () => (
  <Layout>
    <h1>My Blog</h1>
    <ul>
      <PostLink title="Hello Next.js"/>
      <PostLink title="Learn Next.js is awesome"/>
      <PostLink title="Deploy apps with Zeit"/>
    </ul>
  </Layout>
)
```

To get this information from the query string on a different page we use `withRouter` as shown below:

```
import {withRouter} from 'next/router'
import Layout from '../components/MyLayout.js'

const Page = withRouter((props) => (
   <Layout>
      <h1>{props.router.query.title}</h1>
      <p>This is the blog post content.</p>
   </Layout>
))

export default Page
```

`withRouter` is a React-Router higher-order component. According to **React's documentation**, "a higher-order component is a function that takes a component and returns a new component. Whereas a component transforms props into UI, a higher-order component transforms a component into another component."

You can use `withRouter` to get access to the closest `<Route>`'s match. A **match object** contains information about how a `<Route path>` matched the URL. It includes information about the param, path, and URL of the route. `withRouter` will pass updated match, location, and history props to the wrapped component whenever it renders.

FETCHING DATA

To create dynamic web pages we usually need to fetch data from a remote data source. Next.js comes with a standard API to fetch data for pages. To use this API, do the following:

1. To install the library we need to fetch data, install isomorphic-unfetch by typing the following in your command line interface: `npm install --save isomorphic-unfetch`

2. Import this library in the code where you want to use it: `import fetch from 'isomorphic-unfetch'`

3. Fetch the data using an async function called `getInitialProps` as shown below:

```
Index.getInitialProps = async function() {
   const res = await
fetch('https://api.tvmaze.com/search/shows?q=batman')
   const data = await res.json()

   console.log(`Show data fetched. Count: ${data.length}`)

   return {
      shows: data
   }
}
```

`getInitialProps` can asynchronously fetch anything that resolves to a JavaScript plain Object, which populates props. Data returned from getInitialProps is serialised when server rendering.

Things to remember when using `getInitialProps` :

- Make sure the returned object from `getInitialProps` is a plain Object and is not using Date, Map, or Set.
- For the initial page load, `getInitialProps` will execute on the server only. `getInitialProps` will only be executed on the client when navigating to a different route via the `Link` component or using the routing APIs.
- `getInitialProps` cannot be used in child components, only in pages.

In the code above we create an async function that will fetch the data we need from the API we specify ('`https://api.tvmaze.com/search/shows?q=batman`'). The instruction `res.json` then sends a JSON response composed of a stringified version of the specified data. That data is then sent as `props` for the page.

4. Below is an example of how the data that we have fetched can then be used to dynamically display information on our page:

