TASK

Next.js I

Visit our website

Introduction

WELCOME TO THE INTRODUCTION TO NEXT.JS TASK!

Next.js is the final new framework that you will learn in this Bootcamp! In this task, you will learn what Next.js is, and why it is beneficial to learn to use it. By the end of the task, you will be able to use Next.js to create your online developer portfolio.

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.

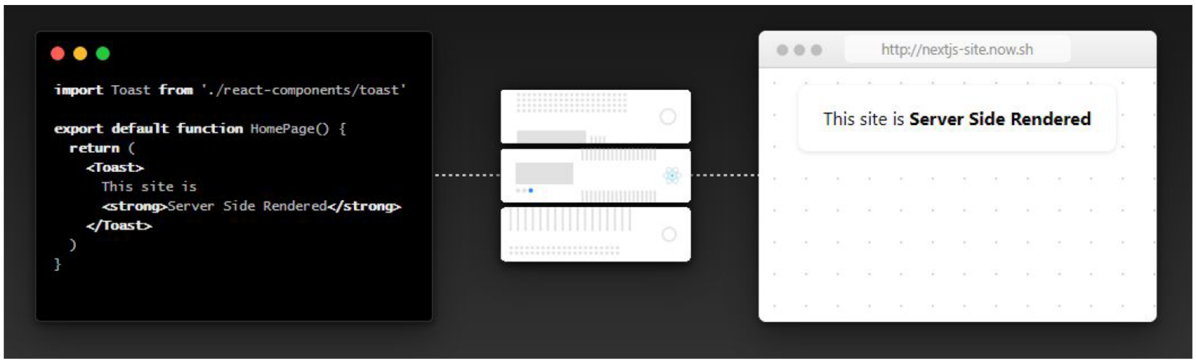
The best way to get help is to login to Discord at **https://discord.com/invite/hyperdev** where our specialist team is ready to support you.

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!

WHAT IS NEXT.JS?

Like Create React App, Next.js is a React Framework that allows us to build web applications using JavaScript and React more quickly. Whereas Create React App is good to use for creating single-page applications, Next.js is better to use when you want to create static and server-rendered applications. What is the difference between a single-page app and a server-rendered app? A single-page app is created when all the resources needed to make the page run (like the HTML, CSS, and JavaScript) are loaded at once. When the user interacts with the page there is no interaction with the server.

With a server-rendered application, JavaScript is executed on the server and the resulting HTML that is rendered is then sent to the browser.



```
import Toast from './react-components/toast'

export default function HomePage() {
  return (
    <Toast>
      This site is
      <strong>Server Side Rendered</strong>
    </Toast>
  )
}
```

http://nextjs-site.now.sh
This site is **Server Side Rendered**

Some of the benefits of server-rendered applications include:

1. The fact that JavaScript is executed on the server rather than on the browser can lead to faster initial page rendering. Slower clients may not be able to process JavaScript as quickly as servers. Prefetching initial data and building pages on the server also reduces the number of round trips required to view your site.

2. Server-rendered applications lead to better search engine optimisation (SEO). A server-rendered application returns HTML instead of JavaScript. This means that pages returned with a server-rendered application are more easily indexable by search engines and previewable on social media platforms.

Extra resource

Watch this **short video** for an overview of what server-side rendering is.

CREATE AN APP WITH NEXT.JS

As you have done with Create React App, you are going to use the command line interface and npm to create a React App using Next.js. To create an app, complete the following steps:

1. Open your command-line interface and create a project directory with a meaningful name. E.g. `mkdir my-first-next-app`

2. Navigate to the directory you have just created. E.g. `cd my-first-next-app`

3. Create a package.json file by typing `npm init`

4. Create a sample project by typing `npm install --save react react-dom next`. You will see that this creates your node_modules directory for your project.

5. Create a directory called “pages” (`mkdir pages`). It is important that you call this directory “pages”.

   With Next.js you don’t have to write all your routes from scratch. Next.js does *file-system routing*. It will serve each file in a directory called ‘pages’ under a pathname matching the filename. For example, ‘/pages/about.js’ is served at http://your_sites_name.com/about.

   You are also able to customise routes. To do this, you create a custom server. For instructions on how to do this, see **here**.

6. Add a JavaScript file called **index.js** to the “pages” directory. In this file, create a React component as you learnt in previous tasks. For example:

```
function Welcome(props) {
   return <h1>Hello, I can use Next!!!</h1>;
}
export default Welcome;
```

7. Add another JavaScript file called **about.js** to the "pages" directory. Add the following code[1]:

```
export default () => (
   <div>
     <p>This is the about page</p>
   </div>
)
```

8. Open the package.json file and add the following:

```
{
   "scripts": {
      "dev": "next",
      "build": "next build",
      "start": "next start"
   }
}
```

9. Run the development server by typing `npm run dev`

10. Then open **http://localhost:3000**. Also, navigate to **http://localhost:3000/about**

Take note:

Arrow function reminder: With ES6 arrow functions are used. An example of an arrow function is shown below. Arrow functions have two key advantages over traditional Functions:

- You use less code to do the same thing
- There is no separate `this` object; `this` has the same value as the enclosing function.

[1] See the note from the HyperionDev team below to explain the syntax used for creating this component.

Notice below how we could rewrite the code in the instructions above using an arrow function.