```
const Index = (props) => (
   <Layout>
      <h1>Batman TV Shows</h1>
      <ul>
         {props.shows.map(({show}) => (
            <li key={show.id}>
               <Link as={`/p/${show.id}`}
href={`/post?id=${show.id}`}>
                  <a>{show.name}</a>
               </Link>
            </li>
         ))}
      </ul>
   </Layout>
)
```

Take note:

The library we use to fetch data (isomorphic-unfetch) is a simple implementation of the browser fetch API but works **in both client and server environments**.

CUSTOMISING YOUR NEXT.JS APP

As you have seen, Next.js by default hides or abstracts some of the code for the apps created. For example, unlike applications created with Create React App, applications created with Next.js do not by default expose the functionality of the App component. However, it is possible to customise default Next.js apps. This section highlights two important customisations.

Customised App Component

Next.js uses the App component to initialise pages. If you need to inject additional data into pages (for example by processing GraphQL queries), keep state when navigating pages or persist layout between page changes, you can override the

App component and control the page initialisation. To override, create the **./pages/_app.js** file and override the App class. Since you are familiar with working with the App component from the work you did with Create React App, we will not cover how to customise this component in detail here. See this **Next.js documentation** for more information.

Customised Error Handling

With Next.js, 404 or 500 errors are handled both client- and server-side by a default component **error.js**. To override this component, create a file called **_error.js** in the **pages** directory. The **pages/_error.js** component is only used in production. In development, you get an error with a call stack to know where the error originated from. For an example of how to override **_error.js**, see **here**.

TOOLS THAT POWER REACT APP AND NEXT.JS

As you know, we use frameworks such as Create React App and Next.js to speed up development. These frameworks rely on very important libraries and modules. Although the details of how to use these tools is beyond the scope of this Bootcamp, you should know something about two of them, Webpack and Babel.

When you create apps using React, you are more than likely going to use and create many modules and other resources like images and CSS files. Webpack is a module bundler that creates a dependency graph that it uses to handle all the resources you need for your app. Learn more about Webpack **here**.

Babel is a JavaScript compiler. Babel is particularly relevant to developers who work with React because it can convert JSX syntax to JavaScript. It can also be used to convert ES2015 and newer versions of JavaScript (next-generation JavaScript) to ES5 for backward compatibility. Check Babel out **here**.

For instructions on how to configure Babel and Webpack for your Next.js applications, see **here**.

SPOT CHECK 1

Let's see what you can remember from this section.

1. What does a query string contain in a URL?
2. What does `withRouter` do?
3. What 3 things do you need to remember when using `getInitialProps`?

Instructions

- Copy all the directories that contain example code that accompany this task to your local computer and then decompress them. Follow the instructions in the readme.md files before attempting this compulsory task.
- This task involves creating apps that need some modules to run. These modules are located in a folder called 'node_modules'. Please note that this folder typically contains hundreds of files which, if you're working directly from Dropbox, has the potential to slow down Dropbox sync and possibly your computer. As a result, please follow this process when creating/running such apps:
  - Create the app on your local machine (outside of Dropbox) by following the instructions in the compulsory task.
  - When you're ready to have a code reviewer review the app, please delete the node_modules folder .
  - Compress the folder and upload it to Dropbox.

  Your code reviewer will, in turn, decompress the folder, install the necessary modules and run the app from their local machine.

Compulsory Task 1

Follow these steps:

- Create a new project using Next.js. This should be a dynamic website that is created using data retrieved from an API. You can use a simple API of your choice. Here are a few suggestions:
  - **Musixmatch API**
  - **The Star Wars API**
  - **Spotify API**

- The web app should contain at least two pages: an index page and a page that displays details about the topic that the user selects on the index page. For example, if you create your app using the Star Wars API you could display all the films (e.g. https://swapi.dev/api/films) on the index page and all the details about the specific film selected by the user on the details page (e.g. https://swapi.dev/api/films/1).

- Be sure that your website is attractively styled and that you share a **layout component**.

If you are having any difficulties, please feel free to contact our specialist team **on Discord** for support.

Optional Bonus Task

Follow these steps:

- Create your own blog. You should be able to log in to your blog application to make a post, but your blog posts should be available for all to see.

Completed the task(s)?

Ask an expert to review your work!

Review work

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved, or think we've done a good job?

Click here to share your thoughts anonymously.

SPOT CHECK 1 ANSWERS

1. The query string contains a ‘?’ symbol followed by a key (used to identify the data we are passing) and then a ‘=’ symbol and finally the value (or data) we want to pass.
2. You can use `withRouter` to get access to the closest `<Route>`'s match. A match object contains information about how a `<Route path>` matched the URL. It includes information about the param, path, and URL of the route. `withRouter` will pass updated match, location, and history props to the wrapped component whenever it renders.
3.
    a. Make sure the returned object from `getInitialProps` is a plain Object and not using Date, Map, or Set.
    b. For the initial page load, `getInitialProps` will execute on the server only. `getInitialProps` will only be executed on the client when navigating to a different route via the `Link` component or using the routing APIs.
    c. `getInitialProps` cannot be used in child components, only in pages.