ES6	Previous version of ES

`const Welcome = () => (`
`   <h1>Hello, I can use Next!!!</h1>`
`)`
 	
`export default Welcome;`
	`function Welcome() {`
	`       return  <h1>Hello,  I  can  use`
	`Next!!!</h1>;`
	`}`
	
	`export default Welcome;`

The basic syntax for an arrow function is shown below. In the example above, there are no parameters for the arrow function, therefore, empty parentheses are used.

```
(param1, param2, …, paramN) => { statements }
```

To learn more about using arrow functions, see **here**.

We could further abbreviate the code needed to export a React component to:

```
export default () => (
   <h1>Hello, I can use Next!!!</h1>
)
```

ADDING LINKS TO PAGES

Since Next.js is used to create server-rendered pages, adding a link using a normal `<a>` tag would cause a link that would be navigated to via the server. This is obviously not always necessary. To support client-side navigation (this takes place in the browser, without making a request to the server) we use **Next.js' Link API**. We do so as shown in the code example below:

```
//Import the Link API
import Link from 'next/Link'

export default() => (
    <div>
        // Use this to create a link that supports client-side navigation
        <Link href="/index">
        <a>Home</a>
        </Link>
        <p>This is the About page</p>
    </div>
)
```

ADDING IMAGES AND OTHER STATIC RESOURCES

To add static resources like images to your app you should create a folder called “static” in your project root directory. This directory MUST be called “static” and is the only directory that Next.js uses for serving static assets. From your code, you can then simply reference those files with /static/ URLs. For example:

```
export default () => <img src="/static/my-image.png" alt="my image" />
```

STYLING NEXT.JS APPS

Although it is possible to use traditional file-based styling, it is recommended that for Next.js apps, you use CSS in JS styling.

Definition of CSS in JS: According to **React** CSS-in-JS “refers to a pattern where CSS is composed using JavaScript instead of defined in external files.”

Next.js comes preloaded with a CSS in JS framework called **styled-jsx**. An example of how styled-jsx is used for writing style rules is shown in the example below:

```
export default () => (
   <div>
      Hello world
      <p>scoped!</p>
      <style jsx>{`
         p {
            color: blue;
          }
         div {
            background: red;
         }
         @media (max-width: 600px) {
            div {
               background: blue;
            }
         }
      `}</style>
      <style global jsx>{`
```

```
          body {
             background: black;
          }
       `}</style>
    </div>
)
```

Notice the following in the code example above:

- The style rules are written within `<style jsx>` and `<style global jsx>` tags. Rules written within `<style jsx>` tags have no effect on elements inside of a child component. If you need to change styles inside of a child component, use `<style global jsx>`. For more information about this, see **here**.

- Within the `<style jsx>` and `<style global jsx>` tags all style rules are written within a ***template string*** .

- If you would like to use similar styles across many pages you can create a common ***Layout component*** and use it for each of your pages. See the example code that accompanies this task to see how this is done.

Take note:

ES6 update: Instead of using concatenation to modify strings, ES6 uses template literals/template strings. To use template literals:

- Enclose strings with `` ` ` `` instead of ‘’ or “”
- Use `${variable_name}` instead of `+ variable_name` (see example below)
- Don't bother with `\n`, these are automatically preserved with template literals.

```
const myName = 'Sue';
const greeting = `Hello, my name is ${myName}

   I LOVE JavaScript!!!
   One of my favourite things about ES6 is that it isn't fussy.
    I didn't have to escape everything in this string to get it to look good
:) `;
console.log(greeting);
```

SPOT CHECK 1

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

1. What is the difference between a single-page app and a server-rendered app?
2. Fill in the missing pieces of code to create a link when using Next.js:

```
import ____ from _____

__________ => (
    <div>
        // Use this to create a link that supports client-side
navigation
        <Link href="/index">
        <a>Home</a>
        </Link>
        <p>This is the About page</p>
    </div>
)
```

3. What are the tags used when creating style components with Next.js?

Instructions

- Copy the 'nextExample' directory that accompanies this task to your local computer and then decompress it. Follow the instructions in the readme.md file 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 reviewer review the app, please delete the node_modules folder.
  - Compress the folder and upload it to Dropbox.

  Your 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 Developer Portfolio using Next.js. Your developer portfolio should contain the following pages/components:

    - About - this page will serve as an introduction to you as a developer – your educational history, your work history, who you are and what you're passionate about.

    - Projects - this page needs to contain your best work. You will want to show off diversity as much as possible here. You should have at least one project deployed to Heroku which you should be able to link to here (your previous Capstone Project). It may be a good idea to add some of your work to Github and link to those repositories from here.

    - Contact - ultimately what you want is for a lead to see your portfolio and contact you so you should include your contact details (email address, telephone number, etc).

- Besides the requirements listed above, your project should also adhere to the following criteria:
    - It should be styled using styled-jsx.
    - It should contain at least one static image.
    - It should contain a header component that the user can use to navigate to the different pages in your app.

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

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. A single-page app is created when all the resources needed to make the page run (like the HTML, CSS and JavaScript) are loaded at once. When the user interacts with the page there is no interaction with the server. With a server-rendered application, JavaScript is executed on the server and the resulting HTML that is rendered is then sent to the browser.

2.

```
//Import the Link API
import Link from 'next/Link'

export default() => (
    <div>
        // Use this to create a link that supports client-side navigation
        <Link href="/index">
        <a>Home</a>
        </Link>
        <p>This is the About page</p>
    </div>
)
```

3. `<style jsx>` and `<style global jsx>